1993

An Intelligent tutoring system for the German language

Kenneth E. Staffan

Title of Thesis: An Intelligent Tutoring System for the German Language

I, Kenneth E. Staffan, prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

100 Indigo Creek Drive
Rochester, New York 14650

or electronically at:

staffan@serum.kodak.com

date 10-May-93

Acknowledgements

All the thanks in the world go to the following people: Kevin Donaghy and Hank Etlinger, for their advice and guidance throughout the project, my wife, Mary, for bearing an inordinate amount of the home and family burden while I worked, and for supplying the original Intelligent Tutoring System concept, and last, but by no means least, my young son, Ben, for being the bright spot at the end of each long day of work and school. May he develop a thirst for learning, himself.

Abstract

This thesis report describes the design and implementation of a prototype Intelligent Tutoring System (ITS), intended to assist students of the German language. Very early in the study of a foreign language, the student is faced with the difficulties of sentence construction. Not only are there numerous rules and combinations to deal with, but it is difficult to verify attempts when the teacher is unavailable. Individual words can be looked up in a dictionary, but the student must often rely on stumbling across a sentence of similar construction in order to verify a trial sentence.

A variety of Computer-Assisted Language Learning (CALL) tools have been developed. Many have been criticized for not being user friendly, containing material which does not match the course curriculum, being inflexible, or being just plain incorrect. The prototype system developed for this thesis experiments with several characteristics - an object-oriented design approach, a masking technique using dynamically built patterns to bridge the gap between hard-coded and full artificial intelligence approaches, and a C++ implementation. It attempts to draw on past failures, as well as past successes.

The system described here provides a means for practicing sentence construction, with interactive diagnosis and feedback. A phrase or sentence is presented to the student for translation. The response is then checked for correctness. If the answer is incorrect, the student is given the option of trying again, receiving increasingly more specific hints, or having the system display the response it was expecting.

Key Words and Phrases

Intelligent Tutoring Systems (ITS), Computer–Assisted Instruction (CAI), Computer–Assisted Language Learning (CALL), Expert Systems, Natural Language Processing, Education, German, C++.

Subject Codes

ACM Computing Review Codes

D.1.5 Software
Programming Techniques
Object–Oriented Programming

I.2.1 Computing Methodologies
Artificial Intelligence
Applications and Expert Systems
I.2.7 Natural Language Processing

K.3.1 Computing Milieux
Computers and Education
Computer Uses in Education

Inspec (IEEE/IEE) Computer and Control Abstracts Codes

1230 Artificial Intelligence

6170 Expert Systems

6180N Natural Language Processing

7110 Education

1. INTRODUCTION AND BACKGROUND

1.1. Problem Statement

When a foreign language student leaves the classroom environment and teacher, a valuable resource is lost. The student no longer has an expert to use as a resource for understanding or clarification. Text books and other "static" media are not always effective reference (or tutoring) tools. A dictionary, for instance, can be used if a student wishes to look up an individual word, but may be of little help if the question concerns sentence construction or other concepts. Intelligent Tutoring Systems (ITS), and other forms of Computer–Assisted Instruction (CAI), have attempted to extend the resources available to a student for practice and reference.

Computer–Assisted Instruction has been a fairly popular research area, in both the academic and commercial worlds, since the early 1960's. A variety of student– and teacher–oriented products are currently on the market or in experimental usage. Questions, however, about the lack of effectiveness of these products, combined with the apparent lack of real progress in the field, are what prompted one researcher to speculate on whether the field might better be called CIA – Computer–Inhibited Acquisition [Farrington,1986]. Part of the difficulty has been in the cross–disciplinary nature of the task. Not just a computer or parsing exercise, CAI researchers must take into account accepted methods in the fields of education, psychology, and, of course, the subject which is to be taught.

This thesis project addresses one area in which an intelligent tutoring system can be of assistance – the area of phrase and sentence construction. This is an area where resources are scant, when no expert is accessible. To verify the correctness of a sentence, the student may have to rely on stumbling across one of similar structure in a text book. Even this may be inaccurate, as different vocabulary may require a variation in the structure of the sentence. Sentence construction is not an advanced topic, either. Most language courses begin introducing some sentence forms with the very first class. Already, the best CAI packages are challenged.

The ITS prototyped as part of this thesis experiments with a masking technique of diagnosis, using run–time generated patterns to identify the discrepancies in a student response. The thesis also experiments with the object–oriented analysis, design and implementation of such a system. A fully functional (though limited in knowledge depth) system has been designed and implemented to illustrate and exercise these concepts. The system presents phrases to the student for translation, and diagnoses the responses. Hints and additional chances are offered if the student desires.

1.2. Previous Work

In the early 1960's, there was not much distinction between Computer–Assisted Language Learning systems, and other disciplines of CAI. Existing systems were mainly

of the drill-and-practice variety, the equivalent of electronic flashcards. By the late 1960's, attempts were being made to use interactive feedback to control the systems' operation [Uhr,1969]. The earliest forms of feedback-control involved merely re-asking problems which were previously answered incorrectly. By the early 1970s, attempts were being made to adjust the difficulty of problems based on performance [Woods,1971]. Discussion ensued on when help should be offered to the student, and whether the student or the tutoring system should determine the level of difficulty of the problems [Hartley,1973].

By the 1980s, many projects were concentrating on the best way to represent the state of the student's knowledge, in order to optimally adapt to it [Goldstein,1981], [Burton,1981a-b], and [Sleeman,1981a-b]. Research had also diverged into two distinct areas, template software for teachers to author their own computer-assisted lessons, and artificial intelligence and auto-adaptation programs which supported little, if any, teacher modification [Boyd,1982].

Today, CALL research and CALL systems typically fall into two broad categories. The largest and most sophisticated systems are developed and maintained by large universities for internal use. They have evolved over many years, and are highly customized to support their own language curriculums. Likewise, the most sophisticated systems in related areas (for instance, business document critiquing systems), have been developed internally by large corporations and large universities. These types of organizations have the advantage of large computing resources, and projects which slowly evolve and grow as they reach the more subtle hurdles. Another distinct advantage is that, as their own customers, the researchers in these universities and corporations have a continuous open channel for feedback, both positive and negative. At the other end of the spectrum, most commercially available CALL software is much less sophisticated. The intended customers, typically individuals and pre-college educational institutions, tend to have much more limited computing resources, and only a very small budget for educational software. In order to meet these customer constraints, and to bring such products to market in a timely manner, most of these packages address very narrow topics at the beginner level.

One large non-commercial effort that has been in-progress since the early 1970's is at Concordia University (Canada). They began experimenting with both student- and teacher-oriented CALL programs, the main focus of which were to improve the effectiveness of their own remedial and second language English programs. Narrow topics, such as specific verb tenses, were addressed individually, with the hopes of eventually building a library of software for a large range of grammatical and syntactical topics. One of the more ambitious later projects was a "whole composition" analyzer, which attempted to diagnose student-constructed sentences [Boyd,1982]. Several notable concepts were illustrated in this project. First was the decision to check sentences for common forms of faults, rather than on a true AI analysis of the sentence for correct-

ness. It proved that a benefit could be still be realized, while narrowing the scope and the complexity of the task. Another successful concept was the extensive compiling of statistics, including comments solicited directly from the students at the end of lessons, for adjusting and enhancing the diagnostics (for continuous improvement).

In the early 1980's, a similar program was being tested at Kings College (Scotland). This program was used for instruction in French, and illustrated some other successful characteristics [Farrington,1982]. A common complaint had been that CAI software often did not recognize alternate correct answers. This program specifically researched and hard-coded multiple correct answers to each problem. User-friendliness (to both those familiar with computers, and those not) was another frequent complaint about early CAI attempts. This program offered hints, encouragement, praise, etc., if desired by the student. This was an improvement over the "display a problem", "read student response", "if incorrect, print correct answer", "display next problem" cycle of earlier drill-and-practice software. Other attention was paid to user-friendliness, giving the student control over the level of help, and the ability to exit at any time. Similar to the Concordia University program, incorrect answers were stored so that future enhancements could handle new variations of answers and new types of errors.

During the same time period, CALL research was also being conducted at the University of Waikato (New Zealand). Various projects examined issues around natural language tutoring. The pros and cons of using the first language (meaning the student's normal language, as opposed to the one being taught) on the screen were explored. This issue is still not resolved, though. It was also concluded that exercises were more effective when not categorized in such a fashion that the student already knew what problem area was being tested. The projects also experimented with overall user-friendliness of CALL systems.

Several researchers have addressed alternatives to hard-coding answers to exercises. One possibility is to hard-code a variety or hierarchy of correct answers, as mentioned earlier. An alternative is to "can" common wrong answers as well as correct answers, for efficient trapping of errors [Ferney,1989]. One successful program which uses canned knowledge, the LITTRE program, pushes the hard-coding technology to its limits [Farrington,1986]. The program has proven very useful, and comes across as quite knowledgeable in French. The disadvantage has been that each individual sentence requires extensive analysis before being canned.

At IBM Research, the work of Roy J. Byrd, and associates, has contributed much useful information on word-based storage and morphological analysis [Byrd,1983, 1986, 1988]. All of these works have been based on a component dictionary system called UDICT. Intended as a building block for general-purpose natural language processing, the concepts for knowledge representation are relevant to CALL. There are two components to the UDICT system. A database component consists of words, feature information (such as part-of-speech, gender, etc.), and morphological information (rules on

how the variations of the word are formed). The second component is a morphological analyzer to process the feature information in order to match variations to the base word. Most of the database information is encoded in binary fashion (e.g. the "masculine" feature would be either on or off) so as to take a minimum of space, and allow fast access. So far this representation has been undertaken for English and French. The straight–forward implementation, however, lends itself to relatively easy adaptation to other western languages.

An examination of the currently available commercial CAI packages for the German language provides some additional insight into the state of the art. Included, as Appendix D, is a list of currently available software packages for German, along with brief descriptions of each. Nearly all are variations of the drill–and–practice method (as is this thesis project). Approximately half of the packages handle single–word vocabulary only, while the other half attempt in some way to address sentence–level constructs. Approximately half have chosen to present the exercises in a game format. While all are described briefly in the Appendix, two which most directly address the same domain (sentence construction) as this thesis are discussed below.

The German Word Order program is an interesting variation on sentence construction exercises. All sentences are canned. The program chooses a sentence from its internal list, then prints the words in random order. The student must attempt to reconstruct the correct sentence. Because the student must use exactly the same words as the original sentence, parsing for correctness becomes much easier. One problem with this particular program, is that it does not accept correct sentences which do not match the original. To take an English example, "The truck hit the car" would not be accepted if the system's version of the original sentence was "The car hit the truck". This can be particularly restrictive if an alternative variation of the correct sentence retains the same meaning as the original. In this case, the student may be misled into believing that only the one form is correct.

The program gives interactive feedback while the student constructs the sentence. As each word is entered, the computer tells whether or not it is correct. This helps to alleviate the problem of alternative correct sentences (in the above example, "truck" would have been flagged as incorrect, giving the student the chance to try "car" instead). Hints are also available along the way. The student can review a brief lesson on the concept which is employed in the current sentence, or receive the correct answer, at any point.

Another program which deals with sentence construction is the Dasher program. It begins much like the system implemented for this thesis, by presenting a sentence for translation. The difference is that it does not attempt any intelligent diagnosis or tutoring. The student response is compared letter–to–letter with the expected response (also greatly simplifying correctness parsing), and echoed back to the student with dashes (hence the name) in place of any discrepancies. This occurs a specified number of times before the correct answer is given. The re–try count is configurable.

The field of CALL research is still drawing interest. One researcher comments that the current trick lies in discovering and exploiting the knowledge representations which would allow systems to take shortcuts, like a human would, in analyzing sentences [Cameron,1989]. Another researcher comments that the pioneering spirit is still alive, and that we are still building a base of background material to establish clear direction [Last,1989]. He also criticizes projects tackled by educators without enough programming theory, and projects tackled by programmers without regard to education theory. These are not unfounded complaints, and such needs of CAI/CALL researches have prompted education experts to publish materials intended for the programming audience [Fox,1989], [Dhaif,1990]. Another resource to CAI/CALL researchers are papers intended for teachers on how to select good CAI software [LaReau,1989]. Both contain information of value to CAI designers – advice on effective and ineffective teaching techniques, user interface techniques, and the perceptions of students and teachers.

1.3. Scope of Project

The editors of a collection of computer-assisted language learning research papers, commented in their introduction:

> "As one reads through the chapters in this book, one realizes that much has been achieved and that much more will be achieved by selecting ideas which have been tried out by individuals and by amalgamating them." [Cameron,1986]

The primary deliverable for this thesis is a prototype Intelligent Tutoring System which attempts to do just that. It is accompanied by design, user, and administrative documentation. It combines some of the concepts described in the previous section, to provide a small package supporting semi-generated phrases, intelligent critiquing of student responses, assistance to the student, if desired, and information gathering for further improvement of its capabilities.

The semi-generated phrases mark a hybrid between the all-canned phrases of such programs as the LITTRE program, and true random sentence generation. The system is programmed to diagnose specific forms, or models, of phrases and sentences. These models draw from pools of semantically correct selections, to eliminate the problem of generating nonsensical phrases. However, all phrase construction, and diagnostic pattern construction, occurs at run-time. The details of diagnosing a specific phrase are abstracted so that they apply to the entire model. The run-time construction, then, allows the system to correctly diagnose whatever vocabulary happens to be used for a specific exercise. Besides the programming advantage, this also makes adding phrases or sentences, of existing forms, very easy.

Externally, the tutoring system is fairly simple. It handles a single type of exercise, sentence translation, with diagnostic feedback. When the student logs in, the system queries the student's current knowledge level. All vocabulary and sentence models within

the system are marked with the chapter in the textbook in which they are introduced. By requesting the student's knowledge level, expressed as the highest chapter completed in the textbook, the system can present only sentence forms and vocabulary which it knows the student has been exposed to. The knowledge level concept is discussed in more detail in section 1.3.2. below, and in the Design and Implementation chapter later in the report. A few general user-interface rules apply. At any user prompt, the student is able to ask for help or exit the program. At any user prompt within a specific exercise, the student is able to abort that exercise and begin a new one. Help text includes information on what is expected at the specific prompt, as well as on these general rules of control. A typical single-exercise dialogue consists of:

- The system presents a phrase or sentence to the student, for translation.
- The system reads the student's response.
- If the response is correct, the system says so, and a new phrase is presented for translation.
- If the response is incorrect, the student is asked if he or she would like to try again, receive the correct answer, or receive a hint from the system.
- If the student chooses to try again, the system reads another translation response (and attempts to diagnose it).
- If the student requests the correct answer, it is given, and the system presents a new phrase or sentence for translation.
- If the student requests a hint, the system attempts to provide a clue as to what is wrong with the response, without giving away the answer. If subsequent hints are requested, the system attempts to give successively more detailed clues.
- During the course of the exercise, statistics and information about specific responses are logged, for the purpose of future enhancements to make the system more robust.

Additional detail about the processing and user interface can be found in the Software System Description chapter later in this report, and in the User and System Administration Manuals included as appendices. Because the project had multiple objectives, the scope of each is addressed below.

1.3.1. System Functionality

Exhaustively addressing all possible sentence constructions for a language would obviously have not been manageable as a thesis project. It was a goal, however, to implement enough variety to illustrate the viability of its concepts. This project does not constitute a major leap forward in CALL technology, but it is hoped that its experimental components represent small steps in the evolution towards more effective tools. By taking concepts from successful systems which used hard-coded data, and attempting to abstract the techniques, the prototype attempts a blending of canned and "intelligent" processing. Another key

experimental element was the adaptation of a sentence masking analysis technique [Phillips,1983] as a processing engine for the system. It, in effect, turns the knowledge-base into the diagnostic programming language of the system. A final experimental element was the design and implementation approach. Natural language handling seemed to logically suggest an object-oriented approach (word objects, sentence objects, etc.), and the design attempted to build on that.

The knowledge base and processing support all aspects of the example dialogue described above, plus additional detail as described later in the report. This includes the ability to choose pseudo-random, and non-repeating exercises and vocabulary. To demonstrate these features required implementing multiple phrase models, multiple vocabulary selections, multiple knowledge levels, multiple types of errors, and multiple levels of error diagnosis (to support increasingly detailed hints). Being somewhat of a feasibility study, though, it was a goal to sample multiple models, vocabulary, errors, etc., which specifically showed different characteristics (as opposed to increasing the volume of similarly processed models).

Phrase models and vocabulary are selected pseudo-randomly, so that the user does not know what phrase form or concepts will come up next. Additionally, during any given session, phrases presented are marked as "used" so that they do not repeat while there are still new phrases to be presented. If the entire knowledge base ends up being marked "used" (which may happen at lower knowledge levels), the system will clear all the "used" markers, and continue drawing pseudo-random selections.

The prototype system delivered with this thesis will not dynamically adapt its behavior based on individual users' previous histories, nor will it be teacher-configurable. These and other possible future enhancements are discussed in the Conclusions chapter, later in this report.

1.3.2. Design and Implementation

The tutoring system is highly tied to a specific textbook. This allowed the phrase model and vocabulary selection, and choice of diagnostic errors, to be done in a logical and pre-established way. This also provided a convenient means for defining the "knowledge level" concept. Using "highest chapter completed" as a measure of knowledge level provides a clear way to mark the internal knowledge. Phrase models and vocabulary are tagged with the chapter in which they were introduced. This also keeps the knowledge level concept straightforward for the student user. The use of the knowledge level addresses a comment complaint against CAI software - that the material does not align with different student's capabilities.

As mentioned, an object-oriented approach was taken. The scope of this objective was to apply object-oriented analysis and design techniques to the requirements for the proposed system, and then implement the design using a language which supports object-oriented constructs. Job-Oriented Object Analysis (JOOA) [Nichols,1991] and Class-Responsibility-Collaboration Design (CRC) [Wirfs-Brock, 1990] methodologies were used. The analysis, design and implementation details are documented in Appendix A. The tutoring system runs on the Computer Science Department Sun network.

1.3.3. System Internals

The principle objective, from an architecture standpoint, was the implementation of the pattern masking diagnosis technique. Most of the software development was related to integrating and illustrating the feasibility of this approach.

The dictionary portion of the system is very similar to the scheme described by Byrd, et al. Words and feature information are stored in the dictionary. When the system is building phrases to present to the user, it requests the variation of words that it needs by specifying the feature information. The dictionary is also used when diagnosing the student's response, to determine if incorrect words are variations of the expected word, synonyms, etc. The dictionary is limited to the vocabulary that the system currently supports (based on knowledge level). A possible future enhancement, to use a full external dictionary, is discussed in the conclusions.

2. SOFTWARE SYSTEM DESCRIPTION

2.1. Introduction

A later Design and Implementation chapter represents the dense, but thorough, documentation of the system development and architecture, using the techniques and terminology described in [Nichols,1991a-b] and [Wirfs-Brock, 1990]. What follows in this chapter is the "plain language" system description, intended as an overview of both its use and the underlying design. Two other documents of interest exist, and have been included as appendices to this report. They are the User Manual, and the System Administration Manual (appendices A and B, respectively). The User Manual is aimed at the student user who wishes to use the tutoring system to practice translation. The System Administration Manual covers two main topics, the organization and building of source code, and the availability of data gathered by the system to characterize and improve its performance.

2.2. User Interface Overview

The prototype system's user interface is not terribly sophisticated. It is text-based, interactive I/O. The user types input at the keyboard, and the system prints output to the screen. Most of the user input consists of attempted phrase translations, and command numbers chosen from menus along the way. Additional detail can be found in the User Manual appendix and the Sample System Runs chapter later in this report, but the brief interchange below illustrates the look and feel of the system. In system logs throughout this report, text printed by the system will appear in bold, and text typed by the user will appear in italics.

```
sys27> glt

                        German Language Tutor, version 1.0


If you are not sure how to begin, type "?".

Please Login: ken

Enter highest chapter completed (currently 2)? 3

        Exercises will be limited to chapter 3 and earlier.

        You will now be asked to translate various phrases and sentences.


Translate:
                the table is round
to German:
```

```
                der Tisch war rund

        That is not the expected translation.  Enter the number
        indicating the action you would like to take:

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        There is something wrong with the third word in your response.

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 1

Translate:
                the table is round
to German:
                der Tisch ist rund

        Yes!  That is the expected response.

Translate:
```

In the example shown, the student invokes the tutor using the 'glt' command, and then logs in using the username 'ken'. Valid usernames are assigned by the system administrator. Two types of users are supported, the "student" and the "system administrator". Upon logging in, student users enter the exercises, as illustrated above. System administrator users have different features available. Refer to the respective manuals for additional details on each.

The types of errors diagnosed are specific to different forms of phrases. Different levels of diagnosis are typically available. If the user above had requested another hint,

instead of choosing to try another translation, the system would have reported that it was the tense of the verb which was wrong. If the user had requested yet another hint, the system would have reported that the choice "war" was the imperfect form of sein, but that it was expecting the present tense.. Eventually, the system would have reported that it could not give anymore clues without revealing the answer, and the option of receiving a hint would have disappeared from the menu.

On-line help is available anytime the system is waiting for input, and the student is always free to quit the current exercise, or exit the system altogether.

2.3. Internal Architecture Overview

The design and implementation of the system was done using object-oriented methods, and that terminology is used here for consistency. The Terminology sections of the Design and Implementation chapter may be referred to for additional clarification, if needed.

2.3.1. High-Level Description

The fundamental object, from a processing standpoint, is the PhraseModel. The PhraseModel represents everything that the system knows about a specific form of phrase or sentence - what semantically correct vocabulary can be used, how to build a syntactically correct phrase from this vocabulary, and how to diagnose a student's translation attempt. The normal execution of the system is basically a sequence of exercises generated by choosing PhraseModels from the list which the system supports.

The fundamental object, from a functionality standpoint, is the TranslationTree. This is a binary tree of masking patterns, through which the student response falls as the system tries to diagnose it. An initial diagnosis is made at the first pattern which matches the response, and additional help (hints) can be supplied to the user by matching subsequent patterns. The TranslationTree and its nodes generate these patterns at run-time, so that they work with the vocabulary which was chosen for the specific exercise. The normal execution of a single exercise consists of the PhraseModel generating a phrase to be translated, and the TranslationTree processing student responses.

Another important object, as far as objectives of the system go, is the Log. This object gathers run-time data to be used in characterizing system usage, and improving its diagnostic capability.

The remainder of the objects in the system function primarily as support for those just mentioned. These are discussed further in the next section.

2.3.2. Detailed Description

A PhraseModel works for a particular form of phrase or sentence. This is represented by a PhraseDescriptor. The PhraseDescriptor contains the information necessary to create a syntactically correct phrase of the intended form. In order

to create a phrase to be presented to the student for translation, the PhraseModel draws a semantically correct BasePhrase from a pool of choices. A BasePhrase is an indication of valid vocabulary selections for this form of phrase, with no syntactic information included. The PhraseDescriptor uses the BasePhrase, and builds the syntactically and semantically correct phrase which the PhraseModel presents to the user.

The TranslationTree works from the same BasePhrase, and builds the system's expected response. All diagnosis is done by matching the student response against patterns, and the expected response is essentially the first pattern in the binary tree – a pattern with no wildcards. If the student response matches the expected response, this exercise is complete, the system logs some data and statistics, and another PhraseModel is chosen.

If the student response does not match the expected response, then it is fed to the first PatternNode in the binary tree. The pattern node builds a pattern, which consists of the expected response with some strategically placed wildcard(s). This pattern is matched against the student response, and if it fails, the student response is fed down the left branch of the tree until a match is found. Each generated pattern is different, and/or increasingly more generic, in an effort to isolate the problem area.

Once the student response matches a pattern, the system reports that the response was not the expected one, and asks the student how to proceed. If the student wishes to try again, the system resets to the root of the TranslationTree, and waits for another attempt. If the student asks for a hint, the one which is stored at the current PatternNode will be printed. If the student asks for additional hints, the system will feed the student's response down the right branch of the tree. Sub-sequent patterns will try to further isolate or qualify the problem, and hints at those levels will be increasingly more specific. Once the tree-parsing hits a leaf node, though, the system will tell the user that it can diagnose no further.

A Dictionary object supports the system, by returning the derived forms of words based on specified syntactic information. The PhraseModel and TranslationTree use this as they build the presentation and expected phrases.

A Statistics object maintains run-time counts, such as how many exercises have been attempted during the current session. This information is made available to the user periodically, as well as logged for future inspection.

Further detail about these, and additional, objects, may be found in the Design Document appendix. The next section is a system walk-through to illustrate how these objects work with an example.

2.3.3. Walk-through to Illustrate Internal Function

This walk-through is included as a way to show how the different objects interact, and how the various dynamic objects are created and used. It also shows how the pattern-tree diagnosis concept works.

2.3.3.1. Static Data

Before actually beginning the processing, it is important to see what static objects are in place. This is the system's knowledge base. A BaseWord is a primitive data element in the system. It is not actually a word, but rather a key into the dictionary indicating a specific word, but not the word's form. For example the BaseWord for "man" would also be the base word for "men". As will be seen later, a WordDescriptor contains information about what form of a word is required, and a BaseWord and WordDescriptor are all that is needed to obtain a Word from the dictionary. BaseWords are combined to form BasePhrases, which is one of the objects that a PhraseModel contains. BasePhrases are grouped into BasePhraseLists, which in turn are grouped into BasePhrasePools. The need for two levels of grouping will be illustrated below. Each PhraseModel has its own BasePhrasePool to draw from. To begin the example, here are two BasePhraseLists, each containing two BasePhrases:

BPL–1: "the", "man", "to be", "good"
"the", "dog", "to be", "hungry"

BPL–2: "a", "boy", "to be", "energetic"
"a", "bird", "to be", "noisy"

As mentioned earlier, a PhraseModel has a PhraseDescriptor which specifies the form of phrase or sentence represented by this model. This contains syntactic information. The PhraseDescriptor also contains information on the order in which the phrase must be built. This is so that words which must agree with each other (in gender, number, etc.) will. Here is an example PhraseDescriptor:

PD–1: "article, gender–dynamic", "singular noun",
"present tense verb", "adjective"
Fill–in order: 2, 1, 3, 4

The actual elements of the PhraseDescriptor are WordDescriptors. The WordDescriptors contain feature (or attribute) information describing the required form of the word. For example the noun specified above is singular, as opposed to plural. A special characteristic of WordDescriptors is that features may be marked as "dynamic" as is the gender of the article in the above example. This supports the run–time building of phrases where words must agree with each other, and will be shown later.

The following should illustrate the difference between BasePhrasePools and BasePhraseLists. A BasePhrase and a PhraseDescriptor together specify a syntactically and semantically correct phrase. The following example BasePhrasePool is composed from both of the example BasePhraseLists.

BPP–1: BPL–1
BPL–2

A PhraseModel using the example PhraseDescriptor, PD–1, and the example BasePhrasePool, BPP–1, would be able to generate the following valid phrases:

The man is good.
The dog is hungry.

A boy is energetic.
A bird is noisy.

BPL–1 and BPL–2 are separate entities, though, because they will not always be valid for the same PhraseModel/PhraseDescriptors. As an example of this, suppose the fixed_phrase_descriptor for a different PhraseModel is:

PD–2: "article, gender–dynamic", "plural noun",
"present tense verb", "adjective"
Fill–in order: 2, 1, 3, 4

A valid BasePhrasePool is:

BPP–2: BPL–1

which would generate the valid phrases:

The men are good.
The dogs are hungry.

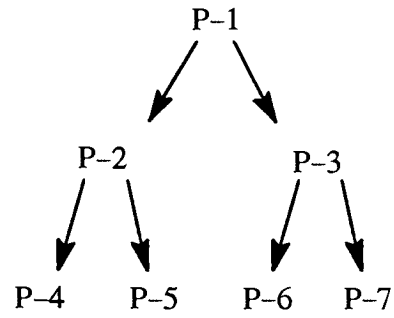
BPP–2 cannot contain BPL–1, however, because the generated phrases would not be valid:

A boys are energetic.
A birds are noisy.

BasePhraseLists, then, exist so that subsets of pools can be shared between PhraseModels.

The first example PhraseModel currently consists of BPP–1 and PD–1. The next component of a PhraseModel is its TranslationTree. A TranslationTree object is at the root of the binary pattern–matching tree. Like the PhraseModel, it also contains a PhraseDescriptor. This describes the expected response. In many cases, this will be the same PhraseDescriptor that the PhraseModel uses. In those cases were it is not the same, the TranslationTree will also contain a BasePhraseTranslator object which describes how to translate the original BasePhrase to the new language. This handles cases where the translated phrase does not take on the same form as the original (e.g. greater/fewer words or different word order). The example here does not require this, so no additional PhraseDescriptor is included.

The final static components of the PhraseModel and TranslationTree are the individual PatternNodes. The tree itself is pre–determined, but there are dynamic actions which occur during an exercise, and these will be illustrated below. In order to have an example to work from, consider the following tree (without yet specifying the details of each pattern):



That completes the description of what the system knows before starting an exercise. The next section examines the processing which would occur using this example data.

2.3.3.2. Run-time Processing

The first step of an exercise is choosing a BasePhrase to work with. It is intended that this be a random selection from the BasePhrasePool (BPP-1, in this case). For this example, say that "the" "man" "to be" "good" was chosen. The PhraseDescriptor (PD-1) is used to generate the phrase to be presented to the student. A special object, the DynamicWordDescriptor is used along the way to help resolve dynamic information in the WordDescriptors. As the phrase is constructed, feature information is set in the Dynamic WordDescriptor. Using the chosen BasePhrase, and following the fill-in order specified by the PhraseDescriptor, the second word of the presentation phrase would be obtained from the dictionary (for this example I'm assuming a German to English translation by the student, so the German forms of the word are filled in):

presentation = ____ Mann ____ ____

In addition to setting this word, its feature information is set in the DynamicWordDescriptor (specifically, the fact that the gender of this word is masculine). As each subsequent WordDescriptor is used, it is first checked to resolve any dynamic feature information against the DynamicWordDescriptor. The next WordDescriptor would then be resolved to "masculine article", and would be obtained from the dictionary:

presentation = der Mann ____ ____

These steps are repeated until the entire presentation phrase has been constructed. It is then presented to the user:

Translate:
der Mann ist gut
to English:

At this point, the TranslationTree takes over, and waits for the student's response. It builds the Phrase that it expects (in this case "the man is good") and compares the two. If the response is what the system expected, the system would inform the user, and this exercise would be over. For sake of example, though (and because gender is a convenient error to illustrate), consider the following exercise:

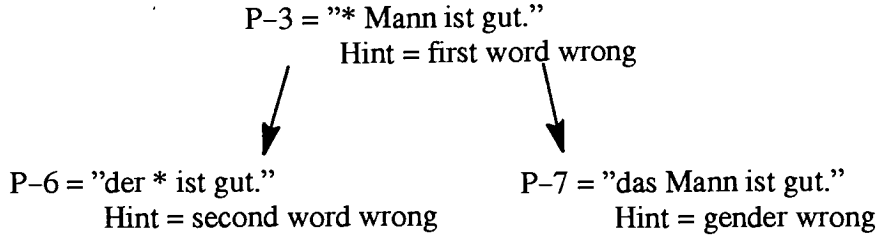
Translate:

the man is good

to German:

In this case, the correct response would be "der Mann ist gut". To illustrate error diagnosis, assume that the response was "das Mann ist gut" (gender of the article is incorrect).

Because the match with the expected response fails, the TranslationTree object feeds the actual response into the pattern tree. The following is simplified example of how a fragment of a pattern tree might look. Every PatternNode in the tree is similar in structure. Each contains the rules necessary for building the pattern needed at that level, and a hint as to the nature of the discrepancy at that level. The patterns are dynamically built, and the ones shown below indicate how they might look for this example vocabulary. (In the notation here, '*' represents the single-word wildcard.)



The student response would fall through the tree until it first matched a pattern. In this case, that would be P-3. At this point the system will ask the student what he or she wishes to do. As mentioned earlier, if the student chooses to try again, or to receive the correct answer, the system will leave the diagnosis tree. If the student requests a hint, though, the system will print the hint associated with the current PatternNode. In this case, it would tell the user that it did not like the first word in the response. If the user asks for another hint, the system will attempt a match down the right branch of the tree. In this case, a match is found with the pattern P-7, and the next hint is printed, this time telling the user that it was the gender of the first word that caused the problem. This can continue as long as the system has a right branch to follow. Once it hits the leaf, though, it will tell the user that no further diagnosis is possible. The left branch, P-6 would only have been taken if the response had not matched at P-3 originally. Internally, the left branch is considered the "no-match" path (taken while looking for the first match) and the right branch is considered the "match" path (taken to supply hints, as long as it continues to match the user's response).

That concludes a single exercise processing walk-through. Diagnosis will continue until the user answers correctly or quits the current exercise, then a new exercise will begin.

3. SAMPLE SYSTEM RUNS

3.1. Introduction

The following sections consist of excerpts from logs of sample sessions with the system. They have been broken into selections which illustrate different features and interesting aspects of the system. In logs, all text printed by the system appears in bold, and all text typed by the user appears in italics. The first few sample runs illustrate the basic functions available to the student user. Later samples illustrate specific types of errors being diagnosed for different phrase models. The last samples indicate the basic functions available to a system administrator user.

3.2. Sample Run

This sample illustrates a normal login, and the correct execution of a single exercise.

```
Sys27> glt
                        German Language Tutor, version 1.0

If you are not sure how to begin, type "?".

Please login: ?

        In order the use this system, you must have an authorized
        username.  If you have not received a username yet, see
        your system administrator.  If you do have a valid username,
        type it in now to gain access to the tutoring system.

Please login: ken

Enter highest chapter completed (currently 5)? 5

        Exercise will be limited to chapter 5 and earlier.

        You will now be asked to translate various phrases and
        sentences.  You may stop at any time - the important command
        to remember is "?".  Type a question mark at anytime to
        be told your options and be given the opportunity to leave

Translate:
                Is the soup warm?
to German:
                Ist die Suppe warm?

        Yes! That is the expected response.

Translate:
        .
        .
        .
```

Also illustrated above is the use of the user knowledge level. The system will not use vocabulary or phrase models from chapters higher than the student indicates.

3.3. Sample Run

This sample illustrates a failed login.

```
Sys27> glt
                        German Language Tutor, version 1.0

If you are not sure how to begin, type "?".

Please login: badname1

        That is not a correct username, please verify that
        you typed what you thought you did, and that you have not
        used upper or lower case when not expected.

Please login: badname2

        That is not a correct username, please verify that
        you typed what you thought you did, and that you have not
        used upper or lower case when not expected.

Please login: badname3

        Discontinuing attempted login after three tries.
        Please verify that you are using a correct username,
        or see the system administrator.

Sys27>
```

3.4. Sample Run

This run begins in the middle of a session, with an incorrect student response to a translation request. It illustrates the option menu, a student request for help, a student choosing to exit the tutor, and the statistics summary which accompanies program termination.

```
        .
        .
        .
Translate:
                here is the glass
to German:
                hier ist glas

        That is not the expected translation.  Enter the number
        indicating the action you would like to take:

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 6

        This is a menu of the options that you have at this point.
        Type the number of the option which you wish to take.
        Your choices at this point are to try answering the exercise
        again, have the system give you a hint about what was wrong,
        have the system give you the correct answer, go on to a new
        exercise, exit the system altogether, or view this help message.

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to next.
        5 - Exit the tutoring system.
        6 - Help.
```

```
Choice? 5

German Language Tutor exiting...

Duration of session - 20 minutes
Number of different phrases asked to translate - 50
Number of correct responses - 44
Number of total responses - 57

Sys27>
```

The summary data indicates that the student was logged into the tutoring system for 20 minutes. During that time, 50 different phrases or sentences were generated for the student to translate. Of that 50, 44 were answered correctly (indicating that the student "quit" or requested the system's answer 6 times). The final count above shows the number of different translation attempts (in this case, 57) that the student entered during the course of the 50 exercises.

3.5. Sample Run

This run begins in the middle of a session, with an incorrect student response to a translation request. It illustrates the user quitting the current exercise, and beginning the next. The next exercise is also answered incorrectly, in such a way that the system cannot perform any intelligent diagnosis. Note that the system does not offer any hints to the user, since it can not make enough of a diagnosis to do so. This run then illustrates the user requesting the system's answer.

```
Translate:
                Mr. Jones is an American
to German:
                Herr Jones ist ein Amerikaner

        That is not the expected translation.  Enter the number
        indicating the action you would like to take:

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 5

        The system was expecting:

                Herr Jones ist Amerikaner

Translate:
                How old is the child?
to German:
                this attempt resembles the original in no way

        That response is incorrect, but the system is unable
        to isolate the specific problem.

        1 - Try again.
```

```
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 3

        The system was expecting:

                Wie alt ist das Kind?

Translate:
```

3.6. Sample Run

This run begins in the middle of a session, with an incorrect student response to a translation request. It illustrates the student requesting a hint from the system. It then illustrates the student requesting subsequent hints, and finally, the system running out of hints. At that point, it illustrates the student choosing to try the translation again.

```
Translate:
                the table is round
to German:
                die Tisch ist rund

        That is not the expected translation.  Enter the number
        indicating the action you would like to take:

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        There is something wrong with the first word in your response.

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        The gender of the first word is wrong.

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        The gender of the article must agree with the noun.  The noun is
        masculine, but the gender of the article you used is feminine.

        1 - Try again.
```

```
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        The system cannot diagnose this any further.

        1 - Try again.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 1

Translate:
                the table is round
to German:
```

3.7. Sample Run

The following sample run illustrates a phrase in which a BasePhraseTranslator was used. In this case, the translation is not literal, it takes a slightly different form. Besides illustrating base phrase translation, this run also shows that the diagnosis tree can be designed to catch the literal translation as a possible student error.

```
Translate:
                the man is a merchant
to German:
                der Mann ist ein Kaufmann

        That is not the expected translation.  Enter the number
        indicating the action you would like to take:

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        An article should not be used in this case.

        1 - Try again.
        2 - Have the system give you another hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 1

Translate:
                the man is a merchant
to German:
                der Mann ist Kaufmann
```

```
Yes! That is the expected response.
```

3.8. Sample Run

This sample run shows a system administrator login. This user chooses to view diagnosis failures. This would be used as a means for determining how to enhance the system's diagnosis capability. Two different types of diagnosis failures are shown in the dialogue below. The first shows a case where the system successfully isolated the error, and provided hints, but the user asked for additional hints when there were no more to give. The second case shows an answer so different from the expected response, that the system was not even able to make an initial diagnosis. Both of these cases, though, may indicate areas where the system could be improved. (In the second case, this would be true if the actual response was something that should have been isolated by the diagnosis tree.)

```
                    German Language Tutor, version 1.0

If you are not sure how to begin, type "?"

Please login: sys_user

        Select desired function from menu below.

        5  - Exit the tutoring system.
        6  - Help.
        7  - Write current log entries to file.
        8  - Delete log file.
        9  - Save log file to new name.
        10 - View system diagnosis failures.
        11 - Generate report, system diagnosis failures.
        12 - View individual past session statistics.
        13 - Generate report, individual past session statistics.
        14 - View cumulative past session statistics.
        15 - Generate report, cumulative past session statistics.
        16 - Generate report, all information available.

Choice? 10

Leaf node encountered while looking for additional hints.
        Date - Sat Apr 03 1993    Username - stu_user
        Index of model chosen - 1
        Phrase which was to be translated - the house is old
        Expected response - das Haus ist alt
        Actual response - der Haus ist alt

        5  - Exit the tutoring system.
        6  - Help.
        4  - Return to main system administrator menu.
        17 - View next entry.

Choice? 17

Leaf node encountered while trying to diagnose.
        Date - Sat Apr 03 1993    Username - stu_user
        Index of model chosen - 2
        Phrase which was to be translated - the doctor drinks coffee
        Expected response - der Doktor trinkt Kaffee
        Actual response - Is this a valid translation attempt?
```

```
        5  - Exit the tutoring system.
        6  - Help.
        4  - Return to main system administrator menu.
        17 - View next entry.

Choice? 17

No further diagnostic failure entries found.


        Select desired function from menu below.

        5  - Exit the tutoring system.
        6  - Help.
        7  - Write current log entries to file.
        8  - Delete log file.
        9  - Save log file to new name.
        10 - View system diagnosis failures.
        11 - Generate report, system diagnosis failures.
        12 - View individual past session statistics.
        13 - Generate report, individual past session statistics.
        14 - View cumulative past session statistics.
        15 - Generate report, cumulative past session statistics.
        16 - Generate report, all information available.

Choice? 5

        Your session with the German Language Tutor is complete.
```

The "index of model chosen" is recorded in the log, in the event that the source code will be entered. This points the programmer to the appropriate phrase model.

3.9. Sample Run

This is another system run, showing a system administrator viewing run-time statistics. The first two sets of statistics are from individual system runs. After returning to the main menu, the system administrator views the cumulative statistics for all statistics entries in the log.

```
                        German Language Tutor, version 1.0


If you are not sure how to begin, type "?".

Please login: sys_user

        Select desired function from menu below.

        5  - Exit the tutoring system.
        6  - Help.
        7  - Write current log entries to file.
        8  - Delete log file.
        9  - Save log file to new name.
        10 - View system diagnosis failures.
        11 - Generate report, system diagnosis failures.
        12 - View individual past session statistics.
        13 - Generate report, individual past session statistics.
        14 - View cumulative past session statistics.
        15 - Generate report, cumulative past session statistics.
        16 - Generate report, all information available.

Choice? 12

Statistics after a student session.
        Date - Sat Apr 03 1993     Username - stu_user
```

```
        This session lasted 9 minutes, 12 seconds.
        Number of different phrases asked to translate - 2
        Number of correct translations - 1
        Number of total responses - 3

        5  - Exit the tutoring system.
        6  - Help.
        4  - Return to main system administrator menu.
        17 - View next entry.

Choice? 17

Statistics after a student session.
        Date - Sat Apr 03 1993    Username - stu_user
        This session lasted 1 minutes, 14 seconds.
        Number of different phrases asked to translate - 3
        Number of correct translations - 0
        Number of total responses - 3

        5  - Exit the tutoring system.
        6  - Help.
        4  - Return to main system administrator menu.
        17 - View next entry.

Choice? 17

No further statistics entries found.


        Select desired function from menu below.

        5  - Exit the tutoring system.
        6  - Help.
        7  - Write current log entries to file.
        8  - Delete log file.
        9  - Save log file to new name.
        10 - View system diagnosis failures.
        11 - Generate report, system diagnosis failures.
        12 - View individual past session statistics.
        13 - Generate report, individual past session statistics.
        14 - View cumulative past session statistics.
        15 - Generate report, cumulative past session statistics.
        16 - Generate report, all information available.

Choice? 14

Cumulative statistics from log file.
        Number of individual sessions counted - 2
        Total duration of sessions - 10 minutes, 26 seconds.
        Average duration of sessions - 5 minutes, 13 seconds.
        Total number of phrases presented by system - 5
        Total number of student responses - 6
        Total number of correct translations - 1

        5  - Exit the tutoring system.
        6  - Help.
        4  - Return to main system administrator menu.

Choice? 5


        Your session with the German Language Tutor is complete.
```

4. DESIGN AND IMPLEMENTATION

4.1. Introduction

This chapter summarizes the three major phases in the German Language Tutoring System development: analysis, design, and implementation. There are various object–oriented analysis and design techniques being tried today, many of which borrow elements from one another. This effort relied heavily on two methodologies – Job–Oriented Object Analysis (JOOA) [Nichols,1991a] and Class–Responsibility–Collaboration (CRC) Design [Wirfs–Brock,1990]. The methodologies are fairly intuitive, at least once the potential confusion with structured analysis and design concepts are overcome. The documentation should be fairly straightforward, and enough background description has been included to allow the reader unfamiliar with these methodologies to follow the design notes.

In general, object–oriented approaches tend to be more iterative, as opposed to using the more rigid phases and gates of other methodologies. The lines between analysis, design and implementation tend to blur, as earlier decisions are re–visited to address modularity, reusability, and responsibility issues uncovered in later phases. The process does not lend itself to intermediate documentation. Instead, the documentation here represents the more–or–less "stable state" which was achieved near the end of each phase.

In addition to the development phases mentioned above, the system itself was approached as having two distinct design components. The traditional "system design" defined the internals of the system – the mechanics of the algorithms, data structures, etc., and the "knowledge base design" supplied the actual data which drives the system. The system design alone specified an empty shell. The knowledge base design filled in the data. For this type of data–driven processing, the data design is critical. Diagnosis of student responses is done by using trees of patterns to isolate and identify errors. These trees are part of the knowledge base data, and must be carefully chosen to correctly identify errors.

The system design is described beginning immediately below. The knowledge base design is described beginning with section 4.3.

4.2. System Design

4.2.1. System Analysis and Design Goals

The fundamental goal was to produce a working prototype system, experimenting with the use of several concepts. It would be designed, implemented and documented in an object–oriented fashion. In conjunction with this, it was a goal to realize some of the benefits of this approach – an intuitive design, modularity and enhanceability. It was a goal to make the system functional, from a user standpoint, and to include enough

breadth of knowledge to illustrate the various internal features (though not the breadth of knowledge necessary to truly be a marketable product at this point). It was a goal to incorporate the pattern masking technique of diagnosis described earlier in this report, and to support the run-time gathering of data for future improvement of its diagnostic capability.

4.2.2. JOOA: Job-Oriented Object Analysis

JOOA is one of several object-oriented analysis methodologies. It, in turn, attempts to integrate the best concepts from two other methods: "Role Analysis" and "Visualization". It was developed in 1987, and has evolved since then as it has been used for large commercial software development projects. The designer who is familiar with structured, or functional, analysis and design, may find it more difficult to use these methodologies effectively than the designer who is already comfortable with object-oriented programming. Once the basic terminology, notation, and concepts are understood, the approach is fairly intuitive.

The main objective in JOOA is to iteratively identify and refine objects, until they can be described neatly and encompass the specified functionality. It involves approaching the system from two different perspectives of jobs (for which the system is intended) in order that the two provide checks and balances for each other. A brief review of the necessary terminology should be sufficient to allow understanding of the actual analysis report which is included below. Object-oriented documentation, like the design itself, will not necessary convey all necessary information in a single sequential read. However, by referring backward and forward to diagrams and descriptions, the reader should be able to reconstruct the design process.

4.2.2.1. JOOA terminology

There are not too many definitions here, so they have been listed in a more logical, as opposed to alphabetical, order.

Work Place – This is a job view of the system from an external standpoint. In its most literal sense, this would be the actual place where one human works, such as the control panel of a certain machine. Workplaces can also be physically identifiable components of a system, again from an external standpoint (e.g. the feeder, duplicator, and collator components of a copier). In the case of a software system, such as this tutoring system, the workplaces can be the different "hats" that a user of the system might wear (e.g. the teacher, the student, the system administrator or system designer, etc.)

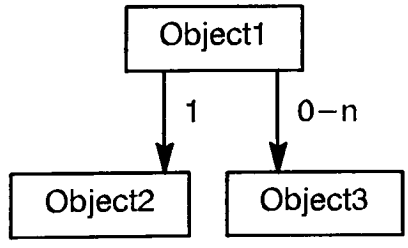
Agent – An agent represents a job step in the system. An easy mistake for the functional designer, is to consider agents to be syn-

onymous with tasks or functions in a structured analysis. In JOOA, an agent is typically at a higher level than this. As the analysis progresses, the agents should become trivial to the point of being just a series of delegations to objects. In fact, many or most agents may go away altogether.

Object – An object is a tangible component of the system. It may represent a physical object (e.g. a "word", in this project's context). An object will have a purpose, some amount of knowledge, and some amount of responsibility (things that it can do to/with its knowledge).

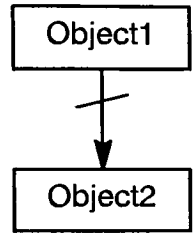
Work Flow – This is a pictorial representation of the flow of objects between agents. Again, the functional designer may confuse these with data flow diagrams. Their true intent is to show agents as black box components, again at a higher level, and show how they are tied together only by the exchange of certain objects. The inputs and outputs of the agents are numbered to represent "pins" in this schematic-like representation of software components. One of the principle reasons for agents and work flows being at a higher and more generic level than tasks and data flows, is to avoid restricting their evolvement. Whereas tasks and data flows tend to be rather stable, and drive the rest of the design, agents and workflows will tend to change as responsibilities are traded between agents and objects as the design coalesces.

Composition Graph – This is the pictorial representation of an object which contains other objects as part of its internal knowledge. The labels on the arrows indicate how many sub-objects are contained, and may be absolute numbers or ranges:

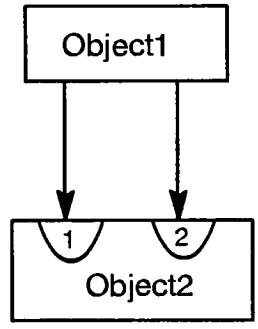


Begetter Graph –This is a pictorial representation showing objects which create or destroy other objects. It is distinguished from a composition graph by the lack of labels on the arrow. A slash through the arrow indicates object destruction. If no explicit destruction graph is shown, an object is assumed to be destroyed

by the same object which created it.



Collaboration Graph – This type of graph shows which agents and objects are "customers" of other objects. The numbers within each object box indicate the specific responsibility that the customer is expecting, and are typically described in a key accompanying the graph. This type of graph is usually used in the lower levels of design, but a high-level collaboration graph has been included early in the documentation below to illustrate the Session agent's context.



Object2
1. Responsibility is...
2. Responsibility is...

There are additional terms used in the JOOA literature, but since this particular design does not require these elements, they have not been included here.

The design of the tutoring system begins with the next section.

4.2.2.2. Work Places and Work Flows

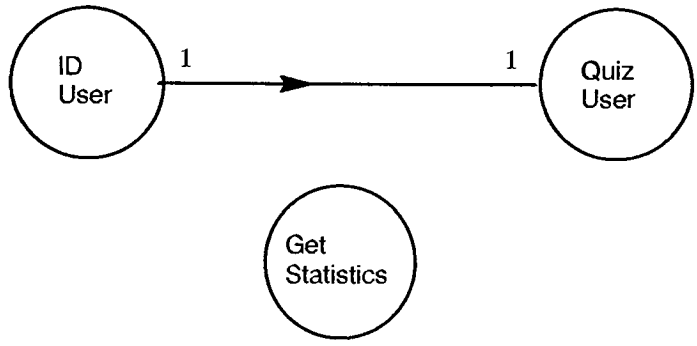
Three workplaces were identified for this system. They are listed below with their basic responsibilities.

Student
- ID User
- Quiz User

System Administrator
- ID User
- View and Manipulate Logs

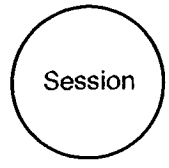
Teacher (future)
- ID User
- Edit Users
- Edit Knowledge Base

These initial agents were identified from the responsibilities listed above. Since the "Teacher" workplace is a future enhancement, those responsibilities were not included:



It turns out that this system is not very complex, from this external perspective, so the work flow diagrams are not extremely interesting. The object path shown above represents the flow of a single object, a User.

Normally, the above diagrams would not be included, since they changed in the course of object identification. They have been left above, however, to comment on the process in spite of the fact that only one agent survived intact. This agent, Session, evolved predominantly from the original "QuizUser" agent. Most of the IDUser and GetStatistics responsibilities were delegated to the UserList and Log objects, respectively. Looking ahead, this is actually an ideal situation, because this single agent could then become the main() function of the C++ implementation. This leaves the final set of Work Flow diagrams for this system as:



Given the above information, the Work Places are now documented:

4.2.2.2.1. Student

This is the primary user of the system. A typical interaction would include logging in, answering some exercises, receiving some feedback, and logging off. A high level of computer competency should not be assumed for this type of user.

4.2.2.2.1.1. Agents Used

- Session

4.2.2.2.2. SystemAdministrator

Because this is an experimental system, it will provide for a class of user whose interest will be in system performance, correctness, and other design issues. A higher level of system and computer knowledge may be assumed if necessary.

4.2.2.2.2.1. Agents Used

- Session

4.2.2.3. Agents

This section defines agents which are not part of any workflow (in this case, the sole agent).

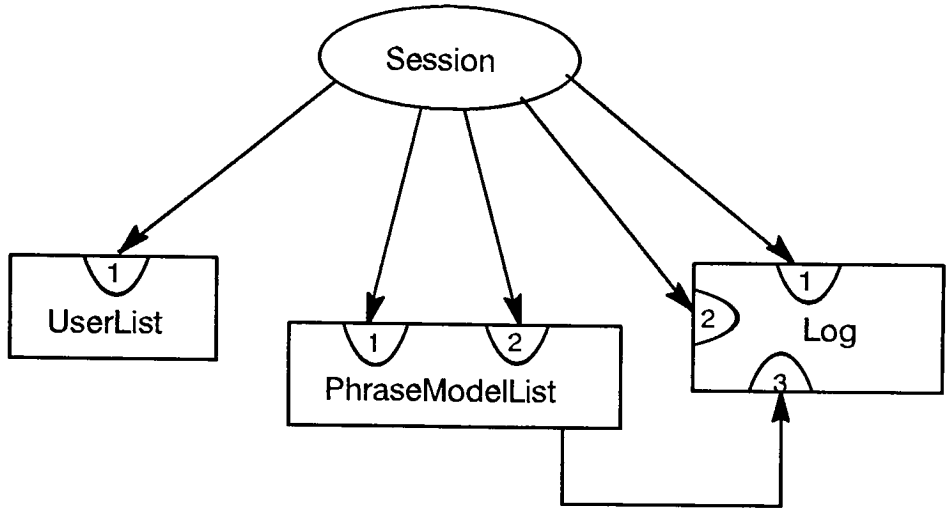
4.2.2.3.1. Session

The essential role of this agent is to control the complete login to logout interactive session with a single user.

4.2.2.3.1.1. Responsibilities

- Perform any general system set–up and shut–down.
- Initiate a user login.
- Coordinate series of exercises for the student user.
- Coordinate the access to and manipulation of logged data for an administrative user.

Because there is only a single agent in this design, the following collaboration graph may help illustrate how this agent relates to other major objects in the system.



UserList
1. read and validate user

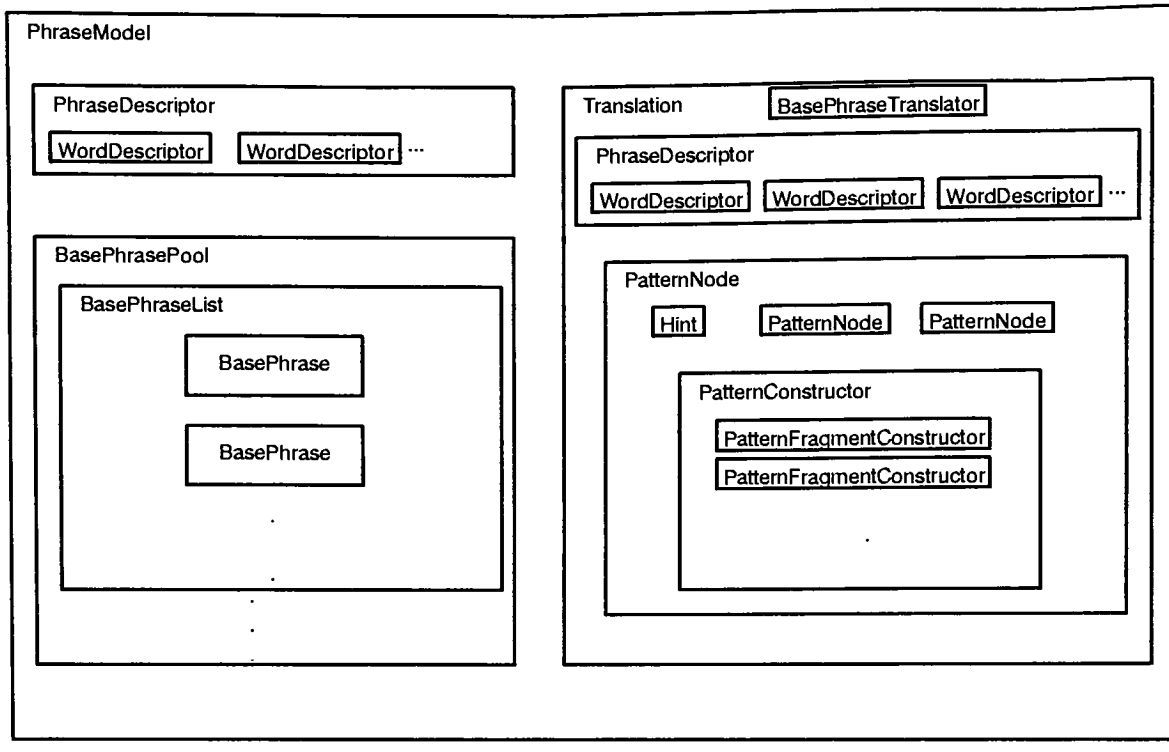
PhraseModelList
1. set current knowledge level
2. do exercise

Log
1. examine data
2. manipulate data
3. add data

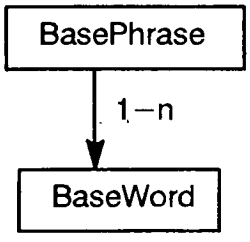
4.2.2.4. Object Graphs

This section provides an alphabetically listed description of each object uncovered during analysis. Those objects which are necessary for the essential purpose of the system (i.e. user exercises) are described in more detail than other less essential objects (i.e. log manipulation). This is intentional, as it leaves greater design- and implementation-time flexibility in those areas which are less critical.

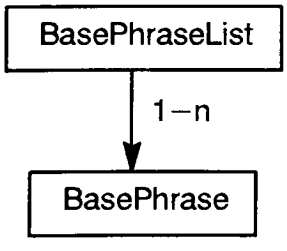
Before the textual descriptions begin, the composition and begetter graphs are presented. These can be referred back to, in order to understand the relationships between the described objects. The composite illustration below is a visual aid that represents the most complex of the composite objects, the PhraseModel. It conveys roughly the same information as the individual composition graphs (with the exception of the count labels), but it may be helpful to see them together for this complex object. Trivial "composition" graphs for objects which are not currently composed of additional objects have also been included. This adds completion to the illustrations, since all objects are now represented. Additionally, a brief statement of each object's responsibilities has been included next to its graph.



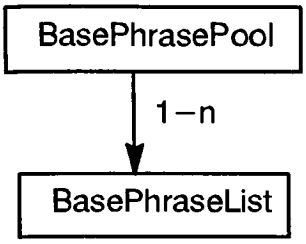
4.2.2.4.1. Composition Graphs



A string of BaseWords representative of a semantically valid phrase. Most likely not syntactically correct.

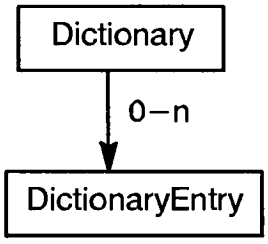


A collection of BasePhrases of like length which can be made syntactically valid by application of the same PhraseDescriptor.



A collection of BasePhraseLists containing BasePhrases of the same length, and which can be made syntactically valid by application of the same PhraseDescriptor.

A set of rules which describes how to translate a specific BasePhrase associated with a specific PhraseModel.

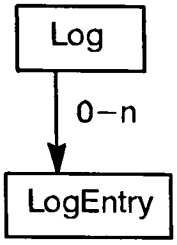


This represents the system's vocabulary.

This represents the entire dictionary entry for a single base word (including how to derive other forms of the word, etc.)

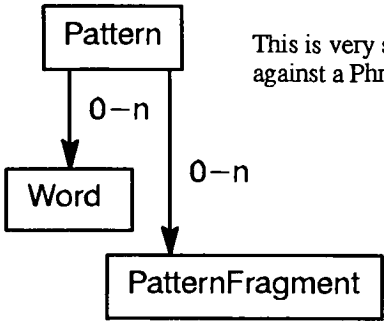
Collects attributes of words in a phrase, in order to dynamically set the attributes of later words which must agree with the earlier ones.

A block of text. Associated with a pattern, it describes the conclusion drawn based on the user's input matching this pattern.

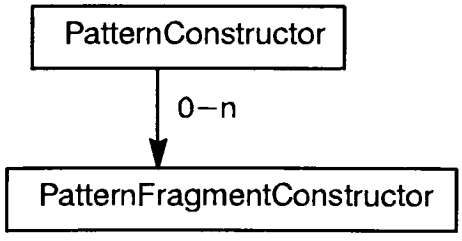


This object represents all run-time gathered data currently being maintained by the system.

This is represents a single recordable incident. There will be multiple formats that share some common sub-format.



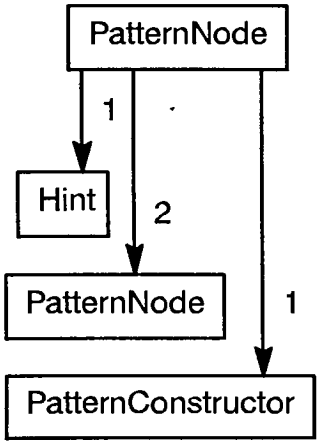
This is very similar to a Phrase, but it likely contains wildcards and can match itself against a Phrase.



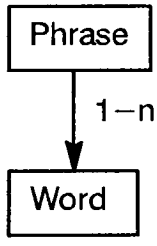
Represents the rules for converting a Phrase of a given semantic model to a pattern which will be used for diagnosis (i.e. containing wildcards).

This is a pattern primitive.

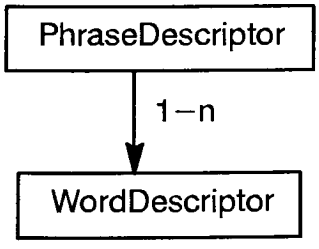
Describes how to modify a Word in a phrase. This is a pattern-building tool, where patterns containing wildcards are constructed from existing Phrases.



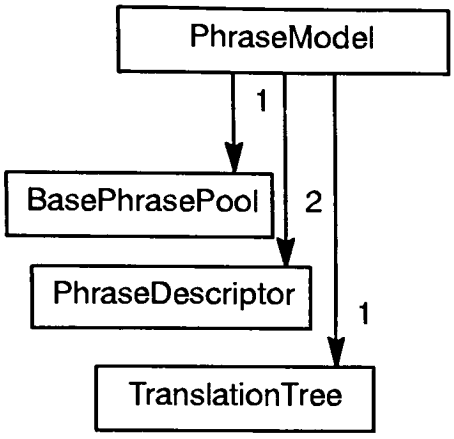
This is a single entry in a binary tree of Patterns which is used to process user responses. It contains information on how to construct the current pattern, as well as which Patterns to move on to, based on whether or not the user response matches at this point.



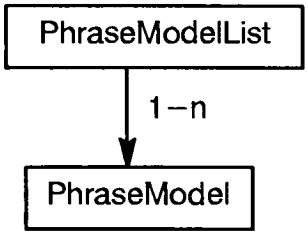
This is an actual semantically and syntactically correct text phrase which can be printed to the screen.



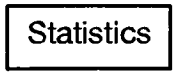
A string of WordDescriptors representative of a syntactically valid phrase. A BasePhrase and a PhraseDescriptor contain all the information necessary to create a syntactically and semantically correct text phrase.



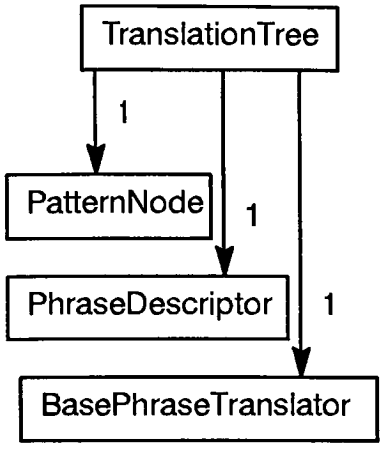
This models everything that the system knows about a specific semantic model phrase – what base phrases may be used in it, how to translate it, how to parse and diagnose the user's response, etc.



A collection of all the model phrases which the system currently supports.

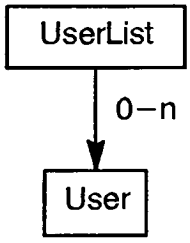


This object will collect and process various run-time statistics (number of exercises completed, etc.)

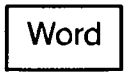


This object contains everything necessary to diagnose responses for a particular PhraseModel. It knows how to build the translation phrase, and holds the root pattern of the diagnosis tree.

This may eventually be a composite object, but at this level of the design it works fine as a base object. This represents everything that needs to be known about a given user - the login name, the user type (student, system designer, teacher), etc.



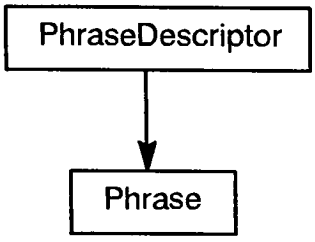
This object represents all the valid users who can log onto the system.



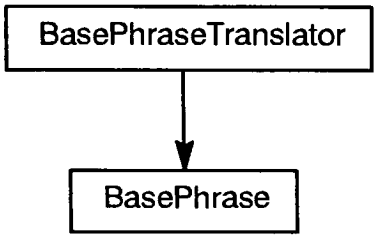
This is an actual text word which can be printed to the screen or used to build phrases.

A collection of attributes which specify what derivative of a word is needed. (Not the rules, just the attributes, e.g. 'plural', 'present tense', etc.)

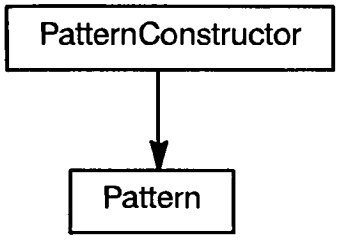
4.2.2.4.2. Begetter Graphs



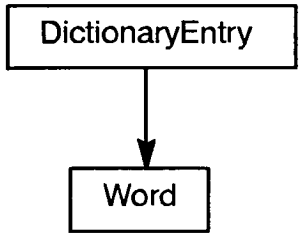
A PhraseDescriptor can create a Phrase when given a BasePhrase to work from.



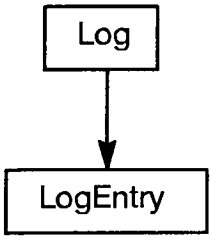
A BasePhraseTranslator creates a BasePhrase in the opposite language.



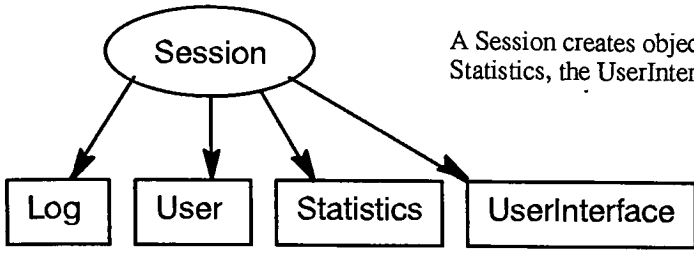
A PatternConstructor can create a Pattern when given a Phrase to work from.



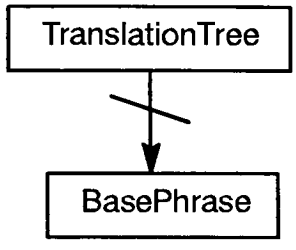
Given desired featured information, the DictionaryEntry will create the appropriate derived form of the desired word.



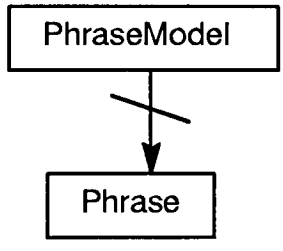
The Log creates individual entry objects.



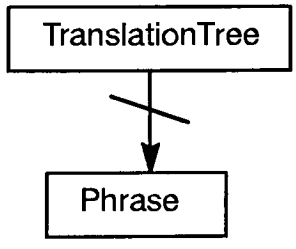
A Session creates objects to represent the current User, current session Statistics, the UserInterface, and the run-time data Log.



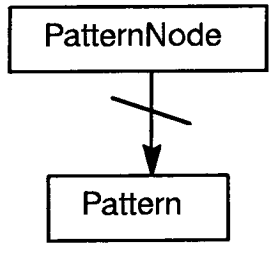
The TranslationTree destroys the translated BasePhrase when finished.



The PhraseModel destroys the presentation phrase once the exercise is completed.



The TranslationTree destroys the user response phrase once it is finished.



The PatternNode destroys the dynamically created pattern once it is done trying to match.

4.2.2.5. Object Descriptions

Each object is described individually below. They appear in alphabetical order.

4.2.2.5.1. BasePhrase

The essential role of this object is to specify a semantically valid phrase. Most likely not syntactically correct.

4.2.2.5.1.1. Responsibilities

- specify a phrase, but not the attributes of the words used.

4.2.2.5.1.2. Attributes

- words – The Dictionary identifiers for the base words in this phrase.
- length – The number of words in this phrase.
- active – a flag which indicates whether or not this phrase is within the current user's knowledge level.

4.2.2.5.1.3. Actions

- get_word_identifier(position) – Used when building syntactically correct phrases to obtain words.
- set_active(know_level) – Sets the active flag if the vocabulary in this phrase is within the user's knowledge level.

4.2.2.5.2. BasePhraseList

The essential role of this object is to collect BasePhrases of like length which can be made syntactically valid by application of the same Phrase Descriptor

4.2.2.5.2.1. Responsibilities

- collect, and allow selection of a random BasePhrase.

4.2.2.5.2.2. Attributes

- phrases – The collection of BasePhrases which make up this list.
- size – The number of phrases in this list.
- selected – An indication of which BasePhrases have been used in the current session.
- active – a flag which indicates whether or not any of the phrases in this list are within the user's knowledge level.

4.2.2.5.2.3. Actions

- get_base_phrase() – gets random base phrase from list.

- reset_selections() – Clean the slate of phrases marked as 'used'. Done to start over after all have been used.
- set_active(know_level) – Sets the active flag if any of the BasePhrases in this list are within the user's knowledge level.

4.2.2.5.3. BasePhrasePool

The essential role of this object is to collect all BasePhraseLists containing BasePhrases of the same length, and which can be made syntactically valid by application of the same PhraseDescriptor. Note that this is very similar to the BasePhraseList. The reason that they both exist is that different subsets of the BasePhrasePool (i.e. BasePhraseLists) may be applicable to different PhraseModels, and are then gathered into BasePhrasePools.

4.2.2.5.3.1. Responsibilities

- collect, and allow selection of a random BasePhrase.

4.2.2.5.3.2. Attributes

- phrase_lists – The collection of BasePhraseLists which make up this pool.
- size – The number of BasePhraseLists in the pool.
- depleted – An indication of which BasePhraseLists have been exhausted in the current session.
- active – a flag which indicates whether or not any of the lists in this pool contain phrases which are within the user's knowledge level.

4.2.2.5.3.3. Actions

- get_base_phrase() – returns random base phrase from pool.
- reset_selections() – Clean the slate of BasePhraseLists marked as "depleted". Done to start over after all have been used.
- set_active(know_level) – Sets the knowledge level of the system to match that of the current user.

4.2.2.5.4. BasePhraseTranslator

The essential role of this object is to specify the rules which describe how to translate a specific BasePhrase associated with a specific PhraseModel.

4.2.2.5.4.1. Responsibilities

- translate a BasePhrase, into a BasePhrase of the other language.

4.2.2.5.4.2. Attributes

- rules – a set of translation actions, specifically designed to translate a BasePhrase from one language to the other. (e.g. this may involve adding or deleting words, changing the word order, etc.)

4.2.2.5.4.3. Actions

- translate(BasePhrase) – returns the translation.

4.2.2.5.5. Dictionary

The essential role of this object is to represent the system's vocabulary – all the words that the system knows.

4.2.2.5.5.1. Responsibilities

- service requests for forms of BaseWords.

4.2.2.5.5.2. Attributes

- entries – the collection of individual DictionaryEntries which make up the Dictionary.

4.2.2.5.5.3. Actions

- get_word(word_identifier, WordDescriptor) – returns the derived form of the indicated word, as specified by the attributes in the WordDescriptor.

4.2.2.5.6. DictionaryEntry

The essential role of this object is to represent a given word in the dictionary. It represents all that the dictionary knows about a single base word (including how to construct other forms of the word, etc.)

4.2.2.5.6.1. Responsibilities

- service requests for information or derived forms of BaseWords.

4.2.2.5.6.2. Attributes

- word_identifier – A unique key used to identify words in the dictionary, and represent them elsewhere in the system.
- word_attributes – The attributes of a word which are pre–defined, such as the gender of a given noun.
- derivations – The derived variations of the base word.

4.2.2.5.6.3. Actions

- derive_word(WordDescriptor) – Applies necessary morphological rules to return the derived form of the word as specified by the attributes in the WordDescriptor.
- get_identifier() – Return the word's key.

4.2.2.5.7. DynamicWordDescriptor

The essential role of this object is to facilitate the propagation of word attributes for words which must agree with one another.

4.2.2.5.7.1. Responsibilities

- Accumulate attributes and resolve dynamic attributes for individual Words.

4.2.2.5.7.2. Attributes

- word_attributes – The list of attributes (gender, etc.) which may need to agree.

4.2.2.5.7.3. Actions

- get_attributes() – Return the current set of attributes.
- resolve_dynamics(Word) – Get any needed attributes from the input Word.

4.2.2.5.8. Hint

The essential role of this object is to assist the student when asked, or in certain error cases. It is basically a block of text, associated with a given Pattern.

4.2.2.5.8.1. Responsibilities

- Draw some conclusions for the user based on current response.

4.2.2.5.8.2. Attributes

- text.

4.2.2.5.8.3. Actions

- print() – display self to screen.

4.2.2.5.9. Log

The essential role of this object is to store and service accesses to run–time logged data.

4.2.2.5.9.1. Responsibilities

- store data.
- add (log) data.
- manipulate data.

4.2.2.5.9.2. Attributes

- count – the number of log entries being stored.
- entries – the list of log entries.
- current_data – some data will be maintained at this level for the current session (e.g. current_user).

4.2.2.5.9.3. Actions

- add_entry(entry data) – log data.

- set_current_data(data) – the log will keep track of some data which does not change on an entry–to–entry basis, e.g. current_user)
- manipulate_entries(commands) – various run–time viewing and report–generating functions.

4.2.2.5.10. LogEntry

The essential role of this object is to capture a significant piece of data for further review. There will be various forms of log entries, with data specific to the type of incident being logged.

4.2.2.5.10.1. Responsibilities

- Save, report, or otherwise process a single recordable piece of system information.

4.2.2.5.10.2. Attributes

- record_type – Identifies the incident–specific data format below.
- username – The current user at the time the data was logged.
- date_time – Date/time stamp, when the incident occurred.
- model – The PhraseModel which was being used at the time.
- type–specific_data – The information which will be significant to the system administrator/designer who will be reviewing it.

4.2.2.5.10.3. Actions

- print() – Prints data in some pre–established format.
- report(type) – Report–generation functions.

4.2.2.5.11. Pattern

This is a Phrase that contains wildcards.

4.2.2.5.11.1. Responsibilities

- Can match itself against a Phrase.

4.2.2.5.11.2. Attributes

- fragments – the string of words and/or wildcards which comprise the phrase.
- size – the number of fragments in this pattern.

4.2.2.5.11.3. Actions

- match(Phrase) – Return success or fail depending on whether or not the input Phrase is matched by this pattern.
- get_word(position) – Return the Word at the specified position.

- set_word(position, Word) – Defines the Word at the specified position.

4.2.2.5.12. PatternConstructor

The essential role of this object is to generate a pattern from a given phrase. It may contain PatternFragmentConstructors for more detailed pieces using Words.

4.2.2.5.12.1. Responsibilities

- build Pattern from Phrase.

4.2.2.5.12.2. Attributes

- rules – the steps needed to convert the expected Phrase form to a Pattern.

4.2.2.5.12.3. Actions

- build_pattern(Phrase) – modify the given text phrase to get a pattern with wild-cards.

4.2.2.5.13. PatternFragmentConstructor

The essential role of this object is to generate a pattern fragment from a given Word. It contains rules specifying how much of the word to use, what wild cards to inject, etc.

4.2.2.5.13.1. Responsibilities

- create pattern fragment from Word.

4.2.2.5.13.2. Attributes

- rules – a set of actions specifically designed to change a given Word. It may replace the entire Word with a wildcard, replace certain letter(s), delete the Word altogether, etc.

4.2.2.5.13.3. Actions

- build_pattern_fragment(Word) – modify the given Word.

4.2.2.5.14. PatternNode

The essential role of this object is to recognize and handle a specific type of error in a student's response. It will be a single entry in a binary tree of Patterns which support a given PhraseModel. It contains information on how to construct the needed matching Pattern, as well as which PatternNodes to move on to, based on whether or not the user response matches at this point.

4.2.2.5.14.1. Responsibilities

- diagnose and act on a student's response.

4.2.2.5.14.2. Attributes

- pattern_descriptor – A PatternConstructor which describes how to construct the actual current Pattern (complete with wildcards, etc.) from the translation phrase.
- hint – The hint which can be given if the user input matches this pattern (note, the user's input is already incorrect, if the system has reached the point of matching against patterns.)
- match_pattern – The next PatternNode to try if the user input matches the current pattern *and* the user wants further diagnosis.
- no–match_pattern – The next PatternNode to try if the user input does not match the current pattern (the system must keep trying.)

4.2.2.5.14.3. Actions

- process_response(expected_response, actual_response) – builds the necessary pattern, attempts to match the user input, and continue to subsequent no–match PatternNodes if necessary.
- more_hints(expected_response, actual_response) – very similar to process_response(), but follows the match PatternNode path at the user's request for additional hints.

4.2.2.5.15. Phrase

This is an actual semantically and syntactically correct text phrase which can be printed to the screen. Note that a BasePhrase and PhraseDescriptor fully specify a given Phrase.

4.2.2.5.15.1. Responsibilities

- a collection of words.

4.2.2.5.15.2. Attributes

- size – the number of words in this phrase.
- words – the string of words which comprise the phrase.

4.2.2.5.15.3. Actions

- print() – Print self to screen.
- read() – Read a phrase from the user.
- get_word(position) – Return the Word at the specified position.
- set_word(position, Word) – Defines the Word at the specified position.

4.2.2.5.16. PhraseDescriptor

The essential role of this object is to specify the syntactic requirements of a given Phrase. A BasePhrase and a PhraseDescriptor contain all the information necessary to create a semantically and syntactically correct text Phrase.

4.2.2.5.16.1. Responsibilities

- create a Phrase from a BasePhrase.

4.2.2.5.16.2. Attributes

- word_descriptor_list – the WordDescriptor for constructing each word in the desired phrase.
- size – the number of WordDescriptors in this PhraseDescriptor.
- fill–in_order – The order in which to generate the resulting Phrase, so that words that must agree with each other do.

4.2.2.5.16.3. Actions

- build_phrase(BasePhrase) – Constructs the syntactically correct phrase from the semantically correct BasePhrase.

4.2.2.5.17. PhraseModel

The essential role of this object is to model everything that the system knows about a specific semantic model phrase – what BasePhrases may be used in it, how to translate it, how to parse and diagnose the user's response, etc.

4.2.2.5.17.1. Responsibilities

- construct and process an exercise based on this model.

4.2.2.5.17.2. Attributes

- active – a flag which indicates whether or not this model is within the current user's knowledge level.
- pool – The BasePhrasePool which this model can draw from.
- fixed_phrase_descriptor – The PhraseDescriptor which contains the constant attributes.
- translation_tree – the root of a pattern tree used to diagnose user inputs for this model.

4.2.2.5.17.3. Actions

- set_active(know_level) – Sets the knowledge level of the system to match that of the current user.
- do_exercise() – Chooses random BasePhrase, constructs semantically correct phrase for user, reads user response, gives to pattern tree for diagnosis.
- reset_depletions() – Clear any internal data which has been marked "used" or "depleted" (e.g. BasePhrases, etc.)

4.2.2.5.18. PhraseModelList

The essential role of this object is to collect all the semantic model phrases (plus associated exercises, etc.) which the system currently supports.

4.2.2.5.18.1. Responsibilities

- coordinate exercises from different PhraseModels.

4.2.2.5.18.2. Attributes

- models – the collection of PhraseModels which make up the list.
- size – number of PhraseModels in the list.
- depleted – indicates which PhraseModels have been "depleted" (i.e. all vocabulary selections used in the current session).

4.2.2.5.18.3. Actions

- set_active(know_level) – Sets the knowledge level of the system to match that of the current user.
- do_exercise() – Chooses a random model for a single translation exercise to be asked of the user.
- reset_depletions() – Mark all data as "OK to use" again.

4.2.2.5.19. Statistics

The essential role of this object is to do run-time statistics gathering and processing.

4.2.2.5.19.1. Responsibilities

- gather, summarize, and print statistics.

4.2.2.5.19.2. Attributes

- various run-time data counts (e.g. number of exercises attempted)

4.2.2.5.19.3. Actions

- update_counts(new counts) – Increment run-time data counts
- print_summary() – Make any statistical conclusions and print.

4.2.2.5.20. TranslationTree

This object is the beginning of the entire diagnosis-side of a given PhraseModel.

4.2.2.5.20.1. Responsibilities

- Generates expected response.
- Reads user responses.
- Begins diagnosis.

4.2.2.5.20.2. Attributes

- pattern_root – This is the PatternNode which represents the root of the binary diagnosis tree.

- base_phrase_trans – This is the BasePhraseTranslator which generates the semantically (but probably not syntactically) correct expected response.
- phrase_descrip – This is the syntactic information for the expected response.

4.2.2.5.20.3. Actions

- check_response(base_phrase) – Performs all of the above responsibilities.

4.2.2.5.21. User

The essential role of this object is to represent and maintain all that is known about an individual authorized to use the system.

4.2.2.5.21.1. Responsibilities

- Supply user information.
- Confirm user information.

4.2.2.5.21.2. Attributes

- name – The user's full name.
- know_level – The student's former (or current, if logged in) knowledge level.
- user_id – The login name of this user.
- user_type – Student, system designer or teacher (future).

4.2.2.5.21.3. Actions

- get_type() – Returns user_type.
- get_know_level() – Returns know_level.
- set_know_level() – Changes know_level.
- match_user(user_id) – See if the input id matches this user's.
- get_user() – Returns the user_id of this user.

4.2.2.5.22. UserInterface

The essential role of this object is to centralize all I/O in order to maintain a consistent external interface, and to facilitate future upgrade to a more elaborate interface.

4.2.2.5.22.1. Responsibilities

- Output to screen and input from keyboard.

4.2.2.5.22.2. Attributes

- none.

4.2.2.5.22.3. Actions

- get_user_response() – Read a user's translation attempt.

- print_menu() – Offer a menu of choices to the user.
- get_menu_response() – Read a user's menu choice.

4.2.2.5.23. UserList

The essential role of this object is to represent and maintain all the valid users who can log onto the system.

4.2.2.5.23.1. Responsibilities

- coordinate/validate an attempted login.
- add/delete users.
- [supply user information?].

4.2.2.5.23.2. Attributes

- users – the collection of User objects which make up the list.
- size – the number of users currently authorized.

4.2.2.5.23.3. Actions

- establish connection() – Accepts an attempted login, validates that login name is in list, if student, queries knowledge level.
- add_user() – Creates a new User object, and adds it to the collection.
- delete_user(User) – Deletes a User object from the collection.

4.2.2.5.24. Word

This is an actual derived word which can be printed to the screen. Note that a BaseWord and WordDescriptor fully specify a given Word.

4.2.2.5.24.1. Responsibilities

- Maintain the text of the word and pertinent attributes.

4.2.2.5.24.2. Attributes

- string – the string of characters which comprise the word.
- word_attributes – saved along with the string for when another object needs to know.
- word_identifier – the dictionary key for this word.

4.2.2.5.24.3. Actions

- print() – Print self to screen.

4.2.2.5.25. WordDescriptor

The essential role of this object is to specify the attributes which indicate what derivative of a BaseWord is needed. (Not the rules, just the attributes, e.g. "plural", "present tense", etc.)

4.2.2.5.25.1. Responsibilities

- create derived Word from BaseWord.

4.2.2.5.25.2. Attributes

- word_attributes – these are specific to different types of words, for instance, a noun would have "gender", a verb would have "tense", etc.

4.2.2.5.25.3. Actions

- no_dynamics() – returns TRUE or FALSE depending on whether or not all of the attributes in the descriptor are specified.
- resolve_dynamics(DynamicWordDescriptor) – Attempts to define any unspecified attributes by using those in the input descriptor.
- get_attributes() – return the attribute settings.
- set_attributes(attributes) – set the explicitly specified input attributes.

4.2.3. CRC: Class-Responsibility-Collaboration Design

The CRC method is sometimes referred to as the 3W method, in reference to the authors Wirfs-Brock, Wilkerson and Wiener [Wirfs-Brock,1990]. This method can be summarized as the following:

- Identify the Classes in the system. This consists of further refinement and definition of the objects identified during the JOOA process.
- Determine the Responsibilities of each class. This is the continuing, iterative arbitration of responsibilities between the different classes.
- Determine which, and how, objects of these classes Collaborate with others. This is done in conjunction with the arbitration process, determining how classes interact, and what capabilities are provided to others, as the responsibilities stabilize. Classes may be grouped into functional subsystems, as well.

4.2.3.1. CRC terminology

Refer also back to section 4.2.2.1. for JOOA terminology, if needed.

3W Method – The design methodology described by Wirfs-Brock, et al. [Wirfs-Brock,1990] (aka CRC Method).

Class – This is a template or description of a type of object. A class describes all of the data and actions that an object of that class would have. *Instances* of a class are the actual objects.

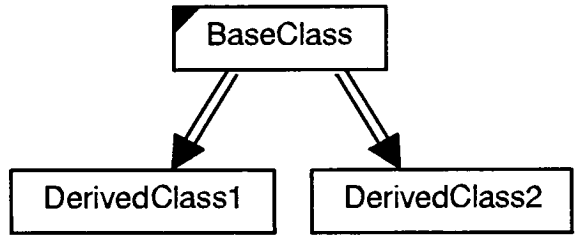
CRC Method – **C**lass-**R**esponsibility-**C**olaboration – the design methodology described by Wirfs-Brock, et al. [Wirfs-Brock,1990] (aka 3W Method).

Contract – The set of responsibilities which one class (server) provides for another class (client). The contract typically repre-

sents the set of member functions used by a client class, and it should be noted that a given class can be both a client and a server (to the same or different classes).

Inheritance – A class is said to "inherit" from another class when its definition is based on the first class, and as a result, it picks up some of the same attributes or functions. The class which inherits is referred to as a Derived Class, and the class it inherits from is referred to as a Base Class.

Inheritance Graph – This is the pictorial representation of classes which inherit attributes or functionality from other classes. A darkened corner on a class box indicates that it strictly exists to be inherited from, and that no actual objects of that class will ever be created (this is a virtual base class).



Responsibility – In this context, this refers to the responsibilities delegated to particular classes.

Collaboration – The way in which different classes work together to get the overall job done.

Subsystem – A logical collection of classes which, from some client perspective, can be considered as being a single object.

4.2.3.2. Subsystems (in alphabetical order)

4.2.3.2.1. Dictionary Subsystem

4.2.3.2.1.1. Purpose

- Supplies words to the rest of the system.

4.2.3.2.1.2. Classes Used

- Dictionary, The derived classes of DictionaryEntry – DE_Noun, DE_Verb, DE_Adjective, DE_Article.

Responsibility	Delegated to
Get a specified word	Dictionary

Get the derived form of the specified word	DictionaryEntry (and derived classes)

4.2.3.2.2. Exercises Subsystem

4.2.3.2.2.1. Purpose

- Provides exercise capability.

4.2.3.2.2.2. Classes Used

- PhraseModelList, PhraseModel, BasePhrasePool, BasePhraseList, BasePhrase, BasePhraseTranslator, TranslationTree, PatternNode, Hint, PhraseDescriptor.

Responsibility	Delegated to
External interface, collects and organizes all exercise support	PhraseModelList
A single phrase or sentence form, exercises and diagnosis	PhraseModel
Supplying a random BasePhrase from multiple lists	BasePhrasePool
Supplying a random BasePhrase	BasePhraseList
Semantically correct vocabulary information	BasePhrase
Supplying a BasePhrase of the opposite language	BasePhraseTranslator
Construction of expected response and initial diagnosis	TranslationTree
Subsequent diagnosis	PatternNode
A clue to the discrepancy at a given level	Hint
Syntactically correct model information	PhraseDescriptor

4.2.3.2.3. Log Subsystem

4.2.3.2.3.1. Purpose

- Storing, viewing and manipulating run-time gathered data.

4.2.3.2.3.2. Classes Used

- Log, The derived classes of LogEntry - LE_LeafNode, LE_Statistics.

Responsibility	Delegated to
Record data	Log
View data	Log
Store and manipulate a specific data record	LogEntry (and derived classes)

4.2.3.3. Classes (in alphabetical order)

4.2.3.3.1. BasePhrase

4.2.3.3.1.1. Purpose

- Specifies a semantically valid phrase. Most likely not syntactically correct.

4.2.3.3.1.2. Base Classes

- None.

4.2.3.3.1.3. Derived Classes

- None.

Responsibility	Collaborators
get_word_identifier	none
set_active	none

4.2.3.3.2. BasePhraseList

4.2.3.3.2.1. Purpose

- Collects BasePhrases of like length which can be made syntactically valid by application of the same Phrase Descriptor

4.2.3.3.2.2. Base Classes

- None.

4.2.3.3.2.3. Derived Classes

- None.

Responsibility	Collaborators
get_base_phrase	none
reset_selections	none
set_active	none

4.2.3.3.3. BasePhrasePool

4.2.3.3.3.1. Purpose

- Collects all BasePhraseLists containing BasePhrases of the same length, and which can be made syntactically valid by application of the same PhraseDescriptor.

4.2.3.3.3.2. Base Classes

- None.

4.2.3.3.3.3. Derived Classes

- None.

Responsibility	Collaborators
get_base_phrase	BasePhraseList
reset_selections	BasePhraseList
set_active	BasePhraseList

4.2.3.3.4. BasePhraseTranslator

4.2.3.3.4.1. Purpose

- Specifies the rules which describe how to translate a specific BasePhrase associated with a specific PhraseModel.

4.2.3.3.4.2. Base Classes

- None.

4.2.3.3.4.3. Derived Classes

- None.

Responsibility	Collaborators
translate	BasePhrase Dictionary

4.2.3.3.5. Dictionary

4.2.3.3.5.1. Purpose

- Represents the system's vocabulary – all the words that the system knows.

4.2.3.3.5.2. Base Classes

- None.

4.2.3.3.5.3. Derived Classes

- None.

Responsibility	Collaborators
get_word	DictionaryEntry

4.2.3.3.6. DictionaryEntry

4.2.3.3.6.1. Purpose

- Represents a given word in the dictionary. It represents all that the dictionary knows about a single base word (including how to construct other forms of the word, etc.)

4.2.3.3.6.2. Base Classes

- None.

4.2.3.3.6.3. Derived Classes

- DE_Noun, DE_Verb, DE_Article, DE_Adjective.

Responsibility	Collaborators
derive_word	Word
get_identifier() – Return the word's key.	none

4.2.3.3.6.4. Notes

- Additional derived forms will be added and documented as the system is enhanced.

4.2.3.3.7. DynamicWordDescriptor

4.2.3.3.7.1. Purpose

- Facilitates the propagation of word attributes for words which must agree with one another.

4.2.3.3.7.2. Base Classes

- None.

4.2.3.3.7.3. Derived Classes

- None.

Responsibility	Collaborators
get_attributes	none
resolve_dynamics	Word

4.2.3.3.8. Hint

4.2.3.3.8.1. Purpose

- Assists the student when asked, or in certain error cases. It is basically a block of text, associated with a given Pattern.

4.2.3.3.8.2. Base Classes

- None.

4.2.3.3.8.3. Derived Classes

- None.

Responsibility	Collaborators
print	UserInterface

4.2.3.3.9. Log

4.2.3.3.9.1. Purpose

- Stores and services accesses to run-time logged data.

4.2.3.3.9.2. Base Classes

- None.

4.2.3.3.9.3. Derived Classes

- None.

Responsibility	Collaborators
add_entry	LogEntry
set_current_user	none
set_current_presentation_phrase	Phrase
set_current_model_index	none
manipulate_entries	LogEntry
save	LogEntry

4.2.3.3.10. LogEntry

4.2.3.3.10.1. Purpose

- Captures a significant piece of data for further review. There will be various forms of log entries, with data specific to the type of incident being logged.

4.2.3.3.10.2. Base Classes

- None.

4.2.3.3.10.3. Derived Classes

- LE_Statistics, LE_LeafNode

Responsibility	Collaborators
print	Statistics Phrase
report	Statistics Phrase

4.2.3.3.10.4. Notes

- Additional derived forms will be added and documented as the system is enhanced.

4.2.3.3.11. Pattern

4.2.3.3.11.1. Purpose

- This is essentially a Phrase that contains wildcards, and which can match itself against other phrases.

4.2.3.3.11.2. Base Classes

- None.

4.2.3.3.11.3. Derived Classes

- None.

Responsibility	Collaborators
match	
get_word	
set_word	

4.2.3.3.12. PatternConstructor

4.2.3.3.12.1. Purpose

- Generates a pattern from a given phrase. It may contain PatternFragmentConstructors for more detailed pieces using Words.

4.2.3.3.12.2. Base Classes

- None.

4.2.3.3.12.3. Derived Classes

- None.

Responsibility	Collaborators
build_pattern	PatternFragment Constructor Dictionary Phrase Word

4.2.3.3.13. PatternFragmentConstructor

4.2.3.3.13.1. Purpose

- Generates a pattern fragment from a given Word. It contains rules specifying how much of the word to use, what wild cards to inject, etc.

4.2.3.3.13.2. Base Classes

- None.

4.2.3.3.13.3. Derived Classes

- None.

Responsibility	Collaborators
build_pattern_fragment	Dictionary Word

4.2.3.3.14. PatternNode

4.2.3.3.14.1. Purpose

- Recognizes and handles a specific type of error in a student's response. It will be a single entry in a binary tree of Patterns which support a given PhraseModel. It contains information on how to construct the needed matching Pattern, as well as which PatternNodes to move on to, based on whether or not the user response matches at this point.

4.2.3.3.14.2. Base Classes

- None.

4.2.3.3.14.3. Derived Classes

- None.

Responsibility	Collaborators
process_response	Pattern PatternConstructor UserInteface PatternNode Log
more_hints	Pattern PatternConstructor UserInteface PatternNode Log

4.2.3.3.15. Phrase

4.2.3.3.15.1. Purpose

- This is an actual semantically and syntactically correct text phrase which can be printed to the screen. Note that a BasePhrase and PhraseDescriptor fully specify a given Phrase.

4.2.3.3.15.2. Base Classes

- None.

4.2.3.3.15.3. Derived Classes

- None.

Responsibility	Collaborators
print	Word
read	Word
get_word	none
set_word	none

4.2.3.3.16. PhraseDescriptor

4.2.3.3.16.1. Purpose

- Specifies the syntactic requirements of a given Phrase. A BasePhrase and a PhraseDescriptor contain all the information necessary to create a semantically and syntactically correct text Phrase.

4.2.3.3.16.2. Base Classes

- None.

4.2.3.3.16.3. Derived Classes

- None.

Responsibility	Collaborators
build_phrase	Word BasePhrase Phrase WordDescriptor

4.2.3.3.17. PhraseModel

4.2.3.3.17.1. Purpose

- Models everything that the system knows about a specific semantic model phrase – what BasePhrases may be used in it, how to translate it, how to parse and diagnose the user's response, etc.

4.2.3.3.17.2. Base Classes

- None.

4.2.3.3.17.3. Derived Classes

- None.

Responsibility	Collaborators
set_active	BasePhrasePool
do_exercise	BasePhrasePool PhraseDescriptor Log Statistics Phrase TranslationTree
reset_depletions	BasePhrasePool

4.2.3.3.18. PhraseModelList

4.2.3.3.18.1. Purpose

- Collects all the semantic model phrases (plus associated exercises, etc.) which the system currently supports.

4.2.3.3.18.2. Base Classes

- None.

4.2.3.3.18.3. Derived Classes

- None.

Responsibility	Collaborators
do_exercise	PhraseModel
set_active	PhraseModel
reset_depletions	PhraseModel

4.2.3.3.19. Statistics

4.2.3.3.19.1. Purpose

- Does run-time statistics gathering and processing.

4.2.3.3.19.2. Base Classes

- None.

4.2.3.3.19.3. Derived Classes

- None.

Responsibility	Collaborators
update_counts	none

print_summary	UserInterface

4.2.3.3.20. TranslationTree

4.2.3.3.20.1. Purpose

- This object is the beginning of the entire diagnosis-side of a given PhraseModel.

4.2.3.3.20.2. Base Classes

- None.

4.2.3.3.20.3. Derived Classes

- None.

Responsibility	Collaborators
check_response	BasePhraseTranslator
	PhraseDescriptor
	Phrase
	Statistics
	Pattern
	PatternNode

4.2.3.3.21. User

4.2.3.3.21.1. Purpose

- Represents and maintains all that is known about an individual authorized to use the system.

4.2.3.3.21.2. Base Classes

- None.

4.2.3.3.21.3. Derived Classes

- None.

Responsibility	Collaborators
get_type	none
get_know_level	none
set_know_level	none
match_user	none
get_user	none

4.2.3.3.22. UserInterface

4.2.3.3.22.1. Purpose

- Centralizes all I/O in order to maintain a consistent external interface, and to facilitate future upgrade to a more elaborate interface.

4.2.3.3.22.2. Base Classes

- None.

4.2.3.3.22.3. Derived Classes

- None.

Responsibility	Collaborators
get_user_response	Phrase
print_menu	none
get_menu_response	none

4.2.3.3.23. UserList

4.2.3.3.23.1. Purpose

- Represents all the valid users who can log onto the system.

4.2.3.3.23.2. Base Classes

- None.

4.2.3.3.23.3. Derived Classes

- None.

Responsibility	Collaborators
establish connection	User
add_user	User
delete_user	User

4.2.3.3.24. Word

4.2.3.3.24.1. Purpose

- This is an actual derived word which can be printed to the screen. Note that a BaseWord and WordDescriptor fully specify a given Word.

4.2.3.3.24.2. Base Classes

- None.

4.2.3.3.24.3. Derived Classes

- None.

Responsibility	Collaborators
print	none

set_string	none
get_attributes	none
set_attributes	none

4.2.3.3.25. WordDescriptor

4.2.3.3.25.1. Purpose

- Specifies the attributes which indicate what derivative of a BaseWord is needed. (Not the rules, just the attributes, e.g. "plural", "present tense", etc.)

4.2.3.3.25.2. Base Classes

- None.

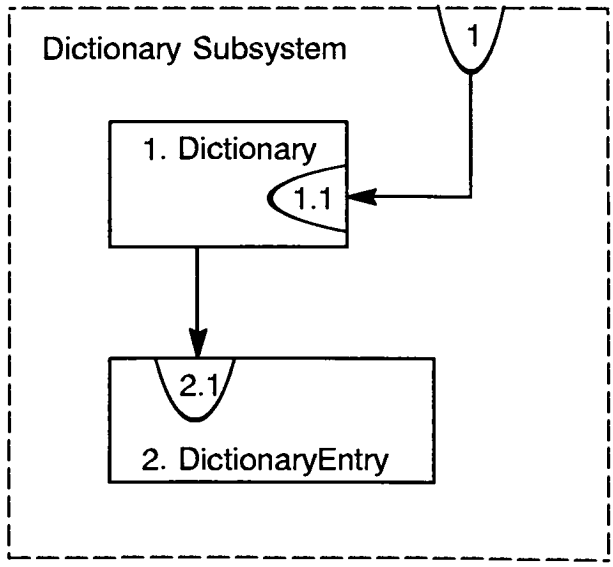
4.2.3.3.25.3. Derived Classes

- None.

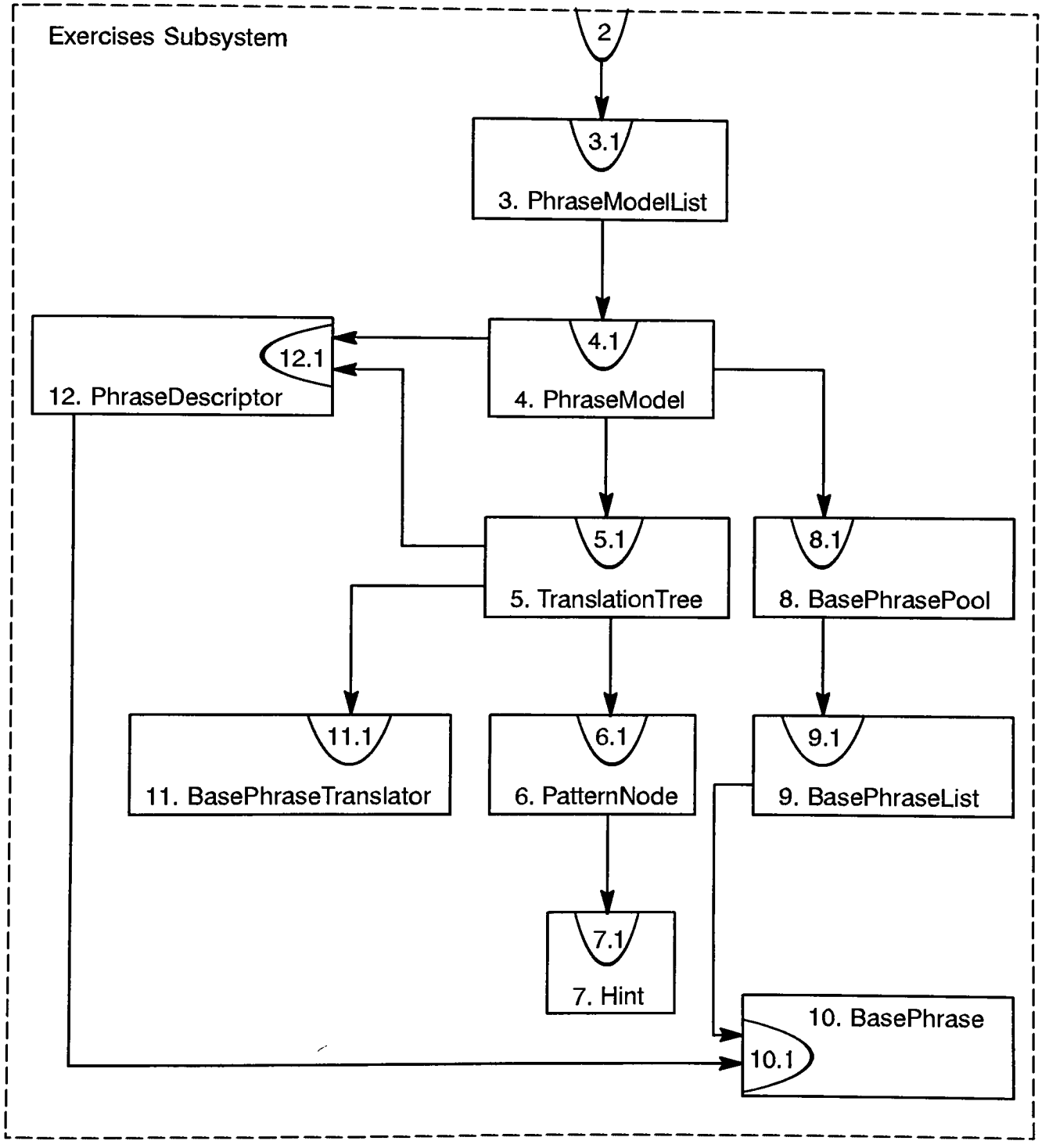
Responsibility	Collaborators
no_dynamics	none
resolve_dynamics	DynamicWord Descriptor
get_attributes	none
set_attributes	none

4.2.3.4. Collaboration Graphs

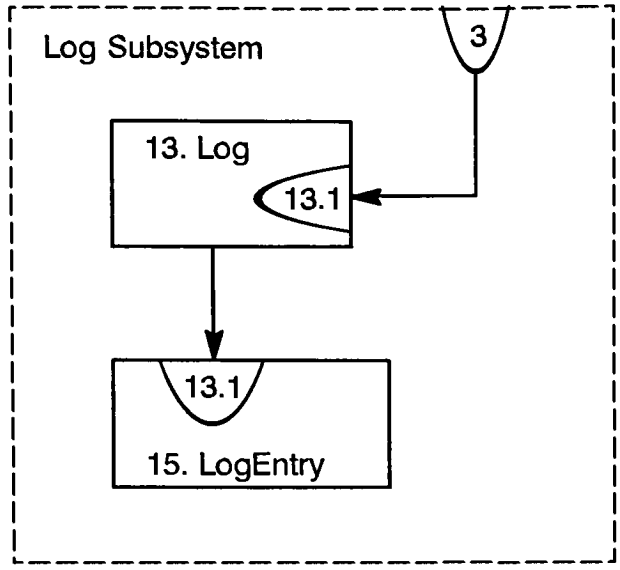
4.2.3.4.1. DictionarySubsystem



4.2.3.4.2. Exercises Subsystem



4.2.3.4.3. Log Subsystem



4.2.3.5. Contract Listings

4.2.3.5.1. Subsystem Contracts

(Numbers refer to Collaboration Graphs above.)

1. Get word from dictionary.
2. Set active knowledge level, do exercise.
3. Set current session information, log data, view data, generate report.

4.2.3.5.2. Class Contracts in Subsystems

(Numbers refer to Collaboration Graphs above.)

1. Dictionary
   1.1 Get word from dictionary.
2. DictionaryEntry
   2.1 Get index, derive word.
3. PhraseModelList
   3.1 Set active knowledge level, do exercise.
4. PhraseModel
   4.1 Set active knowledge level, do exercise, reset depletions
5. TranslationTree
   5.1 Check student response.
6. PatternNode

6.1 Process response.

7. Hint

   7.1 Print.

8. BasePhrasePool

   8.1 Set active knowledge level, get BasePhrase, reset selections.

9. BasePhraseList

   9.1 Set active knowledge level, get BasePhrase, reset selections.

10. BasePhrase

   10.1 Get index.

11. BasePhraseTranslator

   11.1 Translate.

12. PhraseDescriptor

   12.1 Build Phrase.

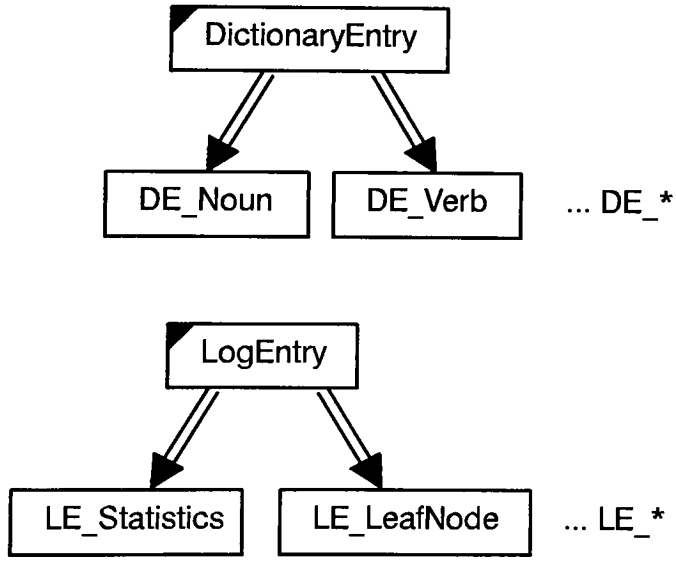
13. Log

   13.1 Set current session information, log data, view data, generate report.

14. LogEntry

   14.1 Print.

4.2.3.6. Inheritance Graphs



4.2.4. Degree of Analysis and Design Goal Satisfaction

The design was completed and documented in an object-oriented fashion. What is documented here is the result of many iterations. The design supports all of the specified

functionality, and the C++ implementation was fairly straightforward. It is not clear that all of the touted benefits of object-oriented design have been realized. It is quite modular, and as intuitive as the chosen parsing algorithms allow it to be. It would seem to lend itself to enhancement. Reusability (or lack thereof) is one area where the design may fall short. Along the way, decisions were made on what responsibilities objects would need to support, and what the division of objects would be. It was not until the implementation phase, though, that the pervasiveness of some of the concepts of the system were appreciated. The knowledge level, for instance, touches nearly every object in one way or another. It would have been nice to isolate such tutoring system-specific needs from the generic word, phrase, etc., processing. Tim Nichols, who promotes the JOOA methodology, advises beginning object designers to tackle a project strictly as a learning exercise, and then, once it reaches the implementation phrase, scrap it entirely and chalk up the time to the design learning curve. It would be nice to have that luxury, because a new design would surely come out much cleaner.

4.3. Knowledge Base Design

It was a design goal for this system to support the gathering of data which would allow its diagnosis ability to be improved. Since this is the best way to find out what type of responses will be encountered, it is not unlikely that, over time, the internal knowledge base reflect more of this feedback than of the original design. This is not unreasonable, or unexpected, but it is still necessary to put in place an initial knowledge base from which to work. This involved identifying several things: the phrase/sentence models to be supported, the vocabulary to be supported for different levels, the types of errors which the system would be able to diagnose for a given model, and the processing details to support the different types of errors. The following sections describe the process of establishing the initial knowledge base and knowledge-specific processing.

4.3.1. Knowledge Base Design Goals

As alluded to above, the primary goal of the initial knowledge design was to put something in place which would allow the system to be demonstrated, run, and capable of gathering data for future enhancement. It was also a goal to have the diagnosis be correct (for those errors handled), and for multiple levels of user knowledge to be supported.

4.3.2. Phrase Model Selection

As stated earlier, this incarnation of the tutoring system is highly tied to a specific textbook. It was also decided that "chapter completed" would be a reasonable definition of knowledge level, at least for the purpose of implementing the prototype system. The textbook chosen was German Made Simple [Jackson, 1985]. This choice was made for several reasons. It was the textbook for two introductory German courses taken as information gathering for this project. This made it a common denominator for the sys-

tem designer, the course instructor, who served as one of the expert consultants for the system, and for former classmates from those classes who acted as a test audience for the system.

The selection of the phrase models implemented resulted from identifying the various models introduced in early chapters of the text, and narrowing it down to a few which illustrated important features of the system. The random selection of models and vocabulary is illustrated merely by having enough different choices built in to the initial knowledge base. Other features are described in the sections below.

4.3.3. Vocabulary Selection

Vocabulary selection was fairly straightforward, once phrase models were selected. Each chapter introduces a finite set of vocabulary. Valid vocabulary choices for the different phrase models were compiled from each vocabulary set. The knowledge level associated with an exercise becomes the higher of the phrase model (chapter in which the sentence form was introduced) and the highest level of any individual word (chapter in which that word was introduced). This allows older (lower level) vocabulary to be used in more advanced phrase models, and more advanced vocabulary to be used in the earlier phrase models.

4.3.4. Base Phrase List and Pool Compilation

This is an offshoot of the vocabulary selection process. Since some phrase models share common lengths and word types, some base phrases can be shared between models. Conceptually, each model could have a single list, but in some cases, only a portion of that list is valid for another phrase model. The added layer of the pool supports all of the choices for a given model, and the list is the mechanism for sharing subsets of choices between models. A given base phrase should only appear in a single list. If a situation arises, where a base phrase appears to be needed in more than one list, this indicates the need to break up the lists.

4.3.5. Diagnosis Tree Design

This is done on a per-phrase model basis. The types of errors to be diagnosed for the given model are identified first. These come from the topics covered in the text where this phrase form is introduced, from previous concepts introduced, from predicting potential incorrect responses, and will be verified and enhanced by run-time information gathering. An example individual diagnosis tree is described in section 4.3.5.2.

4.3.5.1. Documentation Convention

As notational shorthand, several pieces of information are represented in the "Errors Diagnosed" section below. The emboldened lines indicate the specific concepts which

are tested by the given model. The indentation represents the sequential nature of the diagnosis, using the binary tree. For example, in section 4.3.5.2.2. below, it can be seen that it is first established that something is wrong with the first word, then established whether or not it is a gender-agreement problem. With some imagination, the binary tree can be visualized. Starting with any line, a left branch (the no-match path) can be taken by falling down to the next line which is at the same indentation level. A right branch (the match path) can be taken by going to the next contiguous line of increasing indentation. If there is no line with the same indentation level to fall to, the current line represents a left leaf node, and if there is no contiguous line of greater indentation, the current line represents a right node. The path followed on the way to an emboldened line indicates the diagnosis of a particular concept. The path which can be followed from an emboldened line, if any, indicates further diagnosis which the system is capable of (typically for giving the user hints).

4.3.5.2. Example Diagnosis Tree

4.3.5.2.1. Form of Model

The phrase represented by this model is a complete sentence taking the form "direct article, singular noun, present tense verb, adjective".

Knowledge level of model - 2.

Example phrase - The man is good.

Expected response - Der Mann ist gut.

4.3.5.2.2. Error Tree

- First word is different from expected
  - **Gender of article does not agree with noun**
    - Gender variation 1 (see section 4.4.2.2.)
    - Gender variation 2
  - Expecting a direct article
- Second word is different from expected
  - **Noun is not capitalized**
  - **Synonym of noun used**
- Third word is different from expected
  - **Tense of verb is incorrect**
  - **Synonym of verb used**
- Fourth word is different from expected
  - **Synonym of adjective used**
- First and second words different from expected
  - **Synonym of noun used**

4.3.5.2.3. Notes Concerning This Model

Note that use of a synonym for the noun appears twice in the diagnosis hierarchy. This is because an incorrect (or synonymous) noun may be of a different gender, which might also cause the article to be different from that expected. The types of errors initially diagnosed are taken from the text. Feedback from the system's run-time gathered data may lead to enhancements of the tree.

4.3.6. Degree of Knowledge Base Design Goal Satisfaction

The initial knowledge base, illustrated by the sample system runs in the thesis report, supports all of the proposed features of the system. It provides a base for information gathering, and future enhancement should merely involve more of same process, with increasing numbers of selection choices.

4.4. Implementation

The system was implemented in C++. Prolog and C were used for earlier prototypes, and a variety of logic, pattern matching and object-oriented languages were considered for the actual thesis project. A language which supported object-oriented constructs was selected in order to ease the implementation of the object-oriented design. Without arguing the relative merits of the various object-oriented languages (or, for that matter, the merits of "pure" object-oriented languages versus hybrid languages), it is

safe to observe that C++ is becoming the defacto standard for commercial object-oriented implementations. It's availability on a wide variety of platforms was a major factor in Cadre Technologies decision to choose C++ for their object-oriented re-implementation of the popular Teamwork CASE tool [Wybolt-1990].

The tutoring system was implemented on Sun workstations, running SunOs v4.1, and compiled using Sun C++ v2.1.

4.4.1. Implementation Goals and Methodology

The primary goal of the implementation was to produce a functional system which displayed the promised functionality, without deviating drastically from the design as documented here. The method was an incremental approach. The Session agent was implemented as an empty driver only. Establishing a connection with the user, doing exercises, and logging run-time data were all stubs. Objects were coded one at a time, top-down, and placed in the system with the necessary stubs to allow it to be tested. As knowledge base objects were added, they were hard-coded with the most primitive amount of data to allow it to be used by the rest of the system. A major milestone in the implementation was the point at which the entire functional path of the system could be executed - login, exercise, diagnosis, data logging, logout. At that point the effort turned mainly to the addition of more and different knowledge.

4.4.2. Implementation Overview

One important detail that was deferred until the late design and early implementation phase was the actual method to be used for constructing the dynamic patterns needed for the binary diagnosis tree. In the analysis and design phases, this was represented by the PatternConstructor class/object, without specifying a great deal of detail about how the different and specific pattern needs would be handled. Early in the implementation phase, this was an object, but it quickly became apparent that it did not really meet the definition of an object. An object generally has a state, some data, and some things that it can do to/with that data. With the PatternConstructor, there was no clear way to have both data and a function. The pattern construction rules were not generic enough to permit them to be member data, used by a generic pattern building member function. The second approach was to make the PatternConstructors member functions of the PatternNode. Each Node would then know how to make the pattern it needed. This would work fine, but it was also true that certain Nodes, and even different PhraseModels, had a need for the same constructor in different places. Finally, then, the PatternConstructors became utility functions. Each is implemented standalone, and PatternNodes maintain a pointer to the PatternConstructor function which fits its need. The function names are all prefixed with PC_ to indicate their use.

Besides just the manifestation of the PatternConstructor objects or functions, it was also a late decision on how to actually implement the construction. This was a potentially

messy programming area, and it was fundamental to the pattern masking diagnosis technique to be able to abstract this. The solution involved using the standard library function regexp(), and implementing pattern constructors which built patterns which were regular expressions. The actual pattern constructors are surprisingly small, most taking no more than a few lines of code. Two examples are included here to illustrate the implementation.

4.4.2.1. Pattern Constructor - Unexpected Word

Various static pattern fragments or regular expressions are defined externally to be used by phrase constructors, as needed. This constructor has a need for a match-a-single-word wildcard, so it uses the following:

```
typedef  char *  pattern_fragment;

pattern_fragment  any_word = "[a-zA-Z][a-zA-Z]*";
```

The following is the complete constructor function which generates the pattern to determine if the second word of a phrase is different from what was expected.

```
Pattern  PC_unexpected_second_word(Phrase & in_phrase)
{
        Pattern         new_pattern = in_phrase;
        Word            new_word;

        new_word.set_string(any_word);
        new_pattern.set_word(2, new_word);      // First argument is word
                                                // position in Phrase.
        return(new_pattern);
}
```

The creation of new_pattern works, because the Pattern object has a constructor which tells it how to make a Pattern from a Phrase. The pattern created is a regular expression which can be matched against the student response.

4.4.2.2. Pattern Constructor - Gender Variation

This example does not use any pre-defined pattern fragments. It illustrates a pattern constructor which uses the dictionary to create a variation of the expected phrase. This type of pattern is typically used in the tree when it is trying to isolate or diagnosis a potential anticipated error.

```
Pattern PC_first_gender_var_first_word(Phrase & in_phrase)
{
        Pattern         new_pattern = in_phrase;
```

```
        Word        old_word = in_phrase.get_word(1);  // Argument is word
                                                       // position in Phrase.
        WordDescriptor      new_word_WD(old_word);
        int                 word_identifier = old_word.get_index();
        Gender              new_gender = new_word_WD.get_gender();

        new_gender = (Gender) ((((int) new_gender) % (G_none-1)) + 1);
        new_word_WD.set_gender(new_gender);
        Word  new_word = dictionary.get_word( word_index, new_word_WD );
        new_pattern.set_word(1,new_word);

        return(new_pattern);
}
```

This function could no doubt use a bit more explanation than the previous, but really it is not doing anything too difficult. This is one of the most complex pattern constructors in the system currently.

As in the previous example, the initial pattern is merely the input Phrase. The function then gets the current first word from the input Phrase. A WordDescriptor is created from this word (the WordDescriptor containing the feature attributes of the word), and the integer word identifier is obtained. (A word identifier and a WordDescriptor are the two things necessary for getting words from the dictionary. The final piece of set-up involves getting the current Gender setting from the WordDescriptor. The Gender, and most attributes, are simply enumerated constants which represent the various values that the attribute can hold. The Gender definition looks like:

```
enum Gender
{
        G_DYNAMIC,
        G_masculine,
        G_feminine,
        G_neuter,
        G_none
};
```

The next cryptic-looking line in the example function is just calculating, what is internally known as, the first gender variation. It amounts to the next gender setting modulo the valid value. If the input gender had been masculine, the new gender would be feminine. If the input gender had been neuter, the new gender would be masculine. This new gender attribute is then set in the WordDescriptor, and the dictionary is called on to supply the new derived form of the indicated word. This new word is set in the pattern, and the pattern is complete. If the input phrase is "Der Mann..." the output pattern would be "Die Mann...".

5. CONCLUSIONS

5.1. Problems Encountered and Solved

Some implementation and portability problems were experienced. The standard library regular expression functions (regexp(), etc.), work as documented under Sun C++ v2.1, but do not compile properly under AT&T C++ v2.0. This was resolved in the simplest way – Sun C++ was used instead of AT&T C++. Problems were also encountered with the standard library time() function. This is used to seed the pseudo-random number generator for selecting phrase models and vocabulary. Under certain circumstances, invoking time would cause a segmentation fault. This turned out to be just a user error of passing an uninitialized pointer to the function, illustrating that it is just as easy to have pointer problems in C++ as it was in C. A final implementation problem involved getting dynamic binding to work. This is done with a compiler switch, but because of my date-dependent build procedures I was attempting to use the feature when not all modules had been compiled for it.

One problem which was wrestled with off-line was the issue of representing modified vowels, for example the u-umlaut in "grün". This problem is obviously not unique to translation programs. It is also an issue in areas where German is the first language, in relation to computers and various print media. For computers that support "compose character", and display screens that support special characters, the actual modified vowel could be displayed. However, "compose character" is not a friendly operation for the user who must type it. There are three alternative conventions which can be used. The first approximates the umlaut by preceding the modified vowel with a double quote, e.g. gr"un. The second approximates the umlaut as if it was rotated 90 degrees clockwise, by following the vowel with a colon, e.g. gru:n. The third, and most generally accepted convention, is to append an 'e' to the modified vowel, e.g. gruen. This convention was chosen, but in demonstrating the system, and explaining the problem, a number of users expressed a preference for the second convention. This issue might bear re-visiting in a future version (see section 5.3.1., below).

5.2. Discrepancies and Shortcomings of the System

The system demonstrates most of the functionality described earlier in this report. One notable discrepancy is that, currently, the system only translates in one direction. It presents phrases in English, and reads and diagnoses German phrases. It is not believed that any major concepts were missed by implementing only a single direction, in fact, since the system attempts to diagnose German phrases, it must deal with language attributes, such as nouns with gender, which it would not have had to do in the opposite direction.

The user interface is mentioned below as an area where future enhancement is possible. It is worth noting as a discrepancy, also, that the current user interface is a bit fragile.

Unexpected input, such as escape sequences, will crash the system. A control–C sequence will abort the program. It would be desirable to prevent these situations so that the current run–time data log is not lost. From a user–friendliness standpoint, the user-interface exhibits a few other undesirable actions. After a bad menu selection, the menu is not re–displayed for the user. The '?' for "help" is not implemented at the phrase prompt – the user must answer incorrectly in order to see a menu of options. Finally, when asking for subsequent hints after an incorrect response, it is possible for the user's response to scroll off the screen (making it impossible for the user to refer back to it). None of these problems would be difficult to fix, but they were dropped as lower priority issues when time got tight.

Earlier in the report, synonyms were mentioned in the context of a translation tree. It was originally intended that the system would support some degree of alternative translation. Fundamental to this would be the ability of the system to handle synonymous words. This capability, however, did not make it into the prototype system. One means for handling this, would be to have the Word object maintain a data element which indicates synonyms (e.g. via word indices).

Several of the report–generation and file manipulation options shown in the system administration menu in the Sample Runs section are stubbed. The log file, however, is created, and the "view" options function as illustrated.

Additional capability needs to be added to the current DictionaryEntry types. The verb, for example, is basically stubbed. It would need to support additional tenses and variations, in a similar fashion to the current article and noun objects. Also, certain word types are not implemented at all, e.g. the adverb.

The final shortcoming which would prevent this system from truly being an effective tutoring aid, for a class such as the one on which it was based, is its depth of knowledge. It requires the expansion of its knowledge base to cover at least a full semester's worth of vocabulary and sentence forms. This increase in scope and volume would be necessary to give the tool some longevity with the students, as well as to prevent the repetition of exercises too quickly

5.3. Suggestions for Future Work

The following subsections describe other less–essential enhancements, as well as possible project offshoots.

5.3.1. Potential Enhancements

Incorporating the basic enhancements described in section 5.2. would make the system useful for a particular course with a particular textbook. It would be nice, though, instead of hard–coded patterns, vocabulary and knowledge level, if a teacher would be able to customize the system to some extent (ideally, with-

out re-compilation). One of the easiest variations of this capability would be to allow the teacher to set the knowledge level for the vocabulary and sentence forms. This would allow the system's behavior to be adjusted to a different textbook, or to a different resolution of knowledge level (e.g. sections within chapters, instead of whole chapters). A more ambitious variation of this capability would be to allow the teacher to define new sentence forms, choose vocabulary, and "program" the diagnosis by entering or modifying diagnosis trees. These capabilities could be implemented as a new type of user. Besides "student" and "system administrator", there might then also be "teacher".

Neither the documentation conventions in this report, nor the actual source code, are particularly user-friendly ways to view and study the diagnosis trees. A feature which would be useful for future development, and mandatory for adding teacher-configuration features, is a pattern tree browser. This would allow an interactive user to display and traverse the various pattern trees which the system supports internally.

A possible enhancement to improve the system's value to students, might be to examine including some lessons which relate to the concepts being tested/diagnosed by the system. If a student has trouble with an exercise, the system could offer a lesson on the concept involved (e.g. gender agreement) in addition to its current options of a hint or the correct answer.

The user interface for the current prototype is neither flashy, nor robust. This is an area where the system could be improved. The approach here could take two separate tacks, the easier of which being just a flashier front-end to make the system appear more professional or marketable. A more substantial direction, though, would be to research ways in which the user interface can be improved in order to make the system more effective. This could encompass not only the look and feel of the system, but also the manner in which exercises are presented, the timing of presentation, and other factor's which would aid student comprehension.

Increased depth of knowledge of the current language was mentioned as a basic enhancement. This could also be a topic of advanced research, though. The present system was specifically intended as a prototype, and specifically targeted at the beginning student. It remains to be seen to what level the approach could be taken, or if it would fail to work for the more esoteric language issues encountered.

If the current prototype is to be expanded, it might bear examining the availability of other regular expression packages (besides the UNIX version which is currently used). The main shortcoming of the current package is that it does not support alternation (the "or" function, or '|'). This would help tremendously to improve the ease with which different diagnosis nodes could be incorporated.

The system does not currently attempt to adapt its behavior based on a student's history (previous executions of the system, or earlier exercises in the current

session). It may be possible for the system to draw some conclusions based on its diagnosis, for example if a certain type of error is made frequently. The system could then offer lessons, or choose exercise which will help to illustrate and reinforce the concept.

As mentioned earlier, there was a bit of an issue around the system's representation of modified vowels (alternatives to using the umlaut). Three alternative conventions were presented in section 5.1. It is not clear that changing the current approach would be an enhancement, but what might be would be the ability for the teacher to choose the convention which best suits the class. (This may not be a light undertaking, though, since it does have knowledge base and translation implications).

5.3.2. Related Projects

This project, or selected concepts employed, could be applied to similar tutoring systems to support other languages. The current system could be expanded to support multiple languages, but it is not clear that the amount of commonality shared would justify the added size, complexity, and potential run time impact of combining them. Perhaps a better way to tie them together, if desired, would be through the use of a common front-end interface program. It is anticipated that that the concepts of this system would translate fairly well to other western (i.e. right-branching) languages, but its applicability to eastern (i.e. left-branching) languages would need to be examined in more detail.

The current system is fairly self-contained. While this works for the current scope, or even for an enhanced system with a complete semester's worth of material, it does not lend itself as well to some of the more advanced enhancements suggested above. An interesting project would be to take the current system for its core functionality, and attempt to integrate it with external components, such as a commercial dictionary, morphological processor, etc.

A final related project might be to incorporate this system into a larger, multi-exercise system, or adding additional types of exercises to this system. This might include expanding (or narrowing) the current type of exercise to produce others. For instance, an easy variation would be the one-word phrase, amounting to a vocabulary quiz exercise. Alternatively, the phrase translation exercise could become one of multiple choices of types of exercises available.

6. GLOSSARY

Some of the terms below, related to learning/teaching/tutoring with computers, are used interchangeably by some authors. The definitions may be reversed by others. In older research works, certain terms may appear because others had not yet been introduced. The definitions below reflect my usage, and will be consistent with most current writings in this area.

CAI – See Computer–Assisted Instruction.

CAL – See Computer–Assisted Learning.

CALI – See Computer–Assisted Language Instruction.

CALL – See Computer–Assisted Language Learning.

Computer–Assisted (or Aided) Instruction (CAI) – This typically refers to the use of computers and educational software as learning aids by students (See also CAL and ITS).

Computer–Assisted (or Aided) Language Instruction (CALI) – A subset of CAI, specific to foreign language subjects.

Computer–Assisted (or Aided) Language Learning (CALL) – A subset of CAL, specific to foreign language subjects.

Computer–Assisted (or Aided) Learning (CAL) – This is the broadest category. In addition to the use of computers and educational software as learning aids by students (CAI), it includes the use of computers, educational software, and general–purpose software by teachers as tools for class management, testing and/or preparation of materials for students.

Drill–and–Practice – A mode of behavior for a CAI system in which it operates like electronic flashcards. The system presents a problem, the student answers, the system presents the next problem, etc.

Expert System – "... an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. Knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners of the field." [Feigenbaum–1985]

ICAI – See Intelligent Computer–Assisted Instruction.

ITS – See Intelligent Tutoring System.

Intelligent Computer–Assisted (or Aided) Instruction (ICAI) – This term is a bit blurry, and is not commonly used anymore. It implies student– oriented software that exhibits some degree of intelligence. It would presumably fit somewhere between CAI and ITS.

Intelligent Tutoring System (ITS) – A CAI system which in some way attempts to provide intelligent diagnostics or lessons based on student input.

7. BIBLIOGRAPHY

Boyd, Gary, Arnold Keller and Roger Kenner (1982), "Remedial and Second Language English Teaching Using Computer-Assisted Learning", Computers and Education, Vol 6, pp 105-112.

Burton, R. R. (1981a), "An Investigation of Computer Coaching for Informal Learning Activities", in Intelligent Tutoring Systems, Academic Press, pp. 79-97.

Burton, R. R. (1981b), "Diagnosing Bugs in Simple Procedural Skills", in "Intelligent Tutoring Systems", Academic Press, pp. 157-182.

Byrd, Roy J. (1983), "Word formation in natural language processing systems", Proceedings of IJCAI-VIII, pp. 704-706.

Byrd, Roy J., G. Neumann and K. S. B. Andersson (1986), "DAM - A Dictionary Access Method", IBM Research Report, IBM T. J. Watson Research Center, Yorktown Heights, New York.

Byrd, R. J. and N. Calzolari, M. S. Chodorow, J.L. Klavans, M. S. Neff, and O. A. Rizk (1987), "Tools and Methods for Computational Lexicology," *Computational Linguistics*, vol. 13, numbers 3-4, July-December, pp. 219-240.

Byrd, Roy J. and Tzoukermann, Evelyne (1988), "Adapting an English Morphological Analyzer for French", 26th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference.

Cameron, Keith (1989), Computer-Assisted Language Learning, Ablex Publishing Corporation, Norwood, New Jersey.

Cameron, K.C. and W.S. Dodd and S.P.Q. Rahtz, eds. (1986), Computers and Modern Language Studies, Ellis Horwood Limited.

Collett, J. (1982), "Getting in the Right Mood: A CAI Program on the Subjunctive in French", Computers and the Humanities, Vol. 16, pp. 137-143.

Dhaif, H. A. (1990), "Computer-Assisted Language Learning: A Client's View", CALICO Journal, Vol 7(4), pp. 67-81.

Farrington, Brian (1982), "Computer-Based Exercises for Language Learning at University Level", Computers & Education, Vol. 6, pp. 113-116.

Farrington, Brian (1986), "Computer-Assisted Learning or Computer-Inhibited Acquisition", in Computers and Modern Language Studies, Cameron, K. C. and W. S. Dodd and S. P. Q. Rahtz, eds., Ellis Horwood Limited.

Ferney, Derrik (1989), "Small Programs That "Know" What They Teach", in Computers and Modern Language Studies, Cameron, K. C. and W. S. Dodd and S. P. Q. Rahtz, eds., Ellis Horwood Limited.

Fox, Jeremy (1989), "Can Computers Aid Vocabulary Learning?", in Computer-Assisted Language Learning, Cameron, Keith, ed., Ablex Publishing Corporation, Norwood, New Jersey.

Goldstein, I. P. (1981), "The Genetic Graph: A Representation for the Evolution of Procedural Knowledge", in Intelligent Tutoring Systems, Academic Press, pp. 51-75.

Hartley, J. R. and D. H. Sleeman (1973), "Towards Intelligent Teaching Systems", International Journal of Man-Machine Studies, Vol. 5, pp. 215-236.

Heidorn, G. E. and K. Jensen, L. A. Miller, R.J. Byrd, and M. S. Chodorow (1982), "The EPISTLE Text-Critiquing System," IBM Systems Journal, vol. 21, pp. 305-326.

Inwood, Clifford (1992), "Analysis Versus Design - Is There a Difference", *The C++ Journal*, Vol. 2, No. 1.

Jackson, Eugene and Adolph Geiger (1985), *German Made Simple*, Doubleday, textbook for St. John Fisher College's Beginning German I and II courses.

Kline, Paul J. and Steven B. Dolins (1985), "Choosing Architectures for Expert Systems", Rome Air Development Center.

Koenig, Andrew (1992), "Checklist for Class Authors", *The C++ Journal*, Vol. 2, No. 1.

LaReau, Paul and Edward Vockell (1989), The Computer in the Foreign Language Curriculum, Mitchell Publishing, Inc.

Last, R. W. (1989), Artificial Intelligence Techniques in Language Learning, Ellis Horwood, Chichester.

Lippman, Stanley B. (1991), *C++ Primer 2nd Edition*, Addison-Wesley.

Meyer, Bertrand (1988), *Object-Oriented Software Construction*, Prentice Hall.

Nichols, Tim (1991a), "Job Oriented Object Analysis - A Method of Finding and Validating Objects", Eastman Kodak Company.

Nichols, Tim (1991b), course notes: "Object Oriented Concepts", "Object Oriented Analysis", and "Object Oriented Design", Eastman Kodak Company.

Phillips, Robert (1983), "Masking Techniques to Identify and Diagnose Errors in Foreign Language CAI", proceedings of the Sixth International Conference on Computers and the Humanities, Burton, S. K. and D. D. Short, eds., Computer Science Press.

Randolph, Gary L. (1991), "Photo CD C++ Style Guide", Eastman Kodak Company Software Warehouse, WIN # 379.

Randolph, Gary L. (1992), "C++ for C Programmers", course notes, Eastman Kodak Company.

Rochester Institute of Technology, Graduate Computer Science Department (1988), "Guide to the Master's Thesis", RIT School of Computer Science, August 24, 1988.

Sleeman, D. H. (1981a), "Assessing Aspects of Competence in Basic Algebra", in Intelligent Tutoring Systems, Academic Press, pp. 185-197.

Sleeman, D. H. and R. J. Hendley (1981b), "ACE: A System Which Analyzes Complex Explanations", in Intelligent Tutoring Systems, Academic Press, pp. 99-116.

Sowa, John F. (1984), "Interactive Language Implementation System", IBM Journal of Research and Development, vol. 28, no. 1, January, pp. 28-38.

Stroustrup, Bjarne (1987), *The C++ Programming Language*, Addison-Wesley.

Uhr, L. (1969), "Teaching machine programs that generate problems as a function of interaction with students", Proceedings of the 24th National Conference, pp. 125-134.

Wirfs-Brock, Rebecca, Brian Wilkerson, and Lauren Wiener (1990), *Designing Object-Oriented Software*, Prentice Hall.

Woods, P. and J. R. Hartley (1971), "Some learning models for arithmetic tasks and their use in computer-based learning", British Journal of Educational Psychology, Vol. 41, No. 1, pp. 35-48.

Wybolt, Nicholas (1990), "Experiences with Object-Oriented Software Development", proceedings of the USENIX C++ Conference, April 9-11.

A. APPENDIX - USER MANUAL

A.1. Introduction

This manual describes the use of the German Language Tutoring system as a learning and practice aid. The system allows a student to practice phrase and sentence translation, and to receive some degree of assistance if needed. There are two categories of users, the "student" and the "system administrator". This manual describes the features available to the "student" user. "system administrator" features are described in a separate System Administration Manual.

The tutoring system currently supports a single type of exercise - phrase translation. After logging in (see section A.3.), the system will automatically begin presenting phrases and sentences to be translated.

A.2. Input/Output and Documentation Conventions

A.2.1. Help

The most important command to remember, when working with the system, is "?". Typing a question mark (followed by the "Enter" or "Return" key), at anytime that the system is waiting for input, will cause it to print instructions about what is expected.

A.2.2. Examples in this Document

In examples here, anything typed by the computer system, will be shown in **bold**, and anything that would have been typed by the user will be shown in *italics*. For example:

System typed this> *user responded with this*

A.2.3. Enter/Carriage-Return

As with many software packages, the German Language Tutor requires that the user terminate any line (username, phrase, or menu command number) with the "Enter" or "Return" key. This signals to the computer that it should read what was typed, and interpret it. Before hitting "Enter" or "Return", the line may be changed by deleting ("Delete" or "BackSpace" key) and re-typing characters. The "Enter" or "Return" action is often represented in documents as <cr> (short for carriage-return), so that the above example might have been shown as:

System typed this> *user responded with this<cr>*

However, this looks cluttered after a while, so it should just be assumed that it is required at the end of everything typed.

A.3. Getting Started

A.3.1. The Username

The system only allows validated users to execute it. To become validated, you must obtain a username from the designated system administrator. The "account" that you

get will be designated as "student" or "system administrator". If a user wishes to have both capabilities, two separate usernames must be obtained. The rest of this document assumes a "student" account.

A.3.2. Logging In

To start the tutoring system, type the command 'glt' (short for German Language Tutor). The system should display its start-up message and version number. It will then ask for a chapter number. The user should enter the highest chapter number completed in the text. This will prevent the system from using vocabulary or concepts which have not yet been taught/learned. If the "current level" (saved from a previous login) is correct, the "Enter" or "Return key alone may be hit to remain at the same level. The following example illustrates a valid login by the user with the username "ken":

```
sys27> glt

                    German Language Tutor, version 1.0


If you are not sure how to begin, type "?".

Please login: ken

Enter highest chapter completed (currently 2)? 3

        Exercises will be limited to chapter 3 and earlier.

        You will now be asked to translate various phrases and sentences.
                .
                .
```

Note that a "?" may be entered if this manual is not nearby, and the user needs refresher instructions. If the system says that the username is incorrect, it should be checked to see that it was typed correctly. It should be noted that the system is case-sensitive. That means that "a" is not the same as "A". Usernames will typically be all lower-case. If an incorrect username is entered three times, the system will give up and exit. The user should check with the system administrator to determine what his or her correct username is.

A.4. Doing Exercises

Once successfully logged in to the system, the translation exercises are automatically entered. A phrase or sentence will be presented, and the system will wait for a translation. The following example shows a correct response:

```
Translate:
                the water is fresh
```

```
to German:
                das Wasser ist frisch

        Yes!  That is the expected response.
```

If the response does not match what the system was expecting (and note, that this does not necessarily mean that the translation is incorrect...), a list of options will be displayed:

```
        That is not the expected translation.  Enter the number
        indicating the action you would like to take:

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice?
```

The system will wait for the user to enter the number of the desired choice. Different lists of options may appear depending on the current scenario (for instance, if the system runs out of hints, selection 2 will not appear), but the above example shows most of the options, and they are further described below.

A.4.1. Trying Again

If the option of trying again is chosen, the system will offer the same phrase, and wait for another translation attempt. A user may choose to do this if he or she realizes what their initial mistake was, or to see if the correct answer can be reached by a little trial–and–error.

A.4.2. Receiving a Hint

Choosing to have the system give a hint will cause the system to attempt further diagnosis of the response. It will attempt to give a clue about what was wrong, without revealing the expected answer. Hints can be asked for repeatedly, and the system will attempt to supply more and more specific clues. When the system runs out of hints for a given response, receiving further hints will cease to be an option. The following example illustrates this process:

```
Translate:
                the table is round
to German:
                die Tisch ist rund

        That is not the expected translation.  Enter the number
        indicating the action you would like to take:
```

```
        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        There is something wrong with the first word in your response.

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        The gender of the first word is wrong.

        1 - Try again.
        2 - Have the system give you a hint.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice? 2

        The system cannot diagnose this any further.

        1 - Try again.
        3 - Have the system give you its answer.
        4 - Quit this exercise and go on to the next.
        5 - Exit the tutoring system.
        6 - Help.

Choice?
```

A.4.3. Receiving the Answer

If the user selects this option, the system will print the translation that it was expecting, and then present a new phrase for translation.

A.4.4. Quitting an Exercise

Quitting from an exercise amounts to the same thing as receiving the answer – the expected response is printed, and the system moves on to the next exercise.

A.4.5. Exiting the System

Exiting from the system causes the program to print the expected response for the current exercise, print some summary statistics, and terminate, leaving the user back at the computer's command prompt.

A.4.6. Getting Help

Asking for help prints a brief summary that should clarify what the system is currently waiting for. Recall, also, that a "?" at anytime brings the same thing.

A.5. Statistics

When running a question–and–answer system like this, it can be easy to lose track of time. As a reminder, and to break the monotony periodically, the system will interrupt the exercise sequence to present some statistics. The statistics will reflect the current session. The following example illustrates the type of information which is presented.

```
Duration of session, so far - 20 minutes
Number of different phrases asked to translate - 50
Number of correct responses - 44
Number of total responses - 57

        Enter the number indicating the action you would like to take:

        1 - Continue (return to exercises)
        2 - Exit the tutoring system
        6 - Help

Choice? 1

Translate:
      .
      .
      .
```

B. APPENDIX - SYSTEM ADMINISTRATION MANUAL

B.1. Introduction

This manual describes the German Language Tutoring system's administration. A login "username" must be obtained in order to use the system. It is the system administrator's responsibility to add and delete these usernames as required. Currently, a username must be identified as one of two types - a "student" account or an "administrator" account. A "student" account allows the system to be run for its intended purpose - to practice German. The features available to a "student" user are described in a separate User Manual.

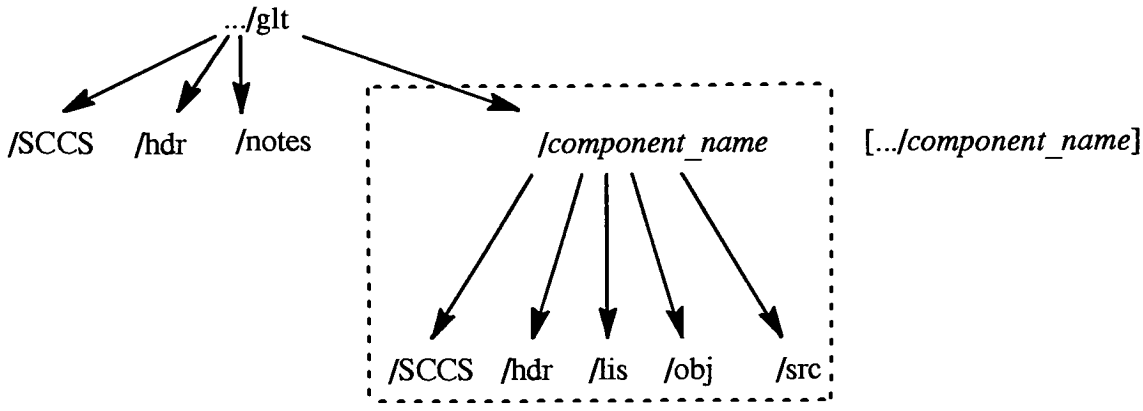
An "administrator" account is not actually required for day-to-day administration of the system. With it, the user gains the ability to look at and manipulate the data gathered by the system at run-time. This data includes logs and statistics on student usage. This information is of particular interest to the designer who wishes to enhance the system, or fix specific shortcomings in diagnosis. This manual is presented in two main sections. Section B.2. discusses the day-to-day system administration - building from source code, adding users, etc. Section B.3. describes the features available to a user with an "administrator" account.

B.2. Off-Line System Administration

This section describes the source code directory structure, the build procedures, the resulting executables and support scripts, and the maintenance of usernames.

B.2.1. Source Code Directories.

The tutoring system source code resides in a directory tree, beginning with the root directory 'glt' (short for German Language Tutor). The location of this tree may vary across systems. UNIX pages for 'CC', 'ld', 'make' and 'sccs', plus knowledge of csh scripts and the C and C++ languages may be of additional help in building and maintaining the system, depending on the level of understanding needed. The following picture illustrates the general directory tree structure.



There is no directory named *"component_name"*. The dotted box above indicates the sub-tree structure which is repeated for multiple "components". A component is a logical grouping of source code and include files, and the directory name will reflect the logical function of the subdirectory (e.g. /models is the component directory which contains the system's PhraseModels). The /glt level can be referred to as the top level, and the /component level can be referred to as the component level (e.g. there is one top-level makefile, and multiple component-level makefiles).

'sccs' (UNIX Source Code Control System) is used for configuration management. All source code and build procedures are checked in to sccs, and should be checked out in order to make any modifications.

'make' (UNIX date-dependency, conditional build control program) is used to build the system. 'make' automatically looks for a file called 'makefile' to operate from. Simply giving the command 'make' in the top-level directory will automatically rebuild whatever portion of the system has been modified since the last time it was built. 'make' can also be invoked at any component level, to build that component (compilation only) without linking it into the system.

The following sections describe the individual directories.

B.2.1.1. Component-Level Directories - */component_name*

As mentioned above, a component is a logical grouping of source code. The current components of the tutoring system are listed below. A component is made up of some number of C++ source code files (.C), some number of include files (.h), and a makefile. All of these are checked into sccs (which maintains the /SCCS sub-directory). Everything else found under the component directory is dynamically built.

After a component is built, reference (read-only) copies of the .C files are left in the /src directory, and reference copies of the .h files are left in the /hdr directory. A reference copy of the makefile is left in the component-level directory. The out-

put of a component build is a single object module (named *component_name*.o). It is left in the component-level directory, and will be used by the top-level makefile to construct a complete system (see below). The /obj directory is filled with the intermediate object modules (one per source file), and the /lis directory contains the intermediate C code which was generated from the C++ code (also one per source file. These are for reference only, and may be deleted to conserve disk space).

B.2.1.1.1. /construction Component

This component contains the source code for objects which are primarily responsible for dynamic phrase construction - word and phrase descriptors, pattern constructors, phrases and words.

B.2.1.1.2. /dictionary Component

This component contains the dictionary and dictionary entry objects, as well as the dictionary data.

B.2.1.1.3. /log Component

This component contains the run-time data gathering log and log entry objects.

B.2.1.1.4. /models Component

This component contains the objects and data which support the phrase models and base phrases.

B.2.1.1.5. /session Component

This component contains the controlling objects and agents for the session, statistics gathering, and the user interface.

B.2.1.1.6. /users Component

This component contains the objects which support user information.

B.2.1.2. Top Level Directory - .../glt

This directory represents the whole system. Here will be a makefile which will build anything that is out-of-date (including down in component directories) resulting in a complete and up-to-date system executable. The makefile will automatically invoke each component-level makefile. No .C files are maintained at the top level, but it does support top-level (global) include files. Include files are maintained at the top level if they are used by more than one component. A change to a global include file causes all components to re-build, so it is advantageous to keep include files at the component level if possible.

As at the component level, the makefile, and global include files are checked into sccs. In addition, any documentation, support, or utility scripts will be checked into sccs. Reference copies of this documentation will be put in the /notes sub-directory, and executable copies of the scripts will be left in the top-level directory. After a top-level build, the system executables will also be left in the top-level directory (see section B.2.2. for additional detail about these).

B.2.2. Executables

Because this system is a prototype, and because most of its future potential involves further modification, all test and debug capabilities have been left available. Simply invoking 'make' (or 'make all') at the top-level will build both debug and non-debug executables.

B.2.2.1. Non-debug executable - 'glt'

The normal executable, 'glt' will be built when 'make' is specified at the top level. To build 'glt' exclusively, specify 'make nodebug'. 'glt' is built with all debug code and feedback removed, so it is the smallest and fastest executable. This should be the one made generally available to students. SPECIAL NOTE: currently the separate 'glt' and 'dglt' executables do not exist. A single executable, 'glt' is built, and it is actually the version described as 'dglt' in this manual.

B.2.2.2. Debug executable - 'dglt'

'dglt' contains all of the code and functionality of 'glt', plus a large amount of debug and informational code. It is essentially a verbose version of glt. When run directly, dglt will display this information to the screen as it runs. The volume of information makes this rather unmanageable, though, so the 'split' script (section B.2.2.2.2. below) may be used to make dglt more practical to use.

B.2.2.2.1. Debug output conventions

As mentioned, a large volume of information is dumped when running the debug executable. To make this somewhat easier to use, a few basic conventions have been adopted. Refer to the following excerpt of debug information which illustrates these. The conventions are summarized following the excerpt.

```
                                    ENTRY BasePhrase::get_index
                        DEBUG returning the index at array location [1]
                        DEBUG       which is 6
"Normal system output"
                                    ENTRY      Word Constructor 1
                                    ENTRY      (for automatics)
```

- Debug information is broken into two categories. Lines beginning with ENTRY indicate a function or member function entry point. Reading down the line of ENTRYs yields a call-trace of the execution. All other information is tagged with DEBUG, to distinguish it from both the ENTRY points and the normal system output.
- DEBUG output lines are preceded by three tabs (24 spaces) and ENTRY output lines are preceded by five tabs (40 spaces). This is to visually separate them from each other, and to move them away from normal, left-justified, system output.
- Constructor and Destructor entry points have four spaces between the ENTRY keyword and the object name. Other functions have a single space. This is another visual aid, helpful when it is desired to ignore constructors and destructors (as peripheral to the execution trace).
- For objects that have multiple constructors, such as the Word object in the above example, the constructors are numbered, and followed by a brief comment, so that it is obvious when reading the log, which constructor was used.
- Subsequent DEBUG statements which have increasing number of spaces after the DEBUG keyword, in multiples of four spaces, reflect continuation or nested comment information.
- In the source code, all DEBUG and ENTRY output is sent to the standard error output stream (cerr) as opposed to the standard output stream (cout).
- Also in the source code, all output (debug or not) is flushed at the end of each line (using endl). This is to keep the output log accurate, should the system hang up somewhere. (If output was not flushed, some output could be buffered when the system hangs, thereby not accurately reflecting the execution prior to the problem.)

B.2.2.2.2. Supporting scripts

The 'split' script is designed to make using the 'dglt' executable easier. 'dglt' outputs a large amount of debug information, but must also output and input normal user interaction information. Running 'dglt' directly results in a large volume of output dumped to the screen. If the 'split' script is invoked, it will run 'dglt', redirecting the standard error output stream to a file named debug.log. Output to the screen will appear normal (the same as 'glt') and the debug output can be examined after the execution is complete. SPECIAL NOTE: As mentioned earlier, only a single executable is currently supported. It is named 'glt', but it contains the debug information described here as 'dglt'. Because of this, the 'split' script currently references 'glt', not 'dglt'.

B.2.3. Sample Build Output

The following sample output is included to illustrate the build procedure. It represents the build which results from changing a single .C file (phrasemodel.C in the 'models'

component. If multiple components had been changed, or if a global include file had been changed, the build output would have been much larger.

```
SYS27> make debug
#************************ Update global header files.
#************************ Check each sub-system of tutor.
for i in construction   dictionary   log   models   session   users ; do \
        cd  $i  ;  make    GBL_HDRS="../hdr/basephrase.h       ../hdr/dictionary.h
../hdr/log_defs.h    ../hdr/log.h    ../hdr/phrase.h    ../hdr/phrasedescriptor.h
../hdr/phrasemodellist.h      ../hdr/statistics.h      ../hdr/stdtype.h      ../hdr/
user_defs.h       ../hdr/userinterface.h       ../hdr/userlist.h       ../hdr/word.h
../hdr/worddescriptor.h" ; cd .. ; \
done
'construction.o' is up to date.
'dictionary.o' is up to date.
'log.o' is up to date.
#***************************** Get 'models' source code file
cd src; sccs -d../ get phrasemodel.C; cd ..;
1.54
177 lines
No id keywords (cm7)
#***************************** Compile 'models' source file
cd src; CC -DDEBUG +i -c -I../hdr -I../../hdr phrasemodel.C ;  \
rm *..c;  \
mv phrasemodel.o ../obj/phrasemodel.o; cd ..
#***************************** Build a single 'models' object module
ld -r obj/BPT_no_translation.o   obj/basephrase.o   obj/basephraselist.o   obj/
basephrasepool.o   obj/basephrasetrans.o   obj/data_phrasemodellist.o   obj/pat-
tern.o   obj/phrasemodel.o   obj/phrasemodellist.o   obj/translationtree.o -o
models.o
'session.o' is up to date.
'user.o' is up to date.
#************************ Build the executable - 'dglt'.
CC construction/construction.o   dictionary/dictionary.o   log/log.o   models/mo-
dels.o   session/session.o   users/users.o -o dglt
SYS27>
```

Specifying 'make debug' requests that the debug executable 'dglt' be built. The build procedure first attempts to update any global include files which have changed (none have in this case). It then loops through the various components (passing down the list of global include files so that their modification dates can be considered by the components) invoking the build procedure at that level. The 'construction', 'dictionary' and 'log' components indicate that they are up to date. When the build procedure reaches

the 'models' component, it finds the file phrasemodel.C to be out of date. The source file is extracted from source control, and compiled with the DEBUG switch. The component then builds its object module (models.o). Back at the top level, the 'session' and 'user' components indicate that they are up to date. Finally, the top level build procedure builds the new debug executable, 'dglt'.

B.2.4. Adding Users

The tutoring system determines whether or not a user is authorized to use the system, by checking a file called users.data in the top level directory. This file is under SCCS control. There is a problem with this, though, in that the system may update this file (for a student's current knowledge level), without going through SCCS. An appropriate procedure for updating this file manually, would be:

- rename the current users.data file to a temporary name.
- sccs edit users.data (to check the file out for change).
- rename the temporary file back to users.data (overwriting the old version).
- make the necessary manual changes (i.e. adding or deleting users).
- sccs delta users.data (to check this version into source control).
- sccs get users.data (for the tutoring system to use).

A username entry consists of four lines in the users.data file. The first line is a single number, indicating whether this user is a student (1) or system administrator (2). The second line is also a single number, indicating the user's current knowledge level. This should be zero originally. The tutoring system will update the student's current level as needed. The third line is a single word which will be the user's login name. It should contain no spaces, and should be all lower case. The fourth line is currently used to describe the user in more detail, such as his or her first name. Currently, this line should also have no spaces in it. A new student entry might look like:

```
1
0
tjones
Thomas_Jones
```

B.3. System Administrator User Features

A separate manual exists describing the student's use of the system, and that manual should be considered a prerequisite to this one (both from a user standpoint, and because the conventions apply to this account as well). This document describes specifically the extra features available with a system administrator username.

Whenever student features are run on the system, data is gathered and appended to a log file called 'glt_admin.log'. This file is in an ASCII, though not too readable, for-

mat. The system administrator login provides some basic log manipulation, viewing, and report generating features. Access to these features is via simple menus, like those used during normal student execution. The available features are described below.

B.3.1. Delete Log

This is just what the name implies – the 'glt_admin.log' file is deleted. This might be done to start with a clean baseline (e.g. because the system has been modified), or to discard data which has outlived its usefulness. An alternative to this, though, is the Save Log feature described below.

B.3.2. Save Log

This effectively clears the data log, by copying its contents to a new file (the system will ask for a new file name). This can be used when it is desirable to start a new log, without losing data that has been previously gathered.

B.3.3. View Diagnosis Failures

One point where the system logs data, is when it must give up on a diagnosis (i.e. attempting to interpret a student's translation response.) It is desirable to examine these cases to identify user errors which should have been handled by the system. This will be one of the key ways to improve the system's performance against its current knowledge base. This feature allows the user to step through each incident report. The following brief interchange with the system after selecting this option illustrates the information available.

```
        Enter the number indicating the action you would like to take:

        [...]
        7 - View Diagnosis Failures
        [...]

Choice? 7

Enter Earliest Date Desired (MM-DD-YY)? 1-1-93

Date - 03-Jan-93        Username - ken
Index of Phrase Model which was used - 5
Phrase which was to be translated - The table is round.
Expected response - Der Tisch ist rund.
Actual response - Bet the system can't tell me what's wrong with this!

        Enter the number indicating the action you would like to take:
```

```
        1 - Continue
        2 - Quit

Choice? 1

Date - ...
```

B.3.4. Summarize Diagnosis Failures

This feature is very similar to on-line viewing of diagnosis failures, except that the system requests a file name, and generates a report summarizing all of the failures since the specified start date.

B.3.5. View Statistics

Whenever a session with the system is completed, an entry containing statistics from this session is entered in the log. This information might be of interest in characterizing users, qualifying other data, or monitoring system usage. Like View Diagnosis Failures, this feature asks for a data, and begins displaying statistics entries made since that date. The following is an example of a statistics entry.

```
Enter Earliest Date Desired (MM-DD-YY)? 1-1-93

Date - 03-Jan-93        Username - ken
User knowledge level at time of session - 2
Duration of session - 2 minutes
Number of different phrases asked to translate - 5
Number of correct responses - 4
Number of total responses - 7
Phrase Model indices answered correctly first time - 7, 3, 1, 9
Phrase Model indices where hints were given - 6
Phrase Model indices where answer had to be given - 6

        Enter the number indicating the action you would like to take:

        1 - Continue
        2 - Quit

Choice? 1

Date - ...
```

B.3.6. View Cumulative Statistics

This feature is very similar to View Statistics. The system requests a date, and than compiles all of the individual statistics entries which have occurred since that date. The summary looks like the following:

```
Summary of statistics 03-Jan-93 through [current date]
Number of sessions - 8
Average user knowledge level - 3
Spread of user knowledge levels - 1-4
Average duration of session - 15 minutes
Total system usage time - 2 hours 0 minutes
Total number of different phrases asked to translate - 100
Total number of correct responses - 82
Total Number of responses - 115
```

B.3.7. Summarize Statistics

This is another report-generation feature. The system requests a file name, and generates a report summarizing all of the statistics entries made since the specified start date. It also includes the cumulative statistics based on all entries included in the report.

B.3.8. Full Report

This generates a report containing everything that is contained in the current log. (Basically a compilation of all of the previously described report generation features.

B.3.9. Help

As in the student user environment, "help" can be selected at any time to obtain more detailed instructions from the system.

C. APPENDIX - COMMERCIAL SOFTWARE FOR GERMAN

Many of the products listed here have "sister" products which support other languages. For instance, in addition to "German Achievement I", Microcomputer Workshops Courseware also markets "French Achievement I" and "Spanish Achievement I".

Dasher - for Apple II computers by Conduit. This program is a drill-and-practice program which allows the teacher to enter the material to be drilled. Pre- designed data sets are available as companions to specific textbooks. Sentences are presented for translation. The student response is compared letter-to-letter with the expected response, and echoed back to the student with dashes (hence "dasher") in the place of discrepancies. This occurs a specified number of times before the correct answer is given. The teacher sets the re-try level.

Deutsch Aktuell - for Apple II computers by EMC Publishing. This is a collection of exercises and lessons covering vocabulary, grammar, reading comprehension, communication skills and culture. The exercises are classic drill-and- practice. If a student answers incorrectly, the correct answer is displayed, and the program moves on. There is no intelligent diagnosis. This program has also been criticized for its distracting shifts between graphics and text on the screen [LaReau-1989].

Foreign Language Courseware - German Level I Vocabulary/German Level II Vocabulary - for Apple II computers by Learning Technology. This is another classic drill-and-practice program for vocabulary. There is no diagnostic feedback, but the program does keep statistics on each exercise, including wrong answers, for later review.

The Game Show Subject Diskette - Foreign Language Words - for IBM PCs, Apple II and Commodore computers by Advanced Ideas. This program is a twist on vocabulary drill-and-practice. It is run as a game, and the student competes for a score. A clue to a word is presented in German, and the student must guess the English word. The more clues needed for a specific word, the fewer points earned. Although the student is only responding with single vocabulary words, he/she must read and comprehend the phrase and sentence clues.

German I Vocabulary/German II Vocabulary - for Apple II computers by Learning Technology, Inc. These are classic drill-and-practice vocabulary programs. The student translates individual words. Incorrect answers get one retry, then the program moves on.

German Achievement I/German Achievement II - for IBM PCs and Apple II computers by Microcomputer Workshops Courseware. German Achievement I is a drill-and-practice program for vocabulary. An incomplete sentence is presented, along with multiple choices for possible completions. If the student answers correctly, the program moves on. If the student answers incorrectly, the correct answer is given, along with a canned explanation associated with the incorrect choice. German Achievement II uses the same drill-and-practice scheme, but grammatical concepts are reviewed as opposed to vocabulary.

German Computer Grammar I and II – for Apple II computers by Lingo Fun. These programs are highly rated drill–and–practice programs for grammatical concepts. Brief canned lessons on each concept are available.

German Hangperson – for Apple II computers by Learning Technology, Inc. This program plays the game "hangman".

German Practice – for Apple II computers by George Earl. This program plays the game "hangman", using sentences, instead of single words. The student still guesses letters for hints.

German Review Packet – for IBM PCs and Apple II computers by COMPress. This is a drill–and– practice program for German Grammar. Each set of exercises begins with a brief lesson on the specific concept to be tested. The student is then asked questions requiring short (one or two word) answers. If the student answers incorrectly, one retry is given before the program gives the correct answer.

German Word Order – for Apple II computers by Gessler Educational Software. This is a sentence construction exercise program. All sentences are canned. The program chooses a sentence, then prints the words in random order. The student must attempt to reconstruct the correct sentence.

Mutlilingual Story Teller – for Apple II computers by Lingo Fun. This program allows the student to practice writing skills. There are no intelligent diagnostics, so teacher feedback is required to determine correctness. The student enters statements in a binary tree, so that the story will be generated in different ways depending on which branch the program takes during play–back.

Snooper Troops - German – for Apple II computers by Gessler Educational Software. This is a role–playing game. The student is the detective, and the program supplies clues to a crime. The student must comprehend the clues, eliminate suspects, and identify the guilty person.

Vocabulary on the Attack – for Apple II computers by Langenscheidt. This is a multiple–choice drill–and–practice program for vocabulary. The student is presented with a word, and must select the correct translation. The correct answer must be given before the program will move on.

Vocabulary Workshop – for Apple IIe computers by Sterling Swift Publishing Company. This is a game variation of vocabulary drill–and–practice. The student begins with some play money, and then places wagers on his/her confidence in the chosen response. The translation words are presented as multiple choices. If the student is very sure of the answer, more money can be won (or lost) than if the student indicates less confidence. The program gives the correct answer and moves on after one attempt.

Wie Sagt Man? – for Apple II computers by Lingo Fun. This is very similar to the German Practice program described above, except that the hangman is missing (I guess that makes it more like "Wheel of Fortune"). The student guesses letters which are filled into the sentence to be guessed. The student has wide control over the level of difficulty. Higher difficulty levels will not show blanks for individual letters, and may not insert every occurrence of a successfully guessed letter.

Wortgegecht - for Apple II computers by Gessler Educational Software. This is a vocabulary drill program which plays the game "word attack". The student selects the category of vocabulary to be used (e.g. "sports and recreation"). Besides "word attack" the student may also browse the vocabulary list, take a multiple choice quiz, or do sentence completion.

D. APPENDIX - SOURCE CODE LISTINGS

All files included in this appendix are listed below. Header files appear first, listed alphabetically, followed by .C files, also listed alphabetically, followed by build procedures and execution scripts. For information on which header files are top-level, and which components the remaining files belong to, refer to the makefiles near the end of the listings (or the banner line of any individual listing).

The Session agent (session.C) is the main() entry point of the program.

Objects	Other	Build Procedures/Scripts
	PC_defs.h	
basephrase.h		
basephraselist.h		
basephrasepool.h		
compoundstring.h		
dictionary.h		
	dictionary_defs.h	
dictionaryentry.h		
dynamicworddescriptor.h		
hint.h		
log.h		
	log_defs.h	
logentry.h		
patternnode.h		
phrasedescriptor.h		
phrasemodel.h		
phrasemodellist.h		
	session.h	
statistics.h		
	statistics_defs.h	
	stdtype.h	
translationtree.h		
user.h		
	user_defs.h	
userinterface.h		
	userinterface_defs.h	
userlist.h		
word.h		
	word_and_phrase_defs.h	
worddescriptor.h		
	worddescriptor_defs.h	
	BPT_drop_fourth_word.C	
	BPT_no_translation.C	
	PC_extra_article_fourth.C	
	PC_gender1_first_word.C	

Objects	Other	Build Procedures/Scripts
	PC_gender2_first_word.C	
	PC_nouncap_second_word.C	
	PC_wrong_first_and_second_word.C	
	PC_wrong_first_word.C	
	PC_wrong_fourth_word.C	
	PC_wrong_second_word.C	
	PC_wrong_third_word.C	
basephrase.C		
basephraselist.C		
basephrasepool.C		
compoundstring.C		
	data_PC_defs.C	
	data_dictionary.C	
	data_phrasemodellist.C	
dictionary.C		
dictionaryentry.C		
dynamicworddescriptor.C		
	exit_tutor.C	
hint.C		
log.C		
logentry.C		
patternnode.C		
phrasedescriptor.C		
phrasemodel.C		
phrasemodellist.C		
	session.C	
statistics.C		
	sys_exit_tutor.C ,	
translationtree.C		
user.C		
userinterface.C		
userlist.C		
	utility_gender1_var.C	
	utility_gender2_var.C	
	utility_not_capitalized.C	
word.C		
worddescriptor.C		
		READ_ME
		top-level makefile
		construction makefile
		dictionary makefile
		log makefile
		models makefile
		session makefile
		users makefile

Objects	Other	Build Procedures/Scripts
		baseline_all
		checkin_all
		checkout_all
		cleanup_all
		print_all
		split
		users.data

```
1     //-----------------------------------------------------------------------------
2     //
3     //  Filename:   PC_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines symbols used by the utility
8     //              function pattern constructors PC_*().
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-----------------------------------------------------------------------------
18
19    #ifndef PC_DEFS_H
20    #define PC_DEFS_H
21
22            //-----------------------------------------------//
23            // This offset can be used for case conversion  //
24            // of ASCII characters.  Adding it to an upper  //
25            // case character yields lower case, and        //
26            // subtracting it from a lower case character   //
27            // yields upper case.                           //
28            //-----------------------------------------------//
29
30    #define CASE_CONVERSION_OFFSET  32
31
32    #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   basephrase.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the BasePhrase class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef BASEPHRASE_H
19    #define BASEPHRASE_H
20
21
22            //-----------------------------------------------//
23            // A BasePhrase object represents a single      //
24            // semantically correct phrase.  (Note that the //
25            // syntactic information in a PhraseDescriptor  //
26            // will complete the information necessary to   //
27            // generate a valid phrase.                     //
28            //-----------------------------------------------//
29
30    class BasePhrase
31    {
32        public:
33            BasePhrase(int in_length, int index1=0, int index2=0, int index3=0,
34                    int index4=0, int index5=0, int index6=0, int index7=0,
35                    int index8=0, int index9=0, int index10=0);
36            BasePhrase(const BasePhrase &);
37            ~BasePhrase();
38            BasePhrase& operator=(BasePhrase & in_BP);
39            int get_index(int position);
40            int get_size();
41            int get_level();
42        private:
43            int length;
44            int knowledge_level;
45            int * indices;
46    };
47
48    #endif
```

```
1     //-------------------------------------------------------------------------
2     //
3     //  Filename:   basephraselist.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the BasePhraseList class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------
17
18    #ifndef BASEPHRASELIST_H
19    #define BASEPHRASELIST_H
20
21    #include "basephrase.h"
22    #include "stdtype.h"
23
24            //-----------------------------------------------//
25            // A BasePhraseList object will represent a      //
26            // collection of BasePhrases (semantically       //
27            // correct phrases).                             //
28            //-----------------------------------------------//
29
30    class BasePhraseList
31    {
32        public:
33            BasePhraseList(int list_size, BasePhrase * first_phrase);
34            ~BasePhraseList();
35            BasePhrase * get_BasePhrase();
36            boolean reset_selections(int in_know_level);
37            boolean set_level(int in_know_level);
38        private:
39            int size;
40            boolean * selected;
41            BasePhrase * * phrases;
42    };
43
44    #endif
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   basephrasepool.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the BasePhrasePool class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------

17
18  #ifndef BASEPHRASEPOOL_H
19  #define BASEPHRASEPOOL_H
20
21  #include "basephraselist.h"
22
23          //-----------------------------------------------//
24          // A BasePhrasePool object will represent a      //
25          // collection of BasePhraseLists (which are      //
26          // themselves, collections of BasePhrases).      //
27          //-----------------------------------------------//
28
29  class BasePhrasePool
30  {
31      public:
32          BasePhrasePool(int pool_size, BasePhraseList * list1=0,
33                         BasePhraseList * list2=0, BasePhraseList * list3=0,
34                         BasePhraseList * list4=0, BasePhraseList * list5=0,
35                         BasePhraseList * list6=0, BasePhraseList * list7=0,
36                         BasePhraseList * list8=0, BasePhraseList * list9=0,
37                         BasePhraseList * list10=0);
38          ~BasePhrasePool();
39          BasePhrase * get_BasePhrase();
40          boolean reset_selections(int in_know_level);
41          boolean set_level(int in_know_level);
42      private:
43          int size;
44          boolean * selected;
45          BasePhraseList * * lists;
46  };
47
48  #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   compoundstring.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the CompoundString class, and
8     //              its derived classes Phrase and Pattern.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-------------------------------------------------------------------------------
18
19    #ifndef COMPOUNDSTRING_H
20    #define COMPOUNDSTRING_H
21
22    #include "word.h"
23    #include "stdtype.h"
24
25            //-----------------------------------------------//
26            // The CompoundString class is a building block //
27            // for the Phrase and Pattern classes.          //
28            //-----------------------------------------------//
29
30    class CompoundString
31    {
32        public:
33            CompoundString();
34            ~CompoundString();
35            CompoundString& operator=(CompoundString & in_CS);
36            void set_word(int position, Word in_word);
37            Word get_word(int position);
38            void get_string(char * in_ptr);
39            int size;
40            Word * * words;
41        protected:
42    };
43
44            //-----------------------------------------------//
45            // A Phrase is a CompoundString representing a  //
46            // syntactically and semantically correct       //
47            // phrase or sentence.  It can be printed       //
48            // to the screen, or input from the user.       //
49            //-----------------------------------------------//
50
51    class Phrase   public CompoundString
52    {
53        public:
54            Phrase(int in_size);
55            Phrase(const Phrase &);
56            ~Phrase();
57            Phrase& operator=(Phrase& in_phrase);
58    #ifdef DEBUG
59            void debug_print();
60    #endif
61            void print();
62            void read();
63            int get_size();
64        private:
65    };
66
```



```
67            //-----------------------------------------------//
68            // The Pattern is a CompoundString which can be //
69            // used for diagnosing input Phrases.  It has   //
70            // the ability to match itself against a        //
71            // Phrase.                                      //
72            //-----------------------------------------------//
73
74    class Pattern · public CompoundString
75    {
76        public:
77            Pattern(const Pattern &);
78            Pattern(const Phrase &);
79            ~Pattern();
80            Pattern& operator=(Pattern& in_pattern);
81            boolean match(Phrase in_phrase);
82        private:
83    };
84
85    #endif
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   dictionary.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the Dictionary class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------
17
18  #ifndef DICTIONARY_H
19  #define DICTIONARY_H
20
21  #include "word.h"
22  #include "worddescriptor.h"
23
24          //-----------------------------------------------//
25          // These forward references are used so that     //
26          // dictionary.h can be a top-level include       //
27          // file, while dictionaryentry.h resides at      //
28          // the component level.                          //
29          //-----------------------------------------------//
30
31  class DictionaryEntry_noun;
32  class DictionaryEntry_article;
33  class DictionaryEntry_adjective;
34  class DictionaryEntry_verb;
35
36          //-----------------------------------------------//
37          // The Dictionary class represents all of the    //
38          // words and morphological derivatives           //
39          // supported by the system.                      //
40          //-----------------------------------------------//
41
42  class Dictionary
43  {
44      public:
45          Dictionary(DictionaryEntry_noun * noun_list,
46                     DictionaryEntry_article * article_list,
47                     DictionaryEntry_adjective * adj_list,
48                     DictionaryEntry_verb * verb_list);
49          ~Dictionary();
50          int get_level(int word_index);
51          Word get_word(int word_index, WordDescriptor * word_desc);
52      private:
53          DictionaryEntry_noun * nouns;
54          DictionaryEntry_article * articles;
55          DictionaryEntry_adjective * adjs;
56          DictionaryEntry_verb * verbs;
57  };
58
59  #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   dictionary_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file defines symbols supporting various dictionary objects.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef DICTIONARY_DEFS_H
19    #define DICTIONARY_DEFS_H
20
21            //-----------------------------------------------//
22            // Ranges for various word types.                //
23            //-----------------------------------------------//
24
25    #define DE_MIN_NOUN_INDEX            1
26    #define DE_MAX_NOUN_INDEX        99999
27
28    #define DE_MIN_DIRART_INDEX     100001
29    #define DE_MAX_DIRART_INDEX     199999
30
31    #define DE_MIN_ADJ_INDEX        200001
32    #define DE_MAX_ADJ_INDEX        299999
33
34    #define DE_MIN_VERB_INDEX       300001
35    #define DE_MAX_VERB_INDEX       399999
36
37    #endif
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   dictionaryentry.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the DictionaryEntry class - a single
8   //              entry in the dictionary and its derived classes.
9   //
10  //  Notes:      The theory and design of this system have been documented
11  //              in the accompanying thesis report.
12  //
13  //  Revisions   By              Reason
14  //  ---------   --              ------
15  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16  //
17  //-------------------------------------------------------------------------
18
19  #ifndef DICTIONARYENTRY_H
20  #define DICTIONARYENTRY_H
21
22  #include "word.h"
23  #include "worddescriptor.h"
24
25          //-----------------------------------------------//
26          // The DictionaryEntry class represents a       //
27          // single word's entry in the dictionary.       //
28          //-----------------------------------------------//
29
30  class DictionaryEntry
31  {
32      public:
33          DictionaryEntry(int init_index, int init_level);
34          ~DictionaryEntry();
35          int get_level();
36          int get_index();
37      private:
38          int index;
39          int knowledge_level;
40  };
41
42          //-----------------------------------------------//
43          // Different derived classes represent the      //
44          // various word types managed by the            //
45          // dictionary, and reflect the different        //
46          // attributes, etc., of each.                   //
47          //-----------------------------------------------//
48
49  class DictionaryEntry_article   public DictionaryEntry
50  {
51      public:
52          DictionaryEntry_article(int init_index,
53                                  int init_know_level,
54                                  char * init_masc,
55                                  char * init_fem,
56                                  char * init_neut, char * init_plural,
57                                  char * init_eng);
58          ~DictionaryEntry_article();
59          Word derive_word(WordDescriptor * word_desc);
60      private:
61          char * masc;
62          char * fem;
63          char * neut;
64          char * plural;
65          char * eng;
66  };
```



```
67
68  class DictionaryEntry_noun · public DictionaryEntry
69  {
70      public:
71          DictionaryEntry_noun(int init_index,
72                               int init_know_level,
73                               Gender init_gend,
74                               char * init_g_sing,
75                               char * init_g_plural, char * init_e_sing,
76                               char * init_e_plural);
77          ~DictionaryEntry_noun();
78          Word derive_word(WordDescriptor * word_desc);
79          Gender gend;
80      private:
81          char * g_sing;
82          char * g_plural;
83          char * e_sing;
84          char * e_plural;
85  };
86
87  class DictionaryEntry_adjective   public DictionaryEntry
88  {
89      public:
90          DictionaryEntry_adjective(int init_index,
91                                    int init_know_level,
92                                    char * init_e,
93                                    char * init_g);
94          ~DictionaryEntry_adjective();
95          Word derive_word(WordDescriptor * word_desc);
96      private:
97          char * ger;
98          char * eng;
99  };
100
101 class DictionaryEntry_verb: public DictionaryEntry
102 {
103     public:
104         DictionaryEntry_verb(int init_index,
105                              int init_know_level,
106                              char * init_e,
107                              char * init_g);
108         ~DictionaryEntry_verb();
109         Word derive_word(WordDescriptor * word_desc);
110     private:
111         char * ger;
112         char * eng;
113 };
114
115 #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   dynamicworddescriptor.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the DynamicWordDescriptor class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef DYNAMICWORDDESCRIPTOR_H
19    #define DYNAMICWORDDESCRIPTOR_H
20
21    #include "stdtype.h"
22    #include "word.h"
23    #include "worddescriptor.h"
24
25            //-----------------------------------------------//
26            // The DynamicWordDescriptor class compiles      //
27            // word attributes which subsequent words may    //
28            // have to agree with.                           //
29            //-----------------------------------------------//
30
31    class DynamicWordDescriptor
32    {
33        public:
34            DynamicWordDescriptor();
35            ~DynamicWordDescriptor();
36            Count get_count();
37            Gender get_gender();
38            Language get_language();
39            void resolve_dynamics(Word * in_word);
40        private:
41            Count count;
42            Gender gender;
43            Language language;
44    };
45
46    #endif
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   hint.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the Hint class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18  #ifndef HINT_H
19  #define HINT_H
20
21          //-----------------------------------------//
22          // The Hint class represents an individual clue //
23          // associated with a particular pattern in a    //
24          // diagnosis tree.                              //
25          //-----------------------------------------//
26
27  class Hint
28  {
29      public:
30          Hint(char * text);
31          ~Hint();
32          void print();
33      private:
34          char * hint;
35  };
36
37  #endif
```

```
1     //-------------------------------------------------------------------------
2     //
3     //  Filename:   log.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the run-time Log class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------
17
18    #ifndef LOG_H
19    #define LOG_H
20
21    #include "log_defs.h"
22    #include "compoundstring.h"
23    #include "statistics.h"
24
25            //-----------------------------------------------//
26            // This forward reference is used to allow       //
27            // log.h to exist as a top-level include file,   //
28            // while logentry.h resides at the component     //
29            // level.                                        //
30            //-----------------------------------------------//
31
32    class LogEntry;
33
34            //-----------------------------------------------//
35            // The Log class compiles any run-time data      //
36            // to be made available to the system            //
37            // administrator.                                //
38            //-----------------------------------------------//
39
40    class Log
41    {
42        public:
43            Log();
44            ~Log();
45            void set_current_user(char * in_user);
46            void at_leaf(LE_at_leaf_type in_al_type, Phrase in_exp_response,
47                                        Phrase in_user_response);
48            void final_stats(Statistics in_stats);
49            void set_current_model_index(int in_index);
50            void set_current_presentation_phrase(Phrase in_pres_phrase);
51            void save();
52            void load();
53            void view_at_leafs();
54            void view_ind_stats();
55            void view_cum_stats();
56        private:
57            int current_entry_count;
58            LogEntry * * list;
59            char * current_username;
60            int current_model_index;
61            char * current_presentation_phrase;
62    };
63
64    #endif
```

```
1     //-----------------------------------------------------------------------------
2     //
3     //  Filename:   log_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines some Log-related symbols.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-----------------------------------------------------------------------------
17
18    #ifndef LOG_DEFS_H
19    #define LOG_DEFS_H
20
21            //---------------------------------------------//
22            // The maximum number of active log entries    //
23            // which the system can hold internally at     //
24            // runtime.                                    //
25            //---------------------------------------------//
26
27    #define MAX_CURRENT_LOG_ENTRIES 1000
28
29            //---------------------------------------------//
30            // The maximum string length of a date/time    //
31            // stamp.                                      //
32            //---------------------------------------------//
33
34    #define MAX_DATE_STAMP_SIZE     50
35
36            //---------------------------------------------//
37            // LogEntry types and sub-types.               //
38            //---------------------------------------------//
39
40    enum LE_type
41    {
42            LE_UNKNOWN_TYPE,
43            LE_at_leaf,
44            LE_final_stats
45    };
46
47    enum LE_at_leaf_type
48    {
49            LE_UNKNOWN_AT_LEAF_TYPE,
50            LE_at_leaf_hint,
51            LE_at_leaf_diag
52    };
53
54    #endif
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   logentry.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the LogEntry class - a single
8   //              entry in the run-time log, and its derived classes.
9   //
10  //  Notes:      The theory and design of this system have been documented
11  //              in the accompanying thesis report.
12  //
13  //  Revisions   By              Reason
14  //  ---------   --              ------
15  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16  //
17  //-------------------------------------------------------------------------
18
19  #ifndef LOGENTRY_H
20  #define LOGENTRY_H
21
22  #include <fstream.h>
23  #include <time.h>
24  #include "log_defs.h"
25  #include "compoundstring.h"
26  #include "statistics.h"
27
28          //-----------------------------------------------//
29          // The LogEntry class represents a single       //
30          // informational entry in the run-time log.     //
31          //-----------------------------------------------//
32
33  class LogEntry
34  {
35      public:
36          LogEntry();
37          LogEntry(char * in_username, int in_index, LE_type in_type);
38          ~LogEntry();
39          virtual void print(ofstream * log_stream)=0;
40          LE_type get_type();
41      protected:
42          time_t date_time_stamp;
43          char * username;
44          int model_index;
45          LE_type record_type;
46      private:
47  };
48
49          //-----------------------------------------------//
50          // This derived class represents the log entry  //
51          // made when the diagnostic tree hits a leaf    //
52          // node and has to give up.                     //
53          //-----------------------------------------------//
54
55  class LogEntry_at_leaf . public LogEntry
56  {
57      public:
58          LogEntry_at_leaf(LE_type in_type);
59          LogEntry_at_leaf(char * in_username, int in_index, char *
60                                  in_pres_phrase,
61                                  LE_at_leaf_type in_type, Phrase in_exp_response,
62                                  Phrase in_user_response);
63          ~LogEntry_at_leaf();
64          void print(ofstream * log_stream);
65          void read(ifstream * log_stream);
66          void display();
```



```
67      private:
68          LE_at_leaf_type type;
69          char * presentation_phrase;
70          char * expected_response;
71          char * user_response;
72  };
73
74          //-----------------------------------------------//
75          // This derived class represents the log        //
76          // entry made at the completion of a session.   //
77          //-----------------------------------------------//
78
79  class LogEntry_final_stats: public LogEntry
80  {
81      public:
82          LogEntry_final_stats(LE_type in_type);
83          LogEntry_final_stats(char * in_username, Statistics in_stats);
84          ~LogEntry_final_stats();
85          void print(ofstream * log_stream);
86          void read(ifstream * log_stream);
87          void display();
88          Statistics * get_stats();
89      private:
90          Statistics current_stats;
91  };
92
93  #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   patternnode.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the PatternNode class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef PATTERNNODE_H
19    #define PATTERNNODE_H
20
21    #include "compoundstring.h"
22    #include "hint.h"
23
24            //-----------------------------------------------//
25            // The PatternNode class represents a single    //
26            // node in a binary translation tree.           //
27            //-----------------------------------------------//
28
29    class PatternNode
30    {
31        public:
32            PatternNode(Pattern (*in_func)(Phrase &), PatternNode * in_mat,
33                                PatternNode * in_nomat, Hint * in_hint);
34            ~PatternNode();
35            boolean process_response(Phrase expected_resp, Phrase user_resp);
36            boolean more_hints(Phrase expected_response, Phrase user_response);
37        private:
38            Hint * hint;
39            Pattern (*pattern_constructor)(Phrase &);
40            PatternNode * match_path_pattern;
41            PatternNode * no_match_path_pattern;
42    };
43
44    #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   phrasedescriptor.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the PhraseDescriptor class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef PHRASEDESCRIPTOR_H
19    #define PHRASEDESCRIPTOR_H
20
21    #include "compoundstring.h"
22    #include "worddescriptor.h"
23    #include "basephrase.h"
24
25            //-----------------------------------------------//
26            // A PhraseDescriptor represents the syntactic  //
27            // information to complete a semantically       //
28            // correct BasePhrase.                          //
29            //-----------------------------------------------//
30
31    class PhraseDescriptor
32    {
33        public:
34            PhraseDescriptor(int list_size, int * in_order, WordDescriptor * wd1,
35                             WordDescriptor * wd2=0, WordDescriptor * wd3=0,
36                             WordDescriptor * wd4=0, WordDescriptor * wd5=0,
37                             WordDescriptor * wd6=0, WordDescriptor * wd7=0,
38                             WordDescriptor * wd8=0, WordDescriptor * wd9=0,
39                             WordDescriptor * wd10=0);
40            ~PhraseDescriptor();
41            Phrase build_phrase(BasePhrase& in_bp);
42        private:
43            int size;
44            int * fillin_order;
45            WordDescriptor * * descriptors;
46    };
47
48    #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   phrasemodel.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the PhraseModel class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef PHRASE_MODEL_H
19    #define PHRASE_MODEL_H
20
21    #include "phrasedescriptor.h"
22    #include "basephrasepool.h"
23    #include "translationtree.h"
24
25            //-----------------------------------------------//
26            // A PhraseModel object represents everything    //
27            // needed to construct and present phrases, and  //
28            // diagnose translations, built from a specific  //
29            // syntactic model.                              //
30            //-----------------------------------------------//
31
32    class PhraseModel
33    {
34        public:
35            PhraseModel(int in_know_level,
36                        BasePhrasePool& pool, PhraseDescriptor& in_desc,
37                        TranslationTree& in_tree);
38            ~PhraseModel();
39            boolean do_exercise();
40            boolean reset_depletions(int in_level);
41            boolean set_level(int in_level);
42        private:
43            int knowledge_level;
44            BasePhrasePool * phrases;
45            PhraseDescriptor * fixed_phrase_desc;
46            TranslationTree * trans_tree;
47    };
48
49    #endif
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   phrasemodellist.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the PhraseModelList class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------
17
18  #ifndef PHRASEMODELLIST_H
19  #define PHRASEMODELLIST_H
20
21  #include "stdtype.h"
22
23          //-----------------------------------------------//
24          // This forward reference is used so that       //
25          // phrasemodellist.h can exist as a top-level   //
26          // makefile, while phrasemodel.h resides at the //
27          // component level.                             //
28          //-----------------------------------------------//
29
30  class PhraseModel;
31
32          //-----------------------------------------------//
33          // The PhraseModelList class represents the     //
34          // collection of all PhraseModels which the     //
35          // system supports.                             //
36          //-----------------------------------------------//
37
38  class PhraseModelList
39  {
40      public:
41          PhraseModelList(int in_size, PhraseModel * list);
42          ~PhraseModelList();
43          void do_exercise();
44          void reset_depletions();
45          void set_current_knowledge_level(int in_level);
46      private:
47          int size;
48          int current_level;
49          boolean * depleted;
50          PhraseModel * model_list;
51          boolean find_and_try();
52  };
53
54  #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   session.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines symbols used by the controlling
8     //              agent "session"
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-------------------------------------------------------------------------------
18
19    #ifndef SESSION_H
20    #define SESSION_H
21
22            //-----------------------------------------------//
23            // NOTE:  To ensure proper behavior of student   //
24            // monitoring, the major value should always be  //
25            // a multiple of the minor value.                //
26            //-----------------------------------------------//
27
28    #define MINOR_INTERVENE_MAX     50
29    #define MAJOR_INTERVENE_MAX     (3 * MINOR_INTERVENE_MAX)
30
31    #endif
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   statistics.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the run-time Statistics class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18  #ifndef STATISTICS_H
19  #define STATISTICS_H
20
21  #include <fstream.h>
22
23          //-----------------------------------------------//
24          // The Statistics class compiles run-time data   //
25          // for the current user's benefit, and to be     //
26          // be made available to the system               //
27          // administrator.                                //
28          //-----------------------------------------------//
29
30  class Statistics
31  {
32      public:
33          Statistics();
34          ~Statistics();
35          void increment_exercise_count();
36          void increment_attempt_count();
37          void increment_correct_count();
38          void print_summary();
39          void print_to_file(ofstream * stats_stream);
40          void read(ifstream * stats_stream);
41          void display();
42          int get_prev_duration();
43          int get_exercise_count();
44          int get_attempt_count();
45          int get_correct_count();
46      private:
47          int prev_duration;
48          int exercise_count;
49          int attempt_count;
50          int correct_count;
51  };
52
53  #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   statistics_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines symbols used by the statistics
8     //              class.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-------------------------------------------------------------------------------
18
19    #ifndef STATISTICS_DEFS_H
20    #define STATISTICS_DEFS_H
21
22            //-----------------------------------------------//
23            // Date/time stamp definitions.                  //
24            //-----------------------------------------------//
25
26    #define NUM_SECONDS_PER_MINUTE  60.0
27    #define NUM_SECONDS_PER_HOUR    (60.0 * NUM_SECONDS_PER_MINUTE)
28    #define NUM_SECONDS_PER_DAY     (24.0 * NUM_SECONDS_PER_HOUR)
29    #define MAX_SESSION_DURATION    NUM_SECONDS_PER_DAY
30
31    #endif
```

```
1   //-----------------------------------------------------------------------------
2   //
3   //  Filename:   stdtype.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines some extensions to "standard" data
8   //              types and/or values.
9   //
10  //  Notes:      The theory and design of this system have been documented
11  //              in the accompanying thesis report.
12  //
13  //  Revisions   By              Reason
14  //  ---------   --              ------
15  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16  //
17  //-----------------------------------------------------------------------------
18
19  #ifndef STDTYPE_H
20  #define STDTYPE_H
21
22          //-----------------------------------------------//
23          // The pattern_fragment type can take the       //
24          // place of Words as Patterns are constructed   //
25          // with embedded wildcards.                     //
26          //-----------------------------------------------//
27
28  typedef char* pattern_fragment;
29
30          //-----------------------------------------------//
31          // A boolean data type and its values.          //
32          //-----------------------------------------------//
33
34  typedef int boolean;
35
36  #define TRUE    1
37  #define FALSE   0
38
39  #endif
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   translationtree.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the TranslationTree class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------
17
18  #ifndef TRANSLATIONTREE_H
19  #define TRANSLATIONTREE_H
20
21  #include "stdtype.h"
22  #include "phrasedescriptor.h"
23  #include "basephrase.h"
24  #include "patternnode.h"
25
26          //-------------------------------------------//
27          // The TranslationTree class knows how to    //
28          // translate, and diagnose a particular phrase //
29          // model.                                    //
30          //-------------------------------------------//
31
32  class TranslationTree
33  {
34      public:
35          TranslationTree(BasePhrase (*in_bpt) (BasePhrase &),
36                                  PatternNode * in_root,
37                                  PhraseDescriptor * in_fixed_desc);
38          ~TranslationTree();
39          boolean check_response(BasePhrase in_phrase);
40      private:
41          BasePhrase (*in_phrase_translator) (BasePhrase &);
42          PatternNode * root_pattern;
43          PhraseDescriptor * fixed_phrase_desc;
44  };
45
46  #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   user.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the User class - a single
8     //              entry in the UserList of authorized users.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-------------------------------------------------------------------------------
18
19    #ifndef USER_H
20    #define USER_H
21
22    #include "fstream.h"
23    #include "stdtype.h"
24    #include "user_defs.h"
25
26            //-----------------------------------------------//
27            // The User class represents a single valid      //
28            // user of the system.                           //
29            //-----------------------------------------------//
30
31    class User
32    {
33        public:
34            User();
35            ~User();
36            boolean read(ifstream * user_stream);
37            Usertype match_user(char * in_string);
38            char * get_string();
39            int get_current_level();
40            boolean set_current_level(int in_level);
41            void write(ofstream * user_stream);
42        private:
43            int last_known_level;
44            Usertype type;
45            char * login_name;
46            char * full_name;
47    );
48
49    #endif
```

```
1     //-----------------------------------------------------------------------------
2     //
3     //  Filename:   user_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines some User-related symbols.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-----------------------------------------------------------------------------
17
18    #ifndef USER_DEFS_H
19    #define USER_DEFS_H
20
21            //---------------------------------------------//
22            // Different login user types.                 //
23            //---------------------------------------------//
24
25    enum Usertype
26    {
27            NO_USERTYPE,
28            STUDENT,
29            SYSTEM
30    };
31
32            //---------------------------------------------//
33            // Maximum sizes for user-related strings.     //
34            //---------------------------------------------//
35
36    #define MAX_LOGIN_NAME_STRING   25
37    #define MAX_FULL_NAME_STRING    100
38
39            //---------------------------------------------//
40            // Maximum number of different usernames       //
41            // supported.                                  //
42            //---------------------------------------------//
43
44    #define MAX_NUMBER_USERS        100
45
46            //---------------------------------------------//
47            // Current range of supported knowledge level  //
48            //---------------------------------------------//
49
50    #define MIN_KNOW_LEVEL 1
51    #define MAX_KNOW_LEVEL 6
52
53    #endif
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   userinterface.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the UserInterface class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18  #ifndef USERINTERFACE_H
19  #define USERINTERFACE_H
20
21  #include "hint.h"
22  #include "userinterface_defs.h"
23
24          //-----------------------------------------------//
25          // The UserInterface object allows for           //
26          // encapsulation of the actual U/I code.  Could //
27          // later be replaced with a "flashier"           //
28          // interface.                                    //
29          //-----------------------------------------------//
30
31  class UserInterface
32  {
33      public:
34          UserInterface();
35          ~UserInterface();
36          ControlCommand  wrong_answer();
37          ControlCommand  after_hint(Hint * in_hint);
38          ControlCommand  cant_diagnose();
39          ControlCommand  cant_diagnose_further();
40          ControlCommand  after_stats();
41          ControlCommand  sysadmin();
42          ControlCommand  after_view();
43          ControlCommand  after_cum();
44      private:
45  );
46
47  #endif
```

```
1     //-----------------------------------------------------------------------------
2     //
3     //  Filename:   userinterface_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines symbols related to the
8     //              UserInterface class.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-----------------------------------------------------------------------------
18
19    #ifndef USERINTERFACE_DEFS_H
20    #define USERINTERFACE_DEFS_H
21
22            //-----------------------------------------------//
23            // ControlCommand enumerates all possible       //
24            // menu selections.  They may not all be valid  //
25            // at a given prompt.                           //
26            //-----------------------------------------------//
27
28    enum ControlCommand
29    {
30            CC_NONE,
31            CC_try_again,
32            CC_hint,
33            CC_correct_answer,
34            CC_quit,
35            CC_exit,
36            CC_help,
37            CC_admin_write,
38            CC_admin_delete,
39            CC_admin_save,
40            CC_admin_diag_fail,
41            CC_admin_report_diag_fail,
42            CC_admin_stats,
43            CC_admin_report_stats,
44            CC_admin_cum_stats,
45            CC_admin_report_cum_stats,
46            CC_admin_report_full,
47            CC_admin_find_next
48    };
49
50    #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   userlist.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the UserList class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef USERLIST_H
19    #define USERLIST_H
20
21    #include "user_defs.h"
22    #include "stdtype.h"
23
24            //-----------------------------------------------//
25            // This forward reference is used so that        //
26            // userlist.h can exist as a top-level include   //
27            // file, while user.h resides at the component   //
28            // level.                                        //
29            //-----------------------------------------------//
30
31    class User;
32
33            //-----------------------------------------------//
34            // The UserList class represents the entire      //
35            // collection of valid users of the system.      //
36            //-----------------------------------------------//
37
38    class UserList
39    (
40        public:
41            UserList();
42            ~UserList();
43            Usertype establish_connection();
44            void read();
45            int get_current_level();
46            void write();
47        private:
48            int current_user;
49            User * * users;
50            int size;
51            boolean list_modified;
52    );
53
54    #endif
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   word.h
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This include file defines the Word class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18  #ifndef WORD_H
19  #define WORD_H
20
21  #include "stdtype.h"
22  #include "worddescriptor.h"
23  #include "word_and_phrase_defs.h"
24
25          //-----------------------------------------------//
26          // The Word class is basically a string with    //
27          // with attributes.                             //
28          //-----------------------------------------------//
29
30  class Word
31  {
32      public:
33          Word(char * in_static, WD_Type in_type, Count in_count, Gender
34                  in_gender, Language in_lang);   // Intended for knowledge base
35                                                  //    build.
36          Word();                                 // Intended for automatics.
37          Word(const Word &);                     // Intended for copies.
38          ~Word();
39          Word& operator=(Word & in_word);
40          boolean read();
41  #ifdef DEBUG
42          void debug_print();
43  #endif
44          void print();
45          void set_string(char *);
46          void set_type(WD_Type in_type);
47          void set_language(Language in_lang);
48          void set_gender(Gender in_gend);
49          void set_count(Count in_count);
50          void set_index(int in_index);
51          int get_index();
52          WD_Type get_type();
53          Count get_count();
54          Gender get_gender();
55          Language get_language();
56          char * get_string();
57      private:
58          char * word;
59          int index;
60          WD_Type type;
61          Count count;
62          Gender gender;
63          Language language;
64  };
65
66  #endif
```

```
1     //-----------------------------------------------------------------------------
2     //
3     //  Filename:   word_and_phrase_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines symbols related to the system's
8     //              use of Words and Phrases.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-----------------------------------------------------------------------------
18
19    #ifndef WORD_AND_PHRASE_DEFS_H
20    #define WORD_AND_PHRASE_DEFS_H
21
22            //-----------------------------------------------//
23            // This word length would typically be used      //
24            // for dimensioning when the actual length is    //
25            // not yet known.                                //
26            //-----------------------------------------------//
27
28    #define MAX_WORD_LENGTH 20
29
30            //-----------------------------------------------//
31            // The following sizes reflect the system's      //
32            // maximum Phrase and Pattern size.  The first   //
33            // is in Words, and the second in characters.    //
34            // This length represents the max size           //
35            // character string which can be built from      //
36            // a Phrase or Pattern.                          //
37            //-----------------------------------------------//
38
39    #define MAX_PHRASE_LENGTH               10
40    #define MAX_PHRASE_STRING_LENGTH        (MAX_PHRASE_LENGTH * MAX_WORD_LENGTH)
41
42            //-----------------------------------------------//
43            // The following enumerate various word          //
44            // attributes.  It is a convention that the      //
45            // first value be the DYNAMIC (i.e. to-be-       //
46            // assigned) value, and the highest symbol       //
47            // be the "none" or "not applicable" value.      //
48            //-----------------------------------------------//
49
50    enum Gender
51    {
52            G_DYNAMIC,
53            G_masculine,
54            G_feminine,
55            G_neuter,
56            G_none
57    };
58
59    enum Count
60    {
61            C_DYNAMIC,
62            C_singular,
63            C_plural,
64            C_none
65    };
66
```



```
67    enum Language
68    {
69            L_DYNAMIC,
70            L_english,
71            L_german,
72            L_none
73    };
74
75    #endif
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   worddescriptor.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines the WordDescriptor class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18    #ifndef WORDDESCRIPTOR_H
19    #define WORDDESCRIPTOR_H
20
21    #include "stdtype.h"
22    #include "worddescriptor_defs.h"
23    #include "word_and_phrase_defs.h"
24
25            //-----------------------------------------------//
26            // These forward references are used so that    //
27            // worddescriptor.h can exist as a top-level    //
28            // include file, while dynamicworddescriptor.h  //
29            // and word.h reside at the component level.    //
30            //-----------------------------------------------//
31
32    class Word;
33    class DynamicWordDescriptor;
34
35            //-----------------------------------------------//
36            // The WordDescriptor class specifies           //
37            // the attributes of a derivation of a word.    //
38            //-----------------------------------------------//
39
40
41    class WordDescriptor
42    {
43        public:
44            WordDescriptor(Count in_count, Gender in_gend, Language in_lang,
45                                                    WD_Type in_type);
46            WordDescriptor(Word & in_word);
47            ~WordDescriptor();
48            Count get_count();
49            Gender get_gender();
50            Language get_language();
51            void set_count(Count in_count);
52            void set_gender(Gender in_gender);
53            void set_language(Language in_language);
54            boolean no_dynamics();
55            void resolve_dynamics(DynamicWordDescriptor& in_dyna);
56            WD_Type type;
57        private:
58            Count count;
59            Gender gender;
60            Language language;
61    };
62
63    #endif
```

```
1     //-----------------------------------------------------------------------------
2     //
3     //  Filename:   worddescriptor_defs.h
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This include file defines symbols related to the
8     //              WordDescriptor class.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-----------------------------------------------------------------------------
18
19    #ifndef WORDDESCRIPTOR_DEFS_H
20    #define WORDDESCRIPTOR_DEFS_H
21
22            //---------------------------------------------//
23            // The various types of word descriptors.      //
24            //---------------------------------------------//
25
26    enum WD_Type
27    {
28            WD_UNKNOWN_TYPE,
29            article,
30            noun,
31            verb,
32            adjective
33    };
34
35    #endif
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   BPT_drop_fourth_word.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the base_phrase_translator function of
8   //              generating a BasePhrase, approriately modified to be correct
9   //              in the other language.  In this case, the fourth word in
10  //              the basephrase is dropped.
11  //
12  //  Notes:      The theory and design of this system have been documented
13  //              in the accompanying thesis report.
14  //
15  //  Revisions   By              Reason
16  //  ---------   --              ------
17  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18  //
19  //-------------------------------------------------------------------------------
20
21          //------------ Included files -------------------//
22
23  #include <iostream.h>
24  #include "stdtype.h"
25  #include "basephrase.h"
26
27          //-----------------------------------------------//
28          // Executable code begins here.                  //
29          //-----------------------------------------------//
30
31  BasePhrase BPT_drop_fourth_word(BasePhrase & in_basephrase)
32  {
33  #ifdef DEBUG
34      cerr << "\t\t\t\t\tENTRY BPT_drop_fourth_word()" << endl;
35  #endif
36
37      int  old_size = in_basephrase.get_size();
38
39      BasePhrase new_bp(  old_size-1,
40                          in_basephrase.get_index(1),
41                          in_basephrase.get_index(2),
42                          in_basephrase.get_index(3),
43                          in_basephrase.get_index(5)  );
44
45      return(new_bp);
46  }
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   BPT_no_translation.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the base_phrase_translator function of
8   //              generating a BasePhrase, approriately modified to be correct
9   //              in the other language.  In this case, no modifications are
10  //              needed.
11  //
12  //  Notes:      The theory and design of this system have been documented
13  //              in the accompanying thesis report.
14  //
15  //  Revisions   By              Reason
16  //  ---------   --              ------
17  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18  //
19  //-------------------------------------------------------------------------------
20
21          //------------ Included files -------------------//
22
23  #include <iostream.h>
24  #include "stdtype.h"
25  #include "basephrase.h"
26
27          //-----------------------------------------------//
28          // Executable code begins here.                  //
29          //-----------------------------------------------//
30
31  BasePhrase BPT_no_translation(BasePhrase & in_basephrase)
32  {
33  #ifdef DEBUG
34      cerr << "\t\t\t\t\tENTRY BPT_no_translation()" << endl;
35  #endif
36      return(in_basephrase);
37  }
```

```
1     //-------------------------------------------------------------------------
2     //
3     //  Filename:   PC_extra_article_fourth.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the pattern_constructor function of
8     //              generating a Pattern which will match the input Phrase,
9     //              regardless of whether or not an extra article has been
10    //              included in the fourth word position.
11    //
12    //  Notes:      The theory and design of this system have been documented
13    //              in the accompanying thesis report.
14    //
15    //  Revisions   By              Reason
16    //  ---------   --              ------
17    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18    //
19    //-------------------------------------------------------------------------
20
21            //----------- Included files ------------------//
22
23    #include <iostream.h>
24    #include "stdtype.h"
25    #include "compoundstring.h"
26
27            //----------- External Data --------------------//
28
29    extern  pattern_fragment  any_word;
30
31            //----------- Calling Syntax -------------------//
32
33    Pattern PC_extra_article_fourth(Phrase & in_phrase)
34    {
35
36            //----------- Automatics - objects -------------//
37
38        Phrase      bigger_phrase(in_phrase.get_size()+1);
39        Word        new_word;
40
41            //----------- Automatics - other ---------------//
42
43            //----------- Code begins ----------------------//
44
45    #ifdef DEBUG
46        cerr << "\t\t\t\t\tENTRY PC_extra_article_fourth()" << endl;
47    #endif
48
49        new_word.set_string(any_word);
50
51        bigger_phrase.set_word(1,in_phrase.get_word(1));
52        bigger_phrase.set_word(2,in_phrase.get_word(2));
53        bigger_phrase.set_word(3,in_phrase.get_word(3));
54        bigger_phrase.set_word(4,new_word);
55        bigger_phrase.set_word(5,in_phrase.get_word(4));
56
57        Pattern  wildcard_pattern(bigger_phrase);
58        return( wildcard_pattern );
59    }
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   PC_gender1_first_word.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the pattern_constructor function of
8   //              generating a Pattern which will match the expected Phrase,
9   //              with the first gender variation of the first word.
10  //
11  //  Notes:      The theory and design of this system have been documented
12  //              in the accompanying thesis report.
13  //
14  //  Revisions   By              Reason
15  //  ---------   --              ------
16  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
17  //
18  //-------------------------------------------------------------------------
19
20          //----------- Included files -------------------//
21
22  #include <iostream.h>
23  #include "stdtype.h"
24  #include "compoundstring.h"
25  #include "word.h"
26  #include "worddescriptor.h"
27  #include "dictionary.h"
28
29          //----------- External Data --------------------//
30
31  extern Dictionary dictionary;           //  The dictionary will be used to
32                                          //  obtain the gender variation of
33                                          //  the needed word.
34
35          //----------- External functions ---------------//
36
37  extern Gender utility_gender1_var(Gender);
38
39          //----------- Calling syntax -------------------//
40
41  Pattern PC_gender1_first_word(Phrase & in_phrase)
42  {
43
44          //----------- Automatics - objects -------------//
45
46      Pattern             new_pattern = in_phrase;
47      Word                old_word = in_phrase.get_word(1);
48      WordDescriptor      new_word_WD(old_word);
49
50          //----------- Automatics - other ---------------//
51
52      int         word_index = old_word.get_index();
53      Gender      new_gender;
54
55          //----------- Code begins ----------------------//
56
57  #ifdef DEBUG
58      cerr << "\t\t\t\t\tENTRY PC_gender1_first_word()" << endl;
59  #endif
60
61          //------------------------------------------------//
62          // Obtain the first gender variation of the     //
63          // current gender.                              //
64          //------------------------------------------------//
65
66      new_gender = utility_gender1_var(new_word_WD.get_gender());
```



```
67
68          //------------------------------------------------//
69          // Set this attribute in the WordDescriptor for //
70          // the new word.                                //
71          //------------------------------------------------//
72
73      new_word_WD.set_gender(new_gender);
74
75          //------------------------------------------------//
76          // Obtain the new gender variation of the word  //
77          // from the dictionary, and set this as the     //
78          // first word in the generated pattern.         //
79          //------------------------------------------------//
80
81      new_pattern.set_word( 1, dictionary.get_word( word_index, &new_word_WD ));
82
83      return( new_pattern );
84  }
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   PC_gender2_first_word.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the pattern_constructor function of
8   //              generating a Pattern which will match the expected Phrase,
9   //              with the second gender variation of the first word.
10  //
11  //  Notes:      The theory and design of this system have been documented
12  //              in the accompanying thesis report.
13  //
14  //  Revisions   By              Reason
15  //  ---------   --              ------
16  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
17  //
18  //-------------------------------------------------------------------------
19
20          //----------- Included files -------------------//
21
22  #include <iostream.h>
23  #include "stdtype.h"
24  #include "compoundstring.h"
25  #include "word.h"
26  #include "worddescriptor.h"
27  #include "dictionary.h"
28
29          //----------- External Data --------------------//
30
31  extern Dictionary dictionary;           //  The dictionary will be used to
32                                          //  obtain the gender variation of
33                                          //  the needed word.
34
35          //----------- External functions ---------------//
36
37  extern Gender utility_gender2_var(Gender);
38
39          //----------- Calling syntax -------------------//
40
41  Pattern PC_gender2_first_word(Phrase & in_phrase)
42  {
43
44          //----------- Automatics - objects -------------//
45
46      Pattern             new_pattern = in_phrase;
47      Word                old_word = in_phrase.get_word(1);
48      WordDescriptor      new_word_WD(old_word);
49
50          //----------- Automatics - other ---------------//
51
52      int         word_index = old_word.get_index();
53      Gender      new_gender;
54
55          //----------- Code begins ----------------------//
56
57  #ifdef DEBUG
58      cerr << "\t\t\t\t\tENTRY PC_gender2_first_word()" << endl;
59  #endif
60
61          //------------------------------------------------//
62          // Convert the current gender                     //
63          // attribute to its second variation.             //
64          //------------------------------------------------//
65
66      new_gender = utility_gender2_var(new_word_WD.get_gender());
```



```
67
68          //------------------------------------------------//
69          // Set this attribute in the WordDescriptor for //
70          // the new word.                                  //
71          //------------------------------------------------//
72
73      new_word_WD.set_gender(new_gender);
74
75          //------------------------------------------------//
76          // Obtain the new gender variation of the word  //
77          // from the dictionary, and set this as the     //
78          // first word in the generated pattern.         //
79          //------------------------------------------------//
80
81      new_pattern.set_word( 1, dictionary.get_word( word_index, &new_word_WD ));
82
83      return( new_pattern );
84  }
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   PC_nouncap_second_word.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the pattern_constructor function of
8   //              generating a Pattern which will match the input Phrase,
9   //              differing from the expected response by the noun not
10  //              being capitalized.
11  //
12  //  Notes:      The theory and design of this system have been documented
13  //              in the accompanying thesis report.
14  //
15  //  Revisions   By              Reason
16  //  ---------   --              ------
17  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18  //
19  //-------------------------------------------------------------------------
20
21          //----------- Included files -------------------//
22
23  #include <iostream.h>
24  #include "stdtype.h"
25  #include "compoundstring.h"
26
27          //----------- External Functions ---------------//
28
29  extern Word     utility_not_capitalized(Word in_word);
30
31          //----------- Calling Syntax -------------------//
32
33  Pattern PC_nouncap_second_word(Phrase & in_phrase)
34  (
35
36          //----------- Automatics - objects -------------//
37
38      Pattern     wildcard_pattern = in_phrase;
39      Word        new_word;
40
41          //----------- Automatics - other ---------------//
42
43          //----------- Code begins ----------------------//
44
45  #ifdef DEBUG
46      cerr << "\t\t\t\t\tENTRY PC_nouncap_second_word()" << endl;
47  #endif
48
49          //------------------------------------------------//
50          // Get the current word from the second          //
51          // position of the input phrase, and convert     //
52          // it to lower case.                             //
53          //------------------------------------------------//
54
55      new_word = utility_not_capitalized( in_phrase.get_word(2) );
56
57          //------------------------------------------------//
58          // Set the new word in the second position of    //
59          // the pattern, and return to the caller.        //
60          //------------------------------------------------//
61
62      wildcard_pattern.set_word( 2, new_word );
63      return( wildcard_pattern );
64  )
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   PC_wrong_first_and_second_word.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the pattern_constructor function of
8     //              generating a Pattern which will match the input Phrase,
9     //              regardless of whether or not the first and second words in
10    //              the Phrase actually match.
11    //
12    //  Notes:      The theory and design of this system have been documented
13    //              in the accompanying thesis report.
14    //
15    //  Revisions   By              Reason
16    //  ---------   --              ------
17    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18    //
19    //-------------------------------------------------------------------------------
20
21            //----------- Included files -------------------//
22
23    #include <iostream.h>
24    #include "stdtype.h"
25    #include "compoundstring.h"
26
27            //----------- External Data --------------------//
28
29    extern  pattern_fragment  any_word;
30
31            //----------- Calling Syntax -------------------//
32
33    Pattern PC_wrong_first_and_second_word(Phrase & in_phrase)
34    {
35
36            //----------- Automatics - objects -------------//
37
38        Pattern     wildcard_pattern = in_phrase;
39        Word        new_word1;
40        Word        new_word2;
41
42            //----------- Automatics - other ---------------//
43
44            //----------- Code begins ----------------------//
45
46    #ifdef DEBUG
47        cerr << "\t\t\t\t\tENTRY PC_wrong_first_and_second_word()" << endl;
48    #endif
49
50            //------------------------------------------------//
51            // Create two pattern fragments which will       //
52            // match any word, and insert them in the first  //
53            // two word positions of the generated pattern.  //
54            //------------------------------------------------//
55
56        new_word1.set_string(any_word);
57        new_word2.set_string(any_word);
58
59        wildcard_pattern.set_word( 1, new_word1 );
60        wildcard_pattern.set_word( 2, new_word2 );
61
62        return( wildcard_pattern );
63    }
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   PC_wrong_first_word.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the pattern_constructor function of
8     //              generating a Pattern which will match the input Phrase,
9     //              regardless of whether or not the first word in the
10    //              Phrase actually matches.
11    //
12    //  Notes:      The theory and design of this system have been documented
13    //              in the accompanying thesis report.
14    //
15    //  Revisions   By              Reason
16    //  ---------   --              ------
17    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18    //
19    //-------------------------------------------------------------------------------
20
21            //----------- Included files -------------------//
22
23    #include <iostream.h>
24    #include "stdtype.h"
25    #include "compoundstring.h"
26
27            //----------- External Data --------------------//
28
29    extern  pattern_fragment  any_word;
30
31            //----------- Calling Syntax -------------------//
32
33    Pattern PC_wrong_first_word(Phrase & in_phrase)
34    {
35
36            //----------- Automatics - objects -------------//
37
38        Pattern     wildcard_pattern = in_phrase;
39        Word        new_word;
40
41            //----------- Automatics - other ---------------//
42
43            //----------- Code begins ----------------------//
44
45    #ifdef DEBUG
46        cerr << "\t\t\t\t\tENTRY PC_wrong_first_word()" << endl;
47    #endif
48
49            //-----------------------------------------------//
50            // Generate a pattern fragment which will       //
51            // match any word, and insert it in the first   //
52            // word position of the generated pattern.      //
53            //-----------------------------------------------//
54
55        new_word.set_string(any_word);
56
57        wildcard_pattern.set_word( 1, new_word );
58        return( wildcard_pattern );
59    }
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   PC_wrong_fourth_word.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the pattern_constructor function of
8     //              generating a Pattern which will match the input Phrase,
9     //              regardless of whether or not the fourth word in the
10    //              Phrase actually matches.
11    //
12    //  Notes:      The theory and design of this system have been documented
13    //              in the accompanying thesis report.
14    //
15    //  Revisions   By              Reason
16    //  ---------   --              ------
17    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18    //
19    //-------------------------------------------------------------------------------
20
21            //------------ Included files -------------------//
22
23    #include <iostream.h>
24    #include "stdtype.h"
25    #include "compoundstring.h"
26
27            //------------ External Data --------------------//
28
29    extern  pattern_fragment  any_word;
30
31            //------------ Calling Syntax -------------------//
32
33    Pattern PC_wrong_fourth_word(Phrase & in_phrase)
34    {
35
36            //------------ Automatics - objects -------------//
37
38        Pattern     wildcard_pattern = in_phrase;
39        Word        new_word;
40
41            //------------ Automatics - other ---------------//
42
43            //------------ Code begins ----------------------//
44
45    #ifdef DEBUG
46        cerr << "\t\t\t\t\tENTRY PC_wrong_fourth_word()" << endl;
47    #endif
48
49            //-----------------------------------------------//
50            // Create a pattern fragment which will match   //
51            // any word, and insert it in the fourth word   //
52            // position of the generated pattern.           //
53            //-----------------------------------------------//
54
55        new_word.set_string(any_word);
56
57        wildcard_pattern.set_word( 4, new_word );
58        return( wildcard_pattern );
59    }
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   PC_wrong_second_word.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the pattern_constructor function of
8     //              generating a Pattern which will match the input Phrase,
9     //              regardless of whether or not the second word in the
10    //              Phrase actually matches.
11    //
12    //  Notes:      The theory and design of this system have been documented
13    //              in the accompanying thesis report.
14    //
15    //  Revisions   By              Reason
16    //  ---------   --              ------
17    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18    //
19    //-------------------------------------------------------------------------------
20
21            //----------- Included files -------------------//
22
23    #include <iostream.h>
24    #include "stdtype.h"
25    #include "compoundstring.h"
26
27            //----------- External Data --------------------//
28
29    extern  pattern_fragment  any_word;
30
31            //----------- Calling Syntax -------------------//
32
33    Pattern PC_wrong_second_word(Phrase & in_phrase)
34    {
35
36            //----------- Automatics - objects -------------//
37
38        Pattern     wildcard_pattern = in_phrase;
39        Word        new_word;
40
41            //----------- Automatics - other ---------------//
42
43            //----------- Code begins ----------------------//
44
45    #ifdef DEBUG
46        cerr << "\t\t\t\t\tENTRY PC_wrong_second_word()" << endl;
47    #endif
48
49            //------------------------------------------------//
50            // Create a pattern fragment which will match   //
51            // any word, and insert it in the second word   //
52            // position of the generated pattern.           //
53            //------------------------------------------------//
54
55        new_word.set_string(any_word);
56
57        wildcard_pattern.set_word( 2, new_word );
58        return( wildcard_pattern );
59    }
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   PC_wrong_third_word.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the pattern_constructor function of
8     //              generating a Pattern which will match the input Phrase,
9     //              regardless of whether or not the third word in the
10    //              Phrase actually matches.
11    //
12    //  Notes:      The theory and design of this system have been documented
13    //              in the accompanying thesis report.
14    //
15    //  Revisions   By              Reason
16    //  ---------   --              ------
17    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18    //
19    //-------------------------------------------------------------------------------
20
21            //----------- Included files -------------------//
22
23    #include <iostream.h>
24    #include "stdtype.h"
25    #include "compoundstring.h"
26
27            //----------- External Data --------------------//
28
29    extern  pattern_fragment  any_word;
30
31            //----------- Calling Syntax -------------------//
32
33    Pattern PC_wrong_third_word(Phrase & in_phrase)
34    {
35
36            //----------- Automatics - objects -------------//
37
38        Pattern     wildcard_pattern = in_phrase;
39        Word        new_word;
40
41            //----------- Automatics - other ---------------//
42
43            //----------- Code begins ----------------------//
44
45    #ifdef DEBUG
46        cerr << "\t\t\t\t\tENTRY PC_wrong_third_word()" << endl;
47    #endif
48
49            //-----------------------------------------------//
50            // Create a pattern fragment which will match   //
51            // any word, and insert it into the third word  //
52            // position of the generated pattern.           //
53            //-----------------------------------------------//
54
55        new_word.set_string(any_word);
56
57        wildcard_pattern.set_word( 3, new_word );
58        return( wildcard_pattern );
59    }
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   basephrase.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the BasePhrase class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include "basephrase.h"
22  #include "dictionary.h"
23
24          //----------- External data --------------------//
25
26  extern Dictionary dictionary;
27
28          //------------------------------------------------//
29          // Constructor 1.                                 //
30          //------------------------------------------------//
31
32  BasePhrase::BasePhrase(int in_length, int index1, int index2, int index3,
33                         int index4, int index5, int index6, int index7,
34                         int index8, int index9, int index10)
35  {
36  #ifdef DEBUG
37      cerr << "\t\t\t\t\tENTRY    BasePhrase CONSTRUCTOR 1" << endl;
38      cerr << "\t\t\t\t\tENTRY    (for creating)" << endl;
39  #endif
40
41      int current_level;
42
43      indices = new int[in_length];
44      length = in_length;
45      knowledge_level = 0;
46
47          //------------------------------------------------//
48          // For each word index which was input, store     //
49          // it in the appropriate position in the          //
50          // internal array of indices, and check to see    //
51          // if it represents the highest knowledge         //
52          // level yet encountered.  If so, adjust the      //
53          // current_level, so that upon completion, the    //
54          // knowledge level of the BasePhrase is equal     //
55          // to the highest knowledge level of any of its   //
56          // words.                                         //
57          //------------------------------------------------//
58
59      switch(length)
60      {
61          case 10:
62                  indices[9] = index10;
63                  current_level = dictionary.get_level(index10);
64                  if (current_level > knowledge_level)
65                  {
66                      knowledge_level = current_level;
```



```
67                  }
68          case 9:
69                  indices[8] = index9;
70                  current_level = dictionary.get_level(index9);
71                  if (current_level > knowledge_level)
72                  {
73                      knowledge_level = current_level;
74                  }
75          case 8:
76                  indices[7] = index8;
77                  current_level = dictionary.get_level(index8);
78                  if (current_level > knowledge_level)
79                  {
80                      knowledge_level = current_level;
81                  }
82          case 7:
83                  indices[6] = index7;
84                  current_level = dictionary.get_level(index7);
85                  if (current_level > knowledge_level)
86                  {
87                      knowledge_level = current_level;
88                  }
89          case 6:
90                  indices[5] = index6;
91                  current_level = dictionary.get_level(index6);
92                  if (current_level > knowledge_level)
93                  {
94                      knowledge_level = current_level;
95                  }
96          case 5:
97                  indices[4] = index5;
98                  current_level = dictionary.get_level(index5);
99                  if (current_level > knowledge_level)
100                 {
101                     knowledge_level = current_level;
102                 }
103         case 4:
104                 indices[3] = index4;
105                 current_level = dictionary.get_level(index4);
106                 if (current_level > knowledge_level)
107                 {
108                     knowledge_level = current_level;
109                 }
110         case 3:
111                 indices[2] = index3;
112                 current_level = dictionary.get_level(index3);
113                 if (current_level > knowledge_level)
114                 {
115                     knowledge_level = current_level;
116                 }
117         case 2:
118                 indices[1] = index2;
119                 current_level = dictionary.get_level(index2);
120                 if (current_level > knowledge_level)
121                 {
122                     knowledge_level = current_level;
123                 }
124         case 1:
125                 indices[0] = index1;
126                 current_level = dictionary.get_level(index1);
127                 if (current_level > knowledge_level)
128                 {
129                     knowledge_level = current_level;
130                 }
131                 break;
132         default:
```

```
133                     cerr << "\t\t\tERROR BasePhrase invalid length specified" << end
    l;
134 #ifdef DEBUG
135                     cout << "\t\t\tERROR BasePhrase invalid length specified" << end
    l;
136 #endif
137                     break;
138         }
139
140 #ifdef DEBUG
141                     cerr << "\t\t\tDEBUG Knowledge level of basephrase is " << knowl
    edge_level << endl;
142 #endif
143 );
144
145             //-----------------------------------------------//
146             // Constructor 2.                                //
147             //-----------------------------------------------//
148
149 BasePhrase::BasePhrase(const BasePhrase & in_BP)
150 {
151 #ifdef DEBUG
152     cerr << "\t\t\t\t\tENTRY    BasePhrase CONSTRUCTOR 2" << endl;
153     cerr << "\t\t\t\t\tENTRY    (for copies)" << endl;
154 #endif
155
156             //-----------------------------------------------//
157             // This is a Copy Constructor.  Create a new     //
158             // index list, and set the values equal to the   //
159             // input BasePhrase.                             //
160             //-----------------------------------------------//
161
162     length = in_BP.length;
163     indices = new int[length];
164     for (int j=0; j<length; j++)
165     {
166         indices[j] = in_BP.indices[j];
167     }
168 );
169
170             //-----------------------------------------------//
171             // Assignment operator.                          //
172             //-----------------------------------------------//
173
174 BasePhrase& BasePhrase::operator=(BasePhrase & in_BP)
175 {
176 #ifdef DEBUG
177     cerr << "\t\t\t\t\tENTRY    BasePhrase::operator=" << endl;
178
179     cerr << "\t\t\tDEBUG in_BP len " << in_BP.length << " ind 1 is ";
180     cerr << in_BP.indices[0] << " ind 2 is " << in_BP.indices[1] << endl;
181 #endif
182
183             //-----------------------------------------------//
184             // If assignment to self, do nothing.            //
185             // Otherwise, clear the internal data, and       //
186             // create new elements to match the right-hand   //
187             // side of the assignment.                       //
188             //-----------------------------------------------//
189
190     if (&in_BP != this)
191     {
192         for (int i=0; i<length; i++)
193         {
194             indices[i] = 0;
195         }
```



```
196         delete [] indices;
197
198         length = in_BP.length;
199         indices = new int[length];
200         for (int j=0; j<length; j++)
201         {
202             indices[j] = in_BP.indices[j];
203         }
204     }
205     return *this;
206 );
207
208             //-----------------------------------------------//
209             // Destructor.                                   //
210             //-----------------------------------------------//
211
212 BasePhrase::~BasePhrase()
213 {
214 #ifdef DEBUG
215     cerr << "\t\t\t\t\tENTRY    BasePhrase DESTRUCTOR" << endl;
216 #endif
217
218     delete [] indices;
219 );
220
221             //-----------------------------------------------//
222             // Member function get_index().                  //
223             //-----------------------------------------------//
224
225 int BasePhrase::get_index(int position)
226 {
227 #ifdef DEBUG
228     cerr << "\t\t\t\t\tENTRY BasePhrase::get_index " << endl;
229
230     cerr << "\t\t\tDEBUG    returning the index at array location [";
231     cerr << position-1;
232     cerr << "]" << endl;
233     cerr << "\t\t\tDEBUG       which is ";
234     cerr << indices[position-1];
235     cerr << endl;
236 #endif
237
238             //----------- Automatics - objects -------------//
239
240             //----------- Automatics - other ---------------//
241
242             //----------- Code begins ----------------------//
243
244     return indices[position-1];
245 )
246
247             //-----------------------------------------------//
248             // Member function get_size().                   //
249             //-----------------------------------------------//
250
251 int BasePhrase::get_size()
252 {
253 #ifdef DEBUG
254     cerr << "\t\t\t\t\tENTRY BasePhrase::get_size" << endl;
255 #endif
256
257             //----------- Automatics - objects -------------//
258
259             //----------- Automatics - other ---------------//
260
261             //----------- Code begins ----------------------//
```

```
262
263         return(length);
264     }
265
266             //-----------------------------------------------//
267             // Member function get_level().                  //
268             //-----------------------------------------------//
269
270     int BasePhrase::get_level()
271     {
272     #ifdef DEBUG
273         cerr << "\t\t\t\t\tENTRY BasePhrase::get_level" << endl;
274     #endif
275
276             //------------ Automatics - objects -------------//
277
278             //------------ Automatics - other ---------------//
279
280             //------------ Code begins ----------------------//
281
282         return(knowledge_level);
283     }
```

```
1   //--------------------------------------------------------------------------------
2   //
3   //  Filename:   basephraselist.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the BasePhraseList class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ----------  --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //--------------------------------------------------------------------------------
17
18          //------------ Included files --------------------//
19
20  #include <iostream.h>
21  #include "basephraselist.h"
22  #include <stdlib.h>
23
24          //-------------------------------------------------//
25          // Constructor.                                    //
26          //-------------------------------------------------//
27
28  BasePhraseList::BasePhraseList(int list_size, BasePhrase * first_phrase)
29  {
30  #ifdef DEBUG
31      cerr << "\t\t\t\t\tENTRY    BasePhraseList CONSTRUCTOR" << endl;
32  #endif
33
34          //-------------------------------------------------//
35          // Create a list of BasePhrase pointers, and       //
36          // initialize from input list.                     //
37          //-------------------------------------------------//
38
39      phrases = new BasePhrase * [list_size];
40      selected = new boolean[list_size];
41
42      BasePhrase * phrase_ptr = first_phrase;
43      for (int i = 0; i < list_size; i++)
44      {
45          phrases[i] = phrase_ptr++;
46          selected[i] = FALSE;
47      }
48      size = list_size;
49
50  };
51
52          //-------------------------------------------------//
53          // Destructor.                                     //
54          //-------------------------------------------------//
55
56  BasePhraseList::~BasePhraseList()
57  {
58  #ifdef DEBUG
59      cerr << "\t\t\t\t\tENTRY    BasePhraseList DESTRUCTOR" << endl;
60  #endif
61
62      delete [] phrases;
63      delete [] selected;
64  };
65
66          //-------------------------------------------------//
```



```
67          // Member function get_BasePhrase().               //
68          //-------------------------------------------------//
69
70  BasePhrase * BasePhraseList::get_BasePhrase()
71  {
72  #ifdef DEBUG
73      cerr << "\t\t\t\t\tENTRY BasePhraseList::get_BasePhrase " << endl;
74  #endif
75
76          //------------ Automatics - objects -------------//
77
78          //------------ Automatics - other ---------------//
79
80      BasePhrase *        BP_ptr = NULL;
81      int                 num_unselected = 0;
82      int                 rand_val = rand();
83      int                 chosen_count;
84      int                 chosen_index;
85
86          //------------ Code begins ----------------------//
87
88
89          //-------------------------------------------------//
90          // First, determine how many BasePhrases in the    //
91          // list have not yet been selected.                //
92          //-------------------------------------------------//
93
94      for (int i=0; i<size; i++)
95      {
96          if (selected[i] == FALSE)
97          {
98              num_unselected++;
99          }
100     }
101
102         //-------------------------------------------------//
103         // If there are no unused BasePhrases, do          //
104         // nothing (BP_ptr of NULL will be returned to     //
105         // the caller).  Otherwise, pick a pseudo-         //
106         // random number in the range of available         //
107         // BasePhrases.                                    //
108         //-------------------------------------------------//
109
110     if (num_unselected != 0)
111     {
112
113         chosen_count = rand_val % num_unselected;
114 #ifdef DEBUG
115         cerr << "\t\t\tDEBUG number of unselected entries is " << num_unselected
     << endl;
116         cerr << "\t\t\tDEBUG random value obtained is " << rand_val << endl;
117         cerr << "\t\t\tDEBUG chosen count is modulo above 2 and is " << chosen_c
    ount << endl;
118 #endif
119         chosen_count++;
120 #ifdef DEBUG
121         cerr << "\t\t\tDEBUG chosen count final value is " << chosen_count << en
    dl;
122 #endif
123         //-------------------------------------------------//
124         // Use this chosen random index to count into      //
125         // previously unselected entries.                  //
126         //-------------------------------------------------//
127
128         for (chosen_index=0; chosen_count > 0; chosen_index++)
129         {
```

```
130             if (selected[chosen_index] == FALSE)
131             {
132                 chosen_count--;
133             }
134         }
135         chosen_index--;
136
137 #ifdef DEBUG
138         cerr << "\t\t\tDEBUG returning the phrase at array location [";
139         cerr << chosen_index << "]" << endl;
140         cerr << "\t\t\tDEBUG    which is:" << endl;
141
142         //  NOTE: get_index currently does some debug output., so let it.
143         int temp;
144         temp = phrases[chosen_index]->get_index(1);
145         temp = phrases[chosen_index]->get_index(2);
146 #endif
147         //-----------------------------------------------//
148         // Now that a pseudo-random entry has been      //
149         // selected, mark it as "used", and return the  //
150         // BasePhrase pointer to the caller.            //
151         //-----------------------------------------------//
152
153         selected[chosen_index] = TRUE;
154         BP_ptr = phrases[chosen_index];
155     }
156     return(BP_ptr);
157 );
158
159         //-----------------------------------------------//
160         // Member function reset_selections().          //
161         //-----------------------------------------------//
162
163 boolean BasePhraseList::reset_selections(int in_knowledge_level)
164 {
165 #ifdef DEBUG
166     cerr << "\t\t\t\t\tENTRY BasePhraseList::reset_selections" << endl;
167 #endif
168
169         //----------- Automatics - objects -------------//
170
171         //----------- Automatics - other ---------------//
172
173     int this_basephrase_level;
174     int number_unselected = 0;
175
176         //----------- Code begins ----------------------//
177
178         //-----------------------------------------------//
179         // Loop through all entries, and mark all that  //
180         // fall in the appropriate knowledge level      //
181         // range as "not selected" (meaning that they   //
182         // are available to be chosen for exercises     //
183         // during the current session.                  //
184         //-----------------------------------------------//
185
186     for (int i = 0; i < size; i++)
187     {
188         this_basephrase_level = phrases[i]->get_level();
189         if (this_basephrase_level <= in_knowledge_level)
190         {
191             number_unselected++;
192             selected[i] = FALSE;
193         }
194         else
195         {
```



```
196             selected[i] = TRUE;
197         }
198     }
199
200         //-----------------------------------------------//
201         // Return FALSE if there are no available       //
202         // entries in this BasePhraseList.              //
203         //-----------------------------------------------//
204
205     if (number_unselected > 0)
206     {
207         return(TRUE);
208     }
209     else
210     {
211         return(FALSE);
212     }
213
214 );
215
216         //-----------------------------------------------//
217         // Member function set_level().                 //
218         //-----------------------------------------------//
219
220 boolean BasePhraseList::set_level(int in_knowledge_level)
221 {
222 #ifdef DEBUG
223     cerr << "\t\t\t\t\tENTRY BasePhraseList::set_level" << endl;
224 #endif
225
226         //-----------------------------------------------//
227         // Same functionality as reset_selections(),    //
228         // but invoked during system initialization to  //
229         // set the initial availability based on the    //
230         // current user's knowledge level.              //
231         //-----------------------------------------------//
232
233     return(this->reset_selections(in_knowledge_level));
234 }
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   basephrasepool.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the BasePhrasePool class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //              It would probably be nice if the constructor took a pointer
13  //              to a variable length list of BasePhraseLists, as opposed
14  //              to the individual arguments currently used.
15  //
16  //  Revisions   By              Reason
17  //  ---------   --              ------
18  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
19  //
20  //-------------------------------------------------------------------------
21
22          //----------- Included files -------------------//
23
24  #include <iostream.h>
25  #include "basephrasepool.h"
26  #include <stdlib.h>
27
28          //-----------------------------------------------//
29          // Constructor.                                  //
30          //-----------------------------------------------//
31
32  BasePhrasePool::BasePhrasePool(int pool_size, BasePhraseList * list1,
33                          BasePhraseList * list2, BasePhraseList * list3,
34                          BasePhraseList * list4, BasePhraseList * list5,
35                          BasePhraseList * list6, BasePhraseList * list7,
36                          BasePhraseList * list8, BasePhraseList * list9,
37                          BasePhraseList * list10)
38  {
39  #ifdef DEBUG
40      cerr << "\t\t\t\t\tENTRY    BasePhrasePool CONSTRUCTOR" << endl;
41  #endif
42
43      size = pool_size;
44      lists = new BasePhraseList * [size];
45      selected = new boolean[size];
46
47          //-----------------------------------------------//
48          // Mark all BasePhraseList choices as initially //
49          // unselected (meaning not yet exhausted).      //
50          //-----------------------------------------------//
51
52      for (int i=0; i<pool_size; i++)
53      {
54          selected[i] = FALSE;
55      }
56
57          //-----------------------------------------------//
58          // Set list entries based on input.              //
59          //-----------------------------------------------//
60
61      switch(size)
62      {
63          case 10:
64                  lists[9] = list10;
65          case 9:
66                  lists[8] = list9;
```



```
67          case 8:
68                  lists[7] = list8;
69          case 7:
70                  lists[6] = list7;
71          case 6:
72                  lists[5] = list6;
73          case 5:
74                  lists[4] = list5;
75          case 4:
76                  lists[3] = list4;
77          case 3:
78                  lists[2] = list3;
79          case 2:
80                  lists[1] = list2;
81          case 1:
82                  lists[0] = list1;
83                  break;
84          default:
85                  cerr << "\t\t\tERROR BasePhrasePool invalid length specified" <<
     endl;
86  #ifdef DEBUG
87                  cout << "\t\t\tERROR BasePhrasePool invalid length specified" <<
     endl;
88  #endif
89      }
90  };
91
92          //-----------------------------------------------//
93          // Destructor.                                   //
94          //-----------------------------------------------//
95
96  BasePhrasePool::~BasePhrasePool()
97  {
98  #ifdef DEBUG
99      cerr << "\t\t\t\t\tENTRY    BasePhrasePool DESTRUCTOR" << endl;
100 #endif
101
102     delete [] lists;
103     delete [] selected;
104 };
105
106         //-----------------------------------------------//
107         // Member function get_BasePhrase().             //
108         //-----------------------------------------------//
109
110 BasePhrase * BasePhrasePool::get_BasePhrase()
111 {
112 #ifdef DEBUG
113     cerr << "\t\t\t\t\tENTRY BasePhrasePool::get_BasePhrase " << endl;
114 #endif
115
116         //----------- Automatics - objects -------------//
117
118         //----------- Automatics - other ---------------//
119
120     BasePhrase *        BP_ptr = NULL;
121     boolean             still_looking = TRUE;
122     int                 num_unselected = 0;
123     int                 rand_val;
124     int                 chosen_count;
125     int                 chosen_index;
126
127         //----------- Code begins ----------------------//
128
129         //-----------------------------------------------//
130         // Determine how many BasePhraseLists have not  //
```

```
131          // yet been exhausted.                          //
132          //-------------------------------------------------//
133
134      for (int i=0; i<size; i++)
135      {
136          if (selected[i] == FALSE)
137          {
138              num_unselected++;
139          }
140      }
141
142          //-------------------------------------------------//
143          // Loop through unexhausted lists, requesting   //
144          // a BasePhrase, until one is obtained.         //
145          //-------------------------------------------------//
146
147      while (still_looking)
148      {
149
150          if (num_unselected == 0)
151          {
152              still_looking = FALSE;
153          }
154          else
155          {
156          //-------------------------------------------------//
157          // Get a pseudo-random number in the range      //
158          // of currently non-exhausted Lists.            //
159          //-------------------------------------------------//
160
161              rand_val = rand();
162              chosen_count = rand_val % num_unselected;
163  #ifdef DEBUG
164              cerr << "\t\t\tDEBUG number of undepleted entries is " << num_unsele
    cted << endl;
165              cerr << "\t\t\tDEBUG random value obtained is " << rand_val << endl;
166              cerr << "\t\t\tDEBUG chosen count is modulo above 2 and is " << chos
    en_count << endl;
167  #endif
168              chosen_count++;
169  #ifdef DEBUG
170              cerr << "\t\t\tDEBUG chosen count final value is " << chosen_count <
    < endl;
171  #endif
172          //-------------------------------------------------//
173          // Use the chosen random index to count into    //
174          // the lists to request a BasePhrase.           //
175          //-------------------------------------------------//
176
177              for (chosen_index=0; chosen_count > 0; chosen_index++)
178              {
179                  if (selected[chosen_index) == FALSE)
180                  {
181                      chosen_count--;
182                  }
183              }
184              chosen_index--;
185
186  #ifdef DEBUG
187              cerr << "\t\t\tDEBUG using the list at array location [";
188              cerr << chosen_index << "]" << endl;
189  #endif
190          //-------------------------------------------------//
191          // A BasePhraseList may report that it has      //
192          // no more unused BasePhrases at this point.    //
193          // If this happens, mark the Lists entry as     //
```



```
194          // exhausted, and continue looking.             //
195          //-------------------------------------------------//
196
197              BP_ptr = lists[chosen_index]->get_BasePhrase();
198              if (BP_ptr == NULL)
199              {
200                  num_unselected--;
201                  selected[chosen_index) = TRUE;
202                  still_looking = TRUE;
203              }
204              else
205              {
206                  still_looking = FALSE;
207              }
208          }
209      }
210
211      return BP_ptr;
212  };
213
214          //-------------------------------------------------//
215          // Member function reset_selections().          //
216          //-------------------------------------------------//
217
218  boolean BasePhrasePool::reset_selections(int in_knowledge_level)
219  {
220  #ifdef DEBUG
221      cerr << "\t\t\t\t\tENTRY BasePhrasePool::reset_selections" << endl;
222  #endif
223
224          //----------- Automatics - objects -------------//
225
226          //----------- Automatics - other ---------------//
227
228      int number_active_lists = 0;
229
230          //----------- Code begins ----------------------//
231
232          //-------------------------------------------------//
233          // Loop through each BasePhraseList, requesting //
234          // that they reset their BasePhrase selections  //
235          // using the current user's knowledge level.    //
236          //-------------------------------------------------//
237
238      for (int i=0; i<size; i++)
239      {
240  #ifdef DEBUG
241      cerr << "\t\t\tDEBUG in BPP::reset_sel, resetting sel for list " << i << end
    l;
242  #endif
243          if (lists[i)->reset_selections(in_knowledge_level) == TRUE)
244          {
245  #ifdef DEBUG
246      cerr << "\t\t\tDEBUG BPL reports it has active BPs. " << endl;
247  #endif
248              selected[i) = FALSE;
249              number_active_lists++;
250          }
251          else
252          {
253  #ifdef DEBUG
254      cerr << "\t\t\tDEBUG BPL reports it does not have active BPs. " << endl;
255  #endif
256              selected[i) = TRUE;
257          }
258      }
```

```
259
260             //-----------------------------------------------//
261             // If the reset process causes all BasePhrase    //
262             // Lists to report that they have no available   //
263             // BasePhrase selections, return FALSE to the    //
264             // caller.                                       //
265             //-----------------------------------------------//
266
267         if (number_active_lists > 0)
268         {
269             return(TRUE);
270         }
271         else
272         {
273             return(FALSE);
274         }
275     };
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   CompoundString.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the CompoundString class, and its
8   //              derived classes Phrase and Pattern.
9   //
10  //  Notes:      The theory and design of this system have been documented
11  //              in the accompanying thesis report.
12  //
13  //              The get_string() member function returns a pointer to the
14  //              internal string.  This should probably be improved, but for
15  //              the time being, it is important for the caller to realize
16  //              that the pointer can only be used while this object is in
17  //              scope.
18  //
19  //  Revisions   By              Reason
20  //  ---------   --              ------
21  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
22  //
23  //-------------------------------------------------------------------------
24
25          //----- Regular expression processing defs -----//
26
27  #define INIT            register char *sp = instring;
28  #define GETC()          (*sp++)
29  #define PEEKC()         (*sp)
30  #define UNGETC(c)       (--sp)
31  #define RETURN(c)       return(c);
32  #define ERROR(c)        printf("regexp() error %d\n",c)
33
34  #define         EXPBUF_SIZE     250
35
36          //----------- Included files -------------------//
37
38  #include <iostream.h>
39  #include "compoundstring.h"
40  #include "word.h"
41  #include <stdio.h>
42  #include <regexp.h>
43  #include <string.h>
44  #include "word_and_phrase_defs.h"
45
46  //------------------------- Base class ---------------------------//
47
48          //-----------------------------------------------//
49          // Constructor.                                  //
50          //-----------------------------------------------//
51
52  CompoundString::CompoundString()
53  {
54  #ifdef DEBUG
55      cerr << "\t\t\t\t\tENTRY    CompoundString CONSTRUCTOR" << endl;
56  #endif
57
58          //-----------------------------------------------//
59          // Create an array of Word pointers, and         //
60          // initialize all entries to NULL.               //
61          //-----------------------------------------------//
62
63      size = MAX_PHRASE_LENGTH;
64      words = new Word * [size];
65      for (int i=0; i<size; i++)
66      {
```



```
67          words[i] = NULL;
68      }
69  );
70
71          //-----------------------------------------------//
72          // Destructor.                                   //
73          //-----------------------------------------------//
74
75  CompoundString::~CompoundString()
76  {
77  #ifdef DEBUG
78      cerr << "\t\t\t\t\tENTRY    CompoundString DESTRUCTOR" << endl;
79  #endif
80
81          //-----------------------------------------------//
82          // Loop through the words array, and if any      //
83          // of the entries point to Words, delete those   //
84          // words.  Then delete the array.                //
85          //-----------------------------------------------//
86
87      for (int i=0; i<size; i++)
88      {
89          if (words[i] != NULL)
90          {
91                  delete words[i];
92          }
93      }
94      delete [] words;
95  );
96
97          //-----------------------------------------------//
98          // Assignment operator.                          //
99          //-----------------------------------------------//
100
101 CompoundString& CompoundString::operator=(CompoundString & in_CS)
102 {
103 #ifdef DEBUG
104     cerr << "\t\t\t\t\tENTRY    CompoundString::operator=" << endl;
105 #endif
106
107         //-----------------------------------------------//
108         // If assignment to itself, don't do anything,   //
109         // otherwise, loop through the Word array,       //
110         // deleting any words found.  Then delete        //
111         // the array itself.                             //
112         //-----------------------------------------------//
113
114     if (&in_CS != this)
115     {
116         for (int i=0; i<size; i++)
117         {
118             if (words[i] != NULL)
119             {
120                 delete words[i];
121             }
122         }
123         delete [] words;
124
125         //-----------------------------------------------//
126         // Create a new words array, of size to match    //
127         // the CompoundString on the right-hand side     //
128         // of the assignment.  Loop through the right-   //
129         // hand Compound string, and create new word     //
130         // entries to match any found.                   //
131         //-----------------------------------------------//
132
```

```
133         size = in_CS.size;
134         words = new Word * [size];
135         for (int j=0; j<size; j++)
136         {
137             if (in_CS.words[j] != NULL)
138             {
139                 words[j] = new Word(*(in_CS.words[j]));
140             }
141         }
142     }
143     return *this;
144 };
145
146         //-----------------------------------------------//
147         // Member function set_word().                   //
148         //-----------------------------------------------//
149
150 void CompoundString::set_word(int position, Word in_word)
151 {
152 #ifdef DEBUG
153     cerr << "\t\t\t\t\tENTRY CompoundString::set_word" << endl;
154 #endif
155
156         //----------- Automatics - objects -------------//
157
158         //----------- Automatics - other ---------------//
159
160         //----------- Code begins ----------------------//
161
162         //-----------------------------------------------//
163         // Make sure the specified position is within    //
164         // the bounds of this CompoundString.            //
165         //-----------------------------------------------//
166
167     if (position > size)
168     {
169         cerr << "\t\t\tERROR set_word at invalid position." << endl;
170 #ifdef DEBUG
171         cout << "\t\t\tERROR set_word at invalid position." << endl;
172 #endif
173     }
174     else
175     {
176         //-----------------------------------------------//
177         // If this position already points to a Word,    //
178         // delete it.  Create a new Word and set it      //
179         // from the input word.                          //
180         //-----------------------------------------------//
181
182         if (words[position-1] != NULL)
183         {
184             delete words[position-1];
185         }
186         words[position-1] = new Word(in_word);
187     }
188 };
189
190         //-----------------------------------------------//
191         // Member function get_word().                   //
192         //-----------------------------------------------//
193
194 Word CompoundString::get_word(int position)
195 {
196 #ifdef DEBUG
197     cerr << "\t\t\t\t\tENTRY CompoundString::get_word" << endl;
198 #endif
```



```
199
200         //----------- Automatics - objects -------------//
201
202         //----------- Automatics - other ---------------//
203
204         //----------- Code begins ----------------------//
205
206         //-----------------------------------------------//
207         // De-reference and return the Word at the       //
208         // specified position.                           //
209         //-----------------------------------------------//
210
211     return( *(words[position-1]) );
212
213 };
214
215         //-----------------------------------------------//
216         // Member function get_string().                 //
217         //-----------------------------------------------//
218
219 void CompoundString::get_string(char * in_ptr)
220 {
221 #ifdef DEBUG
222     cerr << "\t\t\t\t\tENTRY CompoundString::get_string" << endl;
223 #endif
224
225         //----------- Automatics - objects -------------//
226
227         //----------- Automatics - other ---------------//
228
229     int CS_size;
230
231         //----------- Code begins ----------------------//
232
233         //-----------------------------------------------//
234         // Loop through each Word entry in the           //
235         // CompoundString, asking each to do its own     //
236         // get_string().  Concatenate all string         //
237         // fragments into the input string pointer,      //
238         // inserting spaces between each word.           //
239         //-----------------------------------------------//
240
241     CS_size = size;
242     in_ptr[0] = '\0';
243     for (int i=0; i<CS_size; i++)
244     {
245         if (i != 0)
246         {
247             strcat(in_ptr, " ");
248         }
249         strcat(in_ptr, (words[i])->get_string() );
250     }
251 }
252
253 //------------------------- Derived class -----------------------//
254
255         //-----------------------------------------------//
256         // Constructor 1 of 2.                           //
257         //-----------------------------------------------//
258
259 Phrase::Phrase(int in_size)
260 {
261 #ifdef DEBUG
262     cerr << "\t\t\t\t\tENTRY    Phrase CONSTRUCTOR 1" << endl;
263     cerr << "\t\t\t\t\tENTRY    (for autos - size specified)" << endl;
264 #endif
```

```
265
266             //-------------------------------------------------//
267             // Create an array of Word pointers of the         //
268             // specified size.  Initialize each entry to       //
269             // NULL.                                           //
270             //-------------------------------------------------//
271
272         size = in_size;
273         words = new Word * [size];
274         for [int i=0; i<size; i++]
275         [
276             words[i] = NULL;
277         ]
278     ];
279
280             //-------------------------------------------------//
281             // Constructor 2 of 2.                             //
282             //-------------------------------------------------//
283
284     Phrase::Phrase(const Phrase & in_phrase)
285     [
286     #ifdef DEBUG
287         cerr << "\t\t\t\t\tENTRY    Phrase CONSTRUCTOR 2" << endl;
288         cerr << "\t\t\t\t\tENTRY    (for copies)" << endl;
289     #endif
290
291             //-------------------------------------------------//
292             // This is a copy constructor.  Create a Word      //
293             // pointer array of the same size as the           //
294             // input Phrase.  Initialize each position         //
295             // from the input Phrase (creating copies of       //
296             // Words for those entries which are not NULL.)    //
297             //-------------------------------------------------//
298
299         size = in_phrase.size;
300
301         words = new Word * [size];
302         for (int i=0; i<size; i++)
303         {
304             if (in_phrase.words[i] == NULL)
305             {
306                 words[i] = NULL;
307             }
308             else
309             {
310                 words[i] = new Word(*(in_phrase.words[i]));
311             }
312         }
313
314     #ifdef DEBUG
315         cerr << "\t\t\tDEBUG Phrase Constructor done..." << endl;
316     #endif
317     };
318
319
320             //-------------------------------------------------//
321             // Destructor.                                     //
322             //-------------------------------------------------//
323
324     Phrase::~Phrase()
325     {
326     #ifdef DEBUG
327         cerr << "\t\t\t\t\tENTRY    Phrase DESTRUCTOR" << endl;
328     #endif
329
330     };
```



```
331
332
333             //-------------------------------------------------//
334             // Assignment operator.                            //
335             //-------------------------------------------------//
336
337     Phrase& Phrase::operator=[Phrase & in_phrase)
338     {
339     #ifdef DEBUG
340         cerr << "\t\t\t\t\tENTRY    Phrase::operator=" << endl;
341     #endif
342
343             //-------------------------------------------------//
344             // If assignment to itself, don't do anything,     //
345             // otherwise, loop through the Word array,         //
346             // deleting any words found.  Then delete          //
347             // the array itself.                               //
348             //-------------------------------------------------//
349
350         if (&in_phrase != this)
351         {
352             for (int i=0; i<size; i++)
353             {
354                 if (words[i] != NULL)
355                 {
356                     delete words[i];
357                 }
358             }
359             delete [] words;
360
361             //-------------------------------------------------//
362             // Create a new words array, of size to match      //
363             // the Phrase on the right-hand side               //
364             // of the assignment.  Loop through the right-     //
365             // hand Phrase, and create new word                //
366             // entries to match any found.                     //
367             //-------------------------------------------------//
368
369             size = in_phrase.size;
370             words = new Word * [size];
371             for (int j=0; j<size; j++)
372             {
373                 if (in_phrase.words[j] != NULL)
374                 {
375                     words[j] = new Word(*(in_phrase.words[j]));
376                 }
377             }
378         }
379         return *this;
380     };
381
382             //-------------------------------------------------//
383             // Member function read().                         //
384             //-------------------------------------------------//
385
386     void Phrase::read()
387     {
388     #ifdef DEBUG
389         cerr << "\t\t\t\t\tENTRY Phrase::read" << endl;
390     #endif
391
392             //----------- Automatics - objects -------------//
393
394         Word        new_words[MAX_PHRASE_LENGTH];   // Not obj, but "new"ed below.
395
396             //----------- Automatics - other ---------------//
```

```
397
398       boolean     end_of_line = FALSE;
399
400           //----------- Code begins ----------------------//
401
402           //------------------------------------------------//
403           // Since the entire Phrase will be read from     //
404           // the user's input - delete any Words which     //
405           // are already assigned.                         //
406           //------------------------------------------------//
407
408       for (int i=0; i<size; i++)
409       {
410           if (words[i] != NULL)
411           {
412                   delete words[i];
413           }
414       }
415
416           //------------------------------------------------//
417           // Eat any leading white space.                  //
418           //------------------------------------------------//
419
420       char ch;
421       boolean end_of_whitespace = FALSE;
422       while (!end_of_whitespace && cin.get(ch))
423       {
424           switch(ch)
425           {
426               case '\t':
427               case ' ':
428               case '\n':
429                           break;
430               default:
431                           cin.putback(ch);
432                           end_of_whitespace = TRUE;
433                           break;
434           }
435       }
436
437           //------------------------------------------------//
438           // Allow individual Words to be read until the   //
439           // end of the line is encountered.               //
440           //------------------------------------------------//
441
442       for (int j=0; end_of_line == FALSE; j++)
443       {
444           end_of_line = new_words(j).read();
445           words(j) = new Word(new_words[j]);
446       }
447
448       size = j;
449   };
450
451           //------------------------------------------------//
452           // Member function print().                      //
453           //------------------------------------------------//
454
455   void Phrase::print()
456   {
457   #ifdef DEBUG
458       cerr << "\t\t\t\t\tENTRY Phrase::print" << endl;
459   #endif
460
461           //----------- Automatics - objects -------------//
462
```



```
463           //----------- Automatics - other ---------------//
464
465       boolean ok = TRUE;
466
467           //----------- Code begins ----------------------//
468
469
470   #ifdef DEBUG
471       cerr << "\t\t\tDEBUG size of Phrase to be printed is ";
472       cerr << size << endl;
473   #endif
474
475           //------------------------------------------------//
476           // Loop through each Word entry to make sure     //
477           // that all entries have been assigned.          //
478           //------------------------------------------------//
479
480       for (int i=0; i< size; i++)
481       {
482           if (words(i) == NULL)
483           {
484   #ifdef DEBUG
485               cerr << "\t\t\tDEBUG    Word " << i+1 << " is NULL" << endl;
486   #endif
487               ok = FALSE;
488               break;
489           }
490           else
491           {
492   #ifdef DEBUG
493               cerr << "\t\t\tDEBUG    Word " << i+1 << " is not NULL" << endl;
494   #endif
495           }
496       }
497
498           //------------------------------------------------//
499           // If all entries are assigned, loop through     //
500           // them, allowing each word to print itself,     //
501           // and printing a space between each.            //
502           //------------------------------------------------//
503
504       if (ok)
505       {
506           for (int i=0; i<size; i++)
507           {
508               if (i != 0)
509               {
510                   cout << " ";
511               }
512               (*(words(i))).print();
513           }
514       }
515       else
516       {
517           cerr << "ERROR trying to print incomplete phrase" << endl;
518   #ifdef DEBUG
519           cout << "ERROR trying to print incomplete phrase" << endl;
520   #endif
521       }
522   };
523
524           //------------------------------------------------//
525           // Member function debug_print().                //
526           //------------------------------------------------//
527
528   void Phrase::debug_print()
```

```
{

        //----------- Automatics - objects -------------//

        //----------- Automatics - other ---------------//

    boolean ok = TRUE;

        //----------- Code begins ----------------------//

        //--------------------------------------------//
        // This function is essentially the same as   //
        // print(), but it uses Word::debug_print,    //
        // which includes diagnostic output.          //
        //--------------------------------------------//

    for (int i=0; i< size; i++)
    {
        if (words[i] == NULL)
        {
            ok = FALSE;
            break;
        }
    }

    if (ok)
    {
        for (int i=0; i<size; i++)
        {
            if (i != 0)
            {
                cerr << " ";
            }
            (*(words[i])).debug_print();
        }
    }
);

        //--------------------------------------------//
        // Member function get_size().                //
        //--------------------------------------------//

int Phrase::get_size()
{
#ifdef DEBUG
    cerr << "\t\t\t\t\tENTRY Phrase::get_size" << endl;
#endif

    return(size);
);


//------------------------ Derived class ----------------------//

        //--------------------------------------------//
        // Constructor 1 of 2.                        //
        //--------------------------------------------//

Pattern::Pattern(const Phrase & in_phrase)
{
#ifdef DEBUG
    cerr << "\t\t\t\t\tENTRY    Pattern CONSTRUCTOR 1" << endl;
    cerr << "\t\t\t\t\tENTRY    (for autos from Phrase)" << endl;
#endif

#ifdef DEBUG
```



```
    cerr << "\t\t\tDEBUG Phrase to construct pattern from is: " << endl;
    in_phrase.debug_print();
#endif

        //--------------------------------------------//
        // This is basically a copy constructor using //
        // a Phrase as input.  A words array is created //
        // of size to match the input phrase.  Entries //
        // are then initialized to match, with matching //
        // Words created for those entries which are  //
        // not NULL.                                  //
        //--------------------------------------------//

    size = in_phrase.size;

#ifdef DEBUG
    cerr << "\t\t\tDEBUG    size will be " << size << endl;
#endif

    words = new Word * [size];
    for (int i=0; i<size; i++)
    {
        if (in_phrase.words[i] == NULL)
        {
            words[i] = NULL;
        }
        else
        {
            words[i] = new Word(*(in_phrase.words[i]));
        }
    }

#ifdef DEBUG
    cerr << "\t\t\tDEBUG Pattern Constructor done..." << endl;
#endif
);


        //--------------------------------------------//
        // Constructor 2 of 2.                        //
        //--------------------------------------------//

Pattern::Pattern(const Pattern & in_pattern)
{
#ifdef DEBUG
    cerr << "\t\t\t\t\tENTRY    Pattern CONSTRUCTOR 2" << endl;
    cerr << "\t\t\t\t\tENTRY    (for copies)" << endl;
#endif

        //--------------------------------------------//
        // This is a true copy constructor - very     //
        // similar to the previous constructor, except //
        // that it copies from another Pattern.       //
        //--------------------------------------------//

    size = in_pattern.size;

    words = new Word * [size];
    for (int i=0; i<size; i++)
    {
        if (in_pattern.words[i] == NULL)
        {
            words[i] = NULL;
        }
        else
        {
```

```
661             words[i] = new Word(*(in_pattern.words[i]));
662         }
663     }
664
665 #ifdef DEBUG
666     cerr << "\t\t\tDEBUG Pattern Constructor done..." << endl;
667 #endif
668 };
669         //------------------------------------------------//
670         // Destructor.                                    //
671         //------------------------------------------------//
672
673 Pattern::~Pattern()
674 {
675 #ifdef DEBUG
676     cerr << "\t\t\t\t\tENTRY    Pattern DESTRUCTOR" << endl;
677 #endif
678
679 };
680
681
682         //------------------------------------------------//
683         // Assignment operator.                           //
684         //------------------------------------------------//
685
686 Pattern& Pattern::operator=(Pattern & in_pattern)
687 {
688 #ifdef DEBUG
689     cerr << "\t\t\t\t\tENTRY    Pattern::operator=" << endl;
690 #endif
691
692         //------------------------------------------------//
693         // If assignment to itself, don't do anything,    //
694         // otherwise, loop through the Word array,        //
695         // deleting any words found.  Then delete         //
696         // the array itself.                              //
697         //------------------------------------------------//
698
699     if (&in_pattern != this)
700     {
701         for (int i=0; i<size; i++)
702         {
703             if (words[i] != NULL)
704             {
705                 delete words[i];
706             }
707         }
708         delete [] words;
709
710         //------------------------------------------------//
711         // Create a new words array, of size to match     //
712         // the Phrase on the right-hand side              //
713         // of the assignment.  Loop through the right-    //
714         // hand Phrase, and create new word               //
715         // entries to match any found.                    //
716         //------------------------------------------------//
717
718         size = in_pattern.size;
719         words = new Word * [size];
720         for (int j=0; j<size; j++)
721         {
722             if (in_pattern.words[j] != NULL)
723             {
724                 words[j] = new Word(*(in_pattern.words[j]));
725             }
726         }
```



```
727     }
728     return *this;
729 };
730
731         //------------------------------------------------//
732         // Member function match().                       //
733         //------------------------------------------------//
734
735 boolean Pattern::match(Phrase in_phrase)
736 {
737 #ifdef DEBUG
738     cerr << "\t\t\t\t\tENTRY Pattern::match" << endl;
739 #endif
740
741         //----------- Automatics - objects -------------//
742
743         //----------- Automatics - other ---------------//
744
745     boolean     matched = FALSE;
746     char        pattern[MAX_PHRASE_STRING_LENGTH];
747     char        phrase[MAX_PHRASE_STRING_LENGTH];
748     int         phrase_size;
749     char        expbuf[EXPBUF_SIZE];
750     char        eof='\0';
751
752         //----------- Code begins ----------------------//
753
754         //------------------------------------------------//
755         // First create a text string which represents    //
756         // the input Phrase.                              //
757         //------------------------------------------------//
758
759     phrase_size = in_phrase.size;
760     phrase[0] = '\0';
761     for (int i=0; i<phrase_size; i++)
762     {
763         if (i != 0)
764         {
765             strcat(phrase, " ");
766         }
767         strcat(phrase, ((in_phrase.words)[i])->get_string() );
768     }
769 #ifdef DEBUG
770     cerr << "\t\t\tDEBUG phrase to match against pattern is " << endl;
771     cerr << "\t\t\tDEBUG        " << phrase << endl;
772 #endif
773
774         //------------------------------------------------//
775         // Next, create the text string which             //
776         // represents this Pattern.                       //
777         //------------------------------------------------//
778
779     phrase_size = size;
780     pattern[0] = '\0';
781     for (int j=0; j<phrase_size; j++)
782     {
783         if (j != 0)
784         {
785             strcat(pattern, " ");
786         }
787         strcat(pattern, (words[j])->get_string() );
788     }
789 #ifdef DEBUG
790     cerr << "\t\t\tDEBUG the pattern to match against is " << endl;
791     cerr << "\t\t\tDEBUG        " << pattern << endl;
792 #endif
```

```
793
794             //-------------------------------------------------//
795             // Use the regexp() library capabilities to        //
796             // compile the pattern and attempt the match.      //
797             //-------------------------------------------------//
798
799         compile(pattern, expbuf, expbuf+EXPBUF_SIZE, eof);
800
801         if (step(phrase, expbuf))
802         {
803             matched = TRUE;
804     #ifdef DEBUG
805             cerr << "\t\t\tDEBUG match returning TRUE" << endl;
806     #endif
807         )
808         else
809         {
810             matched = FALSE;
811     #ifdef DEBUG
812             cerr << "\t\t\tDEBUG match returning FALSE" << endl;
813     #endif
814         )
815
816         return(matched);
817     )
```

```
1     //-------------------------------------------------------------------------
2     //
3     //  Filename:   data_PC_defs.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file contains no executable code.  It contains various
8     //              regular expression pattern fragments used by various phrase_
9     //              modifiers.
10    //
11    //  Notes:      The theory and design of this system have been documented
12    //              in the accompanying thesis report.
13    //
14    //  Revisions   By              Reason
15    //  ---------   --              ------
16    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
17    //
18    //-------------------------------------------------------------------------
19
20            //----------- Included files -------------------//
21
22    #include "stdtype.h"
23
24            //-----------------------------------------------//
25            // Regular expression pattern fragments.  This  //
26            // one will match any single word.              //
27            //-----------------------------------------------//
28
29    pattern_fragment any_word = "\\<[a-zA-Z][a-zA-Z]*\\>";
```

```
1   //--------------------------------------------------------------------------
2   //
3   //  Filename:   data_dictionary.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file contains no executable code.  Here, all the words
8   //              recognized by the system's dictionary are assembled (part of
9   //              the system's knowledge base).
10  //
11  //  Notes:      The theory and design of this system have been documented
12  //              in the accompanying thesis report.
13  //
14  //  Revisions   By              Reason
15  //  ---------   --              ------
16  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
17  //
18  //--------------------------------------------------------------------------
19
20          //----------- Included files -------------------//
21
22  #include <iostream.h>
23  #include "stdtype.h"
24  #include "dictionaryentry.h"
25  #include "dictionary.h"
26  #include "word.h"
27
28          //------------------------------------------------//
29          // Lists of all words, by type.                   //
30          //------------------------------------------------//
31
32          //------------------------------------------------//
33          // Nouns.  For reference, the initial arguments   //
34          // are the word index, knowledge level, gender,   //
35          // German singular, German plural, English        //
36          // singular and English plural.                   //
37          //------------------------------------------------//
38
39  DictionaryEntry_noun noun_list[] =
40      {
41          DictionaryEntry_noun(1,1,G_masculine,"Mann","Maenner","man","men"),
42          DictionaryEntry_noun(2,1,G_masculine,"Hund","Hunds?","dog","dogs"),
43          DictionaryEntry_noun(3,4,G_masculine,"Junge","Junges?","boy","boys"),
44          DictionaryEntry_noun(4,5,G_feminine,"Maedchen","Maedchens?","girl","girl
    s"),
45          DictionaryEntry_noun(5,6,G_masculine,"Tisch","Tische","table","tables"),
46          DictionaryEntry_noun(6,1,G_feminine,"Mutter","Muetter","mother","mothers
    "),
47          DictionaryEntry_noun(7,2,G_neuter,"Brot","Brote","bread","breads"),
48          DictionaryEntry_noun(8,1,G_neuter,"Gras","Graesser","grass","grasses"),
49          DictionaryEntry_noun(9,1,G_masculine,"Ball","Baelle","ball","balls"),
50          DictionaryEntry_noun(10,1,G_masculine,"Park","Parke","park","parks"),
51          DictionaryEntry_noun(11,1,G_feminine,"Hand","Haende","hand","hands"),
52          DictionaryEntry_noun(12,1,G_masculine,"Hut","Huete","hat","hats"),
53          DictionaryEntry_noun(13,1,G_neuter,"Haus","Haueser","house","houses"),
54          DictionaryEntry_noun(14,1,G_masculine,"Wind","Winde","wind","winds"),
55          DictionaryEntry_noun(15,1,G_masculine,"Sohn","Soehne","son","sons"),
56          DictionaryEntry_noun(16,1,G_masculine,"Garten","Gaerten","garden","garde
    ns"),
57          DictionaryEntry_noun(17,1,G_masculine,"Sommer","Sommer","summer","summer
    s"),
58          DictionaryEntry_noun(18,1,G_masculine,"Winter","Winter","winter","winter
    s"),
59          DictionaryEntry_noun(19,1,G_feminine,"Butter","","butter","butters"),
60          DictionaryEntry_noun(20,1,G_masculine,"Vater","Vaeter","father","fathers
    "),
```



```
61          DictionaryEntry_noun(21,1,G_masculine,"Onkel","Onkel","uncle","uncles"),
62          DictionaryEntry_noun(22,1,G_feminine,"Schule","Schule","school","schools
    "),
63          DictionaryEntry_noun(23,1,G_masculine,"Finger","Finger","finger","finger
    s"),
64          DictionaryEntry_noun(24,1,G_neuter,"Papier","Papiere","paper","papers"),
65          DictionaryEntry_noun(25,1,G_feminine,"Frau","Frauen","woman","women"),
66          DictionaryEntry_noun(26,1,G_neuter,"Kind","Kinder","child","children"),
67          DictionaryEntry_noun(27,2,G_neuter,"Plan","Plaene","plan","plans"),
68          DictionaryEntry_noun(28,2,G_neuter,"Glas","Glaesser","glass","glasses"),
69          DictionaryEntry_noun(29,2,G_neuter,"Jahr","Jahre","year","years"),
70          DictionaryEntry_noun(30,2,G_masculine,"Tee","","tea","teas"),
71          DictionaryEntry_noun(31,2,G_feminine,"See","","sea","seas"),
72          DictionaryEntry_noun(32,2,G_neuter,"Bett","Betten","bed","beds"),
73          DictionaryEntry_noun(33,2,G_feminine,"Welt","Welten","world","worlds"),
74          DictionaryEntry_noun(34,2,G_neuter,"Wasser","Wasser","water","waters"),
75          DictionaryEntry_noun(35,2,G_masculine,"Lehrer","Lehrer","teacher","teach
    ers"),
76          DictionaryEntry_noun(36,2,G_feminine,"Klasse","Klassen","class","classes
    "),
77          DictionaryEntry_noun(37,2,G_neuter,"Bier","Biere","beer","beers"),
78          DictionaryEntry_noun(38,2,G_feminine,"Rosen","Rosen","roses","roses"),
79          DictionaryEntry_noun(39,2,G_masculine,"Monat","Monate","month","months")
    ,
80          DictionaryEntry_noun(40,2,G_masculine,"Stock","Stoecke","stick","sticks"
    ),
81          DictionaryEntry_noun(41,2,G_masculine,"Fuss","Fuesse","foot","feet"),
82          DictionaryEntry_noun(42,2,G_masculine,"Bruder","Brueder","brother","brot
    hers"),
83          DictionaryEntry_noun(43,2,G_masculine,"Stuhl","Stuehle","chair","chairs"
    ),
84          DictionaryEntry_noun(44,2,G_feminine,"Suppe","Suppen","soup","soups"),
85          DictionaryEntry_noun(45,2,G_masculine,"Sport","Sporte","sport","sports")
    ,
86          DictionaryEntry_noun(46,2,G_feminine,"Schwester","Schwestern","sister","
    sisters"),
87          DictionaryEntry_noun(47,2,G_masculine,"Kaffee","","coffee","coffees"),
88          DictionaryEntry_noun(48,2,G_feminine,"Tante","Tanten","aunt","aunts"),
89          DictionaryEntry_noun(49,2,G_masculine,"Schuh","Schuhe","shoe","shoes"),
90          DictionaryEntry_noun(50,2,G_masculine,"Doktor","Doktoren","doctor","doct
    ors"),
91          DictionaryEntry_noun(51,3,G_neuter,"Deutsch","","German",""),
92          DictionaryEntry_noun(52,5,G_masculine,"Kaufmann","Kaufleute","merchant",
    "merchants")
93      };
94
95          //------------------------------------------------//
96          // Articles.  For reference, the                  //
97          // initial arguments are the word index, the      //
98          // knowledge level, the German masculine          //
99          // singular, the German feminine singular, the    //
100         // German neuter singular, the German plural,     //
101         // and the English.                               //
102         //------------------------------------------------//
103
104 DictionaryEntry_article article_list[] =
105     {
106         DictionaryEntry_article(100001,1,"der","die","das","die","the"),
107         DictionaryEntry_article(100002,1,"ein","eine","ein","","a")
108     };
109
110         //------------------------------------------------//
111         // Adjectives.  For reference, the initial        //
112         // arguments are the word index, the knowledge    //
113         // level, the English and the German.             //
114         //------------------------------------------------//
```

```
115
116  DictionaryEntry_adjective adj_list[] =
117      {
118          DictionaryEntry_adjective(200001,1,"good","gut"),
119          DictionaryEntry_adjective(200002,1,"warm","warm"),
120          DictionaryEntry_adjective(200003,4,"young","jung"),
121          DictionaryEntry_adjective(200004,1,"round","rund"),
122          DictionaryEntry_adjective(200005,1,"cold","kalt"),
123          DictionaryEntry_adjective(200006,1,"old","alt"),
124          DictionaryEntry_adjective(200007,1,"new","neu"),
125          DictionaryEntry_adjective(200008,1,"blue","blau"),
126          DictionaryEntry_adjective(200009,1,"brown","braun"),
127          DictionaryEntry_adjective(200010,1,"long","lang"),
128          DictionaryEntry_adjective(200011,1,"here","hier"),
129          DictionaryEntry_adjective(200012,1,"full","voll"),
130          DictionaryEntry_adjective(200013,2,"clear","klar"),
131          DictionaryEntry_adjective(200014,2,"fresh","frisch"),
132          DictionaryEntry_adjective(200015,2,"red","rot"),
133          DictionaryEntry_adjective(200016,2,"tall","gross"),
134          DictionaryEntry_adjective(200017,3,"there","dort")
135      };
136
137          //-----------------------------------------------//
138          // Verbs.  For reference, the initial arguments //
139          // are the word index, the knowledge level,     //
140          // and the English and German present tense.    //
141          //-----------------------------------------------//
142
143  DictionaryEntry_verb verb_list[] =
144      {
145          DictionaryEntry_verb(300001,2,"is","ist"),
146          DictionaryEntry_verb(300002,1,"sings","singt"),
147          DictionaryEntry_verb(300003,1,"sends","sendet"),
148          DictionaryEntry_verb(300004,1,"finds","findet"),
149          DictionaryEntry_verb(300005,1,"falls","faellt"),
150          DictionaryEntry_verb(300006,1,"springs","springt"),
151          DictionaryEntry_verb(300007,1,"binds","bindet"),
152          DictionaryEntry_verb(300008,1,"brings","bringt"),
153          DictionaryEntry_verb(300009,1,"begins","beginnt"),
154          DictionaryEntry_verb(300010,1,"sees","sieht"),
155          DictionaryEntry_verb(300011,1,"washes","waescht"),
156          DictionaryEntry_verb(300012,1,"helps","hilft"),
157          DictionaryEntry_verb(300013,1,"has","hat"),
158          DictionaryEntry_verb(300014,1,"comes","kommt"),
159          DictionaryEntry_verb(300015,1,"drinks","trinkt"),
160          DictionaryEntry_verb(300015,2,"goes","geht") ,
161          DictionaryEntry_verb(300016,2,"learns","lernt"),
162          DictionaryEntry_verb(300017,2,"plays","spielt"),
163          DictionaryEntry_verb(300018,2,"stays","steht"),
164          DictionaryEntry_verb(300019,2,"rolls","rollt")
165      );
166
167          //-----------------------------------------------//
168          // The whole dictionary.                         //
169          //-----------------------------------------------//
170
171  Dictionary dictionary(noun_list, article_list, adj_list, verb_list);
```

```
1     //-------------------------------------------------------------------------
2     //
3     //  Filename:   data_phrasemodellist.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file contains no executable code.  Here, all the phrase
8     //              models for the phrase model list are assembled (part of the
9     //              system's knowledge base).
10    //
11    //  Notes:      The theory and design of this system have been documented
12    //              in the accompanying thesis report.
13    //
14    //              Even with the current limited comments, this file is
15    //              quickly becoming unreadable (e.g. see the PatternNode
16    //              definitions below!).  It would be desirable to devise a
17    //              more readable format, or even better, to provide tools
18    //              which allow for viewing and modifying the data structures
19    //              in a friendlier way.
20    //
21    //  Revisions   By              Reason
22    //  ---------   --              ------
23    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
24    //
25    //-------------------------------------------------------------------------
26
27          //----------- Included files -------------------//
28
29    #include <iostream.h>
30    #include "dictionary.h"
31    #include "stdtype.h"
32    #include "basephrase.h"
33    #include "basephraselist.h"
34    #include "basephrasepool.h"
35    #include "phrasemodel.h"
36    #include "phrasemodellist.h"
37    #include "worddescriptor.h"
38    #include "phrasedescriptor.h"
39
40          //----------- External data --------------------//
41
42    extern  Dictionary dictionary;
43
44          //----------- External functions ---------------//
45
46    extern  Pattern PC_gender1_first_word(Phrase &);
47    extern  Pattern PC_gender2_first_word(Phrase &);
48    extern  Pattern PC_nouncap_second_word(Phrase &);
49    extern  Pattern PC_wrong_first_and_second_word(Phrase &);
50    extern  Pattern PC_wrong_first_word(Phrase &);
51    extern  Pattern PC_wrong_fourth_word(Phrase &);
52    extern  Pattern PC_wrong_second_word(Phrase &);
53    extern  Pattern PC_wrong_third_word(Phrase &);
54    extern  Pattern PC_extra_article_fourth(Phrase &);
55
56    extern  BasePhrase BPT_no_translation(BasePhrase &);
57    extern  BasePhrase BPT_drop_fourth_word(BasePhrase &);
58
59          //-----------------------------------------------//
60          // The PhraseModelList must be built from the    //
61          // bottom up.  First the base phrases, then      //
62          // the lists, then the pools, etc.               //
63          //-----------------------------------------------//
64
65          //----------- The Base Phrases -----------------//
66
```



```
67
68    #define BP_0001_SIZE    4
69    #define BPL_0001_SIZE   12
70    BasePhrase bp_0001[] =
71        {
72            // the man is good
73            BasePhrase(BP_0001_SIZE,100001, 1,300001,200001),
74            // the bread is warm
75            BasePhrase(BP_0001_SIZE,100001, 7,300001,200002),
76            // the beer is cold
77            BasePhrase(BP_0001_SIZE,100001,37,300001,200005),
78            // the house is old
79            BasePhrase(BP_0001_SIZE,100001,13,300001,200006),
80            // the plan is new
81            BasePhrase(BP_0001_SIZE,100001,27,300001,200007),
82            // the water is blue
83            BasePhrase(BP_0001_SIZE,100001,34,300001,200008),
84            // the hat is brown
85            BasePhrase(BP_0001_SIZE,100001,12,300001,200009),
86            // the grass is long
87            BasePhrase(BP_0001_SIZE,100001, 8,300001,200010),
88            // the glass is full
89            BasePhrase(BP_0001_SIZE,100001,28,300001,200012),
90            // the sea is clear
91            BasePhrase(BP_0001_SIZE,100001,31,300001,200013),
92            // the chair is red
93            BasePhrase(BP_0001_SIZE,100001,43,300001,200015),
94            // the woman is tall
95            BasePhrase(BP_0001_SIZE,100001,25,300001,200016)
96        };
97
98    #define BP_0002_SIZE    4
99    #define BPL_0002_SIZE   2
100   BasePhrase bp_0002[] =
101       {
102           // the mother is young
103           BasePhrase(BP_0002_SIZE,100001,6,300001,200003),
104           // the table is round
105           BasePhrase(BP_0002_SIZE,100001,5,300001,200004)
106       };
107
108   #define BP_0003_SIZE    4
109   #define BPL_0003_SIZE   4
110   BasePhrase bp_0003() =
111       {
112           // the girl learns German
113           BasePhrase(BP_0003_SIZE,100001, 4,300016,51),
114           // the doctor drinks coffee
115           BasePhrase(BP_0003_SIZE,100001,50,300015,47),
116           // the garden has roses
117           BasePhrase(BP_0003_SIZE,100001,16,300013,38),
118           // the teacher brings paper
119           BasePhrase(BP_0003_SIZE,100001,35,300008,24)
120       };
121
122
123   #define BP_0004_SIZE    5
124   #define BPL_0004_SIZE   1
125   BasePhrase bp_0004() =
126       {
127           // the man is a merchant
128           BasePhrase(BP_0004_SIZE,100001,1,300001,100002,52)
129       };
130
131
132           //----------- The Base Phrase Lists ------------//
```

```
133
134   BasePhraseList bpl_0001( BPL_0001_SIZE, bp_0001 );
135   BasePhraseList bpl_0002( BPL_0002_SIZE, bp_0002 );
136   BasePhraseList bpl_0003( BPL_0003_SIZE, bp_0003 );
137   BasePhraseList bpl_0004( BPL_0004_SIZE, bp_0004 );
138
139           //------------ The Base Phrase Pools ------------//
140
141   #define BPP_0001_SIZE   1
142   BasePhrasePool bpp_0001( BPP_0001_SIZE,
143                            &bpl_0001
144                          );
145
146   #define BPP_0002_SIZE   2
147   BasePhrasePool bpp_0002( BPP_0002_SIZE,
148                            &bpl_0001,
149                            &bpl_0002
150                          );
151
152   #define BPP_0003_SIZE   1
153   BasePhrasePool bpp_0003( BPP_0003_SIZE,
154                            &bpl_0003
155                          );
156
157   #define BPP_0004_SIZE   1
158   BasePhrasePool bpp_0004( BPP_0004_SIZE,
159                            &bpl_0004
160                          );
161
162           //------------ The Word Descriptors -------------//
163
164   WordDescriptor wd_0001(C_singular, G_DYNAMIC, L_english, article);
165   WordDescriptor wd_0002(C_singular, G_masculine, L_english, noun);
166   WordDescriptor wd_0003(C_singular, G_DYNAMIC, L_german, article);
167   WordDescriptor wd_0004(C_singular, G_masculine, L_german, noun);
168
169   WordDescriptor wd_0005(C_none, G_none, L_english, adjective);
170   WordDescriptor wd_0006(C_none, G_none, L_english, verb);
171   WordDescriptor wd_0007(C_none, G_none, L_german, adjective);
172   WordDescriptor wd_0008(C_none, G_none, L_german, verb);
173
174           //------------ The Phrase Descriptors -----------//
175
176   #define PD_SIZE_0001    4
177   int pd_0001_order[] = { 2, 1, 3, 4 };
178   PhraseDescriptor pd_0001(PD_SIZE_0001, pd_0001_order, &wd_0001, &wd_0002,
179                                                         &wd_0006, &wd_0005);
180   PhraseDescriptor pd_0002(PD_SIZE_0001, pd_0001_order, &wd_0003, &wd_0004,
181                                                         &wd_0008, &wd_0007);
182
183   #define PD_SIZE_0002    4
184   int pd_0002_order[] = { 2, 1, 3, 4 };
185   PhraseDescriptor pd_0003(PD_SIZE_0002, pd_0002_order, &wd_0001, &wd_0002,
186                                                         &wd_0006, &wd_0002);
187   PhraseDescriptor pd_0004(PD_SIZE_0002, pd_0002_order, &wd_0003, &wd_0004,
188                                                         &wd_0008, &wd_0004);
189
190   #define PD_SIZE_0003    5
191   #define PD_SIZE_0004    4
192   int pd_0003_order[] = { 2, 1, 3, 5, 4 };
193   int pd_0004_order[] = { 2, 1, 3, 4 };
194   PhraseDescriptor pd_0005(PD_SIZE_0003, pd_0003_order, &wd_0001, &wd_0002,
195                                                         &wd_0006, &wd_0001, &wd_0002);
196   PhraseDescriptor pd_0006(PD_SIZE_0004, pd_0004_order, &wd_0003, &wd_0004,
197                                                         &wd_0008, &wd_0004);
198
```



```
199           //------------ The Patterns --------------------//
200
201   Hint hint_0001("The first word is not what was expected.");
202   Hint hint_0002("The second word is not what was expected.");
203   Hint hint_0003("The third word is not what was expected.");
204   Hint hint_0004("The fourth word is not what was expected.");
205   Hint hint_0005("The first and second words are not what were expected.");
206   Hint hint_0006("Nouns in German should be capitalized.");
207   Hint hint_0007("The gender of the article is not correct.");
208   Hint hint_0008("The gender of the article is not correct.");
209   Hint hint_0019("An article should not be used in this case.");
210
211   PatternNode pat_0008(&PC_gender2_first_word, NULL, NULL, &hint_0008);
212   PatternNode pat_0007(&PC_gender1_first_word, NULL, &pat_0008, &hint_0007);
213   PatternNode pat_0006(&PC_nouncap_second_word, NULL, NULL, &hint_0006);
214   PatternNode pat_0005(&PC_wrong_first_and_second_word, NULL, NULL, &hint_0005);
215   PatternNode pat_0004(&PC_wrong_fourth_word, NULL, &pat_0005, &hint_0004);
216   PatternNode pat_0003(&PC_wrong_third_word, NULL, &pat_0004, &hint_0003);
217   PatternNode pat_0002(&PC_wrong_second_word, &pat_0006, &pat_0003, &hint_0002);
218   PatternNode pat_0001(&PC_wrong_first_word, &pat_0007, &pat_0002, &hint_0001);
219
220   PatternNode pat_0019(&PC_extra_article_fourth, NULL, NULL, &hint_0019);
221   PatternNode pat_0018(&PC_gender2_first_word, NULL, NULL, &hint_0008);
222   PatternNode pat_0017(&PC_gender1_first_word, NULL, &pat_0018, &hint_0007);
223   PatternNode pat_0016(&PC_nouncap_second_word, NULL, NULL, &hint_0006);
224   PatternNode pat_0015(&PC_wrong_first_and_second_word, NULL, &pat_0019, &hint_000
      5);
225   PatternNode pat_0014(&PC_wrong_fourth_word, NULL, &pat_0015, &hint_0004);
226   PatternNode pat_0013(&PC_wrong_third_word, NULL, &pat_0014, &hint_0003);
227   PatternNode pat_0012(&PC_wrong_second_word, &pat_0016, &pat_0013, &hint_0002);
228   PatternNode pat_0011(&PC_wrong_first_word, &pat_0017, &pat_0012, &hint_0001);
229
230   //Hint hint_0001("The gender of the first word is wrong.");
231   //PatternNode pat_0001(&PC_gender1_first_word, NULL, NULL, &hint_0001);
232   //Hint hint_0002("There is something wrong with the second word in your response
      .");
233   //PatternNode pat_0002(&PC_wrong_second_word, NULL, NULL, &hint_0002);
234   //Hint hint_0003("There is something wrong with the first word in your response.
      ");
235   //PatternNode pat_0003(&PC_wrong_first_word, &pat_0001, &pat_0002, &hint_0003);
236
237           //------------ The Translation Trees -----------//
238
239   TranslationTree tt_0001(BPT_no_translation, &pat_0001, &pd_0002);
240   TranslationTree tt_0002(BPT_no_translation, &pat_0001, &pd_0004);
241   TranslationTree tt_0003(BPT_drop_fourth_word, &pat_0011, &pd_0006);
242
243           //------------ The Phrase Models ---------------//
244
245   #define PML_0001_SIZE 4
246   PhraseModel  pml_0001[] =
247       {
248           PhraseModel(7, bpp_0001,pd_0001,tt_0001),
249           PhraseModel(1, bpp_0002,pd_0001,tt_0001),
250           PhraseModel(3, bpp_0003,pd_0003,tt_0002),
251           PhraseModel(5, bpp_0004,pd_0005,tt_0003)
252       };
253
254
255   PhraseModelList  models(PML_0001_SIZE,pml_0001);
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   dictionary.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the Dictionary class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include "dictionary.h"
22  #include "dictionaryentry.h"
23  #include "dictionary_defs.h"
24
25          //-----------------------------------------------//
26          // Constructor.                                  //
27          //-----------------------------------------------//
28
29  Dictionary::Dictionary(DictionaryEntry_noun * noun_list,
30                  DictionaryEntry_article * article_list,
31                  DictionaryEntry_adjective * adj_list,
32                  DictionaryEntry_verb * verb_list)
33  {
34  #ifdef DEBUG
35      cerr << "\t\t\t\t\tENTRY    Dictionary CONSTRUCTOR" << endl;
36  #endif
37
38      nouns = noun_list;
39      articles = article_list;
40      adjs = adj_list;
41      verbs = verb_list;
42  };
43
44          //-----------------------------------------------//
45          // Destructor.                                   //
46          //-----------------------------------------------//
47
48  Dictionary::~Dictionary()
49  {
50  #ifdef DEBUG
51      cerr << "\t\t\t\t\tENTRY    Dictionary DESTRUCTOR" << endl;
52  #endif
53  };
54
55          //-----------------------------------------------//
56          // Member function get_word().                   //
57          //-----------------------------------------------//
58
59  Word Dictionary::get_word(int word_index, WordDescriptor * word_desc)
60  {
61  #ifdef DEBUG
62      cerr << "\t\t\t\t\tENTRY Dictionary::get_word" << endl;
63  #endif
64
65          //----------- Automatics - objects -------------//
66
```



```
67          //----------- Automatics - other ---------------//
68
69      WD_Type  word_type;
70
71          //----------- Code begins ----------------------//
72
73          //-----------------------------------------------//
74          // Determine the indicated word type, in order  //
75          // to search the appropriate DictionaryEntry    //
76          // list.                                        //
77          //-----------------------------------------------//
78
79      word_type = word_desc->type;
80
81      switch(word_type)
82      {
83          case article:
84              int temp;
85  #ifdef DEBUG
86              cerr << "\t\t\tDEBUG Dictionary::get_word looking for article" << en
    dl;
87  #endif
88          //-----------------------------------------------//
89          // Search article list for word index           //
90          // match, then ask DictionaryEntry for the      //
91          // necessary derivation.                        //
92          //-----------------------------------------------//
93
94              for(DictionaryEntry_article * ptr1 = articles;
95                                  temp = ptr1->get_index() != word_index;ptr1++)
96              {
97  #ifdef DEBUG
98                  cerr << "\t\t\tDEBUG Dictionary::get_word loops and finds index
    ";
99                  cerr << temp;
100                 cerr << " (looking for ";
101                 cerr << word_index;
102                 cerr << ")" << endl;
103 #endif
104             }
105             return ptr1->derive_word(word_desc);
106         case noun:
107 #ifdef DEBUG
108             cerr << "\t\t\tDEBUG Dictionary::get_word looking for noun" << endl;
109 #endif
110         //-----------------------------------------------//
111         // Search noun list for word index              //
112         // match, then ask DictionaryEntry for the      //
113         // necessary derivation.                        //
114         //-----------------------------------------------//
115
116             for(DictionaryEntry_noun * ptr2 = nouns;
117                                 ptr2->get_index() != word_index;ptr2++)
118             {
119 #ifdef DEBUG
120                 cerr << "\t\t\tDEBUG Dictionary::get_word loops and finds index
    ";
121                 cerr << ptr2->get_index();
122                 cerr << " (looking for ";
123                 cerr << word_index;
124                 cerr << ")" << endl;
125 #endif
126             }
127             return ptr2->derive_word(word_desc);
128         case verb:
129 #ifdef DEBUG
```

```
130                 cerr << "\t\t\tDEBUG Dictionary::get_word looking for verb" << endl;
131     #endif
132             //-----------------------------------------------//
133             // Search verb list for word index               //
134             // match, then ask DictionaryEntry for the       //
135             // necessary derivation.                         //
136             //-----------------------------------------------//
137
138                 for(DictionaryEntry_verb * ptr3 = verbs;
139                                 ptr3->get_index() != word_index;ptr3++)
140                 {
141     #ifdef DEBUG
142                     cerr << "\t\t\tDEBUG Dictionary::get_word loops and finds index
    ";
143                     cerr << ptr3->get_index();
144                     cerr << " (looking for ";
145                     cerr << word_index;
146                     cerr << ")" << endl;
147     #endif
148                 }
149                 return ptr3->derive_word(word_desc);
150             case adjective:
151     #ifdef DEBUG
152                 cerr << "\t\t\tDEBUG Dictionary::get_word looking for adjective" <<
    endl;
153     #endif
154             //-----------------------------------------------//
155             // Search adjective list for word index          //
156             // match, then ask DictionaryEntry for the       //
157             // necessary derivation.                         //
158             //-----------------------------------------------//
159
160                 for(DictionaryEntry_adjective * ptr4 = adjs;
161                                 ptr4->get_index() != word_index;ptr4++)
162                 {
163     #ifdef DEBUG
164                     cerr << "\t\t\tDEBUG Dictionary::get_word loops and finds index
    ";
165                     cerr << ptr4->get_index();
166                     cerr << " (looking for ";
167                     cerr << word_index;
168                     cerr << ")" << endl;
169     #endif
170                 }
171                 return ptr4->derive_word(word_desc);
172             case WD_UNKNOWN_TYPE:
173                 cerr << "\t\t\tERROR Unknown word type passed to dict" << endl;
174     #ifdef DEBUG
175                 cout << "\t\t\tERROR Unknown word type passed to dict" << endl;
176     #endif
177             default:
178                 break;
179         }
180     }
181
182             //-----------------------------------------------//
183             // Member function get_level().                  //
184             //-----------------------------------------------//
185
186     int Dictionary::get_level(int word_index)
187     {
188     #ifdef DEBUG
189         cerr << "\t\t\t\t\tENTRY Dictionary::get_level" << endl;
190
191         cerr << "\t\t\tDEBUG input word index is " << word_index << endl;
192     #endif
```



```
193
194             //----------- Automatics - objects -------------//
195
196             //----------- Automatics - other ---------------//
197
198         boolean  not_found=TRUE;
199
200             //----------- Code begins ----------------------//
201
202             //-----------------------------------------------//
203             // This member function is used when             //
204             // determining what vocabulary and models are    //
205             // appropriate for the current user's knowledge  //
206             // level.  The input word index is range         //
207             // checked to determine the word type, and       //
208             // the DictionaryEntry is asked for its          //
209             // knowledge level.                              //
210             //-----------------------------------------------//
211
212         if (word_index >= DE_MIN_NOUN_INDEX && word_index <= DE_MAX_NOUN_INDEX)
213         {
214             for(DictionaryEntry_noun * ptr2 = nouns;
215                                 ptr2->get_index() != word_index; ptr2++)
216             {
217             }
218             return ptr2->get_level();
219         }
220         else if (word_index >= DE_MIN_DIRART_INDEX && word_index <= DE_MAX_DIRART_IN
    DEX)
221         {
222             for(DictionaryEntry_article * ptr1 = articles;
223                                 ptr1->get_index() != word_index; ptr1++)
224             {
225             }
226             return ptr1->get_level();
227         }
228         else if (word_index >= DE_MIN_VERB_INDEX && word_index <= DE_MAX_VERB_INDEX)
229         {
230             for(DictionaryEntry_verb * ptr3 = verbs;
231                                 ptr3->get_index() != word_index; ptr3++)
232             {
233             }
234             return ptr3->get_level();
235         }
236         else if (word_index >= DE_MIN_ADJ_INDEX && word_index <= DE_MAX_ADJ_INDEX)
237         {
238             for(DictionaryEntry_adjective * ptr4 = adjs;
239                                 ptr4->get_index() != word_index; ptr4++)
240             {
241             }
242             return ptr4->get_level();
243         }
244         else
245         {
246             cerr << "ERROR - Invalid word index" << endl;
247     #ifdef DEBUG
248             cout << "ERROR - Invalid word index" << endl;
249     #endif
250         }
251     }
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   dictionaryentry.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the DictionaryEntry class and its derived
8   //              classes.
9   //
10  //  Notes:      The theory and design of this system have been documented
11  //              in the accompanying thesis report.
12  //
13  //  Revisions   By              Reason
14  //  ---------   --              ------
15  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16  //
17  //-------------------------------------------------------------------------------
18
19          //----------- Included files -------------------//
20
21  #include <iostream.h>
22  #include "dictionaryentry.h"
23
24  //------------------------- Base class --------------------------//
25
26          //-----------------------------------------------//
27          // Constructor.                                  //
28          //-----------------------------------------------//
29
30  DictionaryEntry::DictionaryEntry(int init_index, int init_level)
31  {
32  #ifdef DEBUG
33      cerr << "\t\t\t\t\tENTRY    DictionaryEntry CONSTRUCTOR" << endl;
34  #endif
35
36      index = init_index;
37      knowledge_level = init_level;
38  };
39
40          //-----------------------------------------------//
41          // Destructor.                                   //
42          //-----------------------------------------------//
43
44  DictionaryEntry::~DictionaryEntry()
45  {
46  #ifdef DEBUG
47      cerr << "\t\t\t\t\tENTRY    DictionaryEntry DESTRUCTOR" << endl;
48  #endif
49  };
50
51          //-----------------------------------------------//
52          // Member function get_index().                  //
53          //-----------------------------------------------//
54
55  int DictionaryEntry::get_index()
56  {
57  #ifdef DEBUG
58      cerr << "\t\t\t\t\tENTRY DictionaryEntry::get_index()" << endl;
59
60      cerr << "\t\t\tDEBUG Returning index " << index << endl;
61  #endif
62
63          //----------- Automatics - objects -------------//
64
65          //----------- Automatics - other ---------------//
66
```



```
67          //----------- Code begins ----------------------//
68
69      return index;
70  }
71
72          //-----------------------------------------------//
73          // Member function get_level().                  //
74          //-----------------------------------------------//
75
76  int DictionaryEntry::get_level()
77  {
78  #ifdef DEBUG
79      cerr << "\t\t\t\t\tENTRY DictionaryEntry::get_level()" << endl;
80  #endif
81
82          //----------- Automatics - objects -------------//
83
84          //----------- Automatics - other ---------------//
85
86          //----------- Code begins ----------------------//
87
88      return knowledge_level;
89  }
90
91
92  //------------------------- Derived class ----------------------//
93
94          //-----------------------------------------------//
95          // Constructor.                                  //
96          //-----------------------------------------------//
97
98  DictionaryEntry_article::DictionaryEntry_article(int init_index,
99                          int init_know_level,
100                         char * init_masc,
101                         char * init_fem, char * init_neut, char * init_plural,
102                         char * init_eng) : DictionaryEntry(init_index,
103                                                 init_know_level)
104 {
105 #ifdef DEBUG
106     cerr << "\t\t\t\t\tENTRY    DictionaryEntry_article CONSTRUCTOR" << endl;
107 #endif
108
109     masc = init_masc;
110     fem = init_fem;
111     neut = init_neut;
112     plural = init_plural;
113     eng = init_eng;
114 };
115
116         //-----------------------------------------------//
117         // Destructor.                                   //
118         //-----------------------------------------------//
119
120 DictionaryEntry_article::~DictionaryEntry_article()
121 {
122 #ifdef DEBUG
123     cerr << "\t\t\t\t\tENTRY    DictionaryEntry_article DESTRUCTOR" << endl;
124 #endif
125 };
126
127         //-----------------------------------------------//
128         // Member function derive_word().                //
129         //-----------------------------------------------//
130
131 Word DictionaryEntry_article::derive_word(WordDescriptor * word_desc)
132 {
```

```
133  #ifdef DEBUG
134      cerr << "\t\t\t\t\tENTRY DictionaryEntry_article::derive_word" << endl;
135  #endif
136
137          //----------- Automatics - objects -------------//
138
139      Word   derived_word;
140
141          //----------- Automatics - other ---------------//
142
143      Count  current_count;
144      Gender current_gender;
145
146          //----------- Code begins ----------------------//
147
148      current_count = word_desc->get_count();
149      current_gender = word_desc->get_gender();
150
151  #ifdef DEBUG
152      cerr << "\t\t\tDEBUG count from input word descriptor is ";
153      cerr << current_count;
154      cerr << endl;
155      cerr << "\t\t\tDEBUG gender from input word descriptor is ";
156      cerr << current_gender;
157      cerr << endl;
158  #endif
159
160      derived_word.set_index(DictionaryEntry::get_index());
161      derived_word.set_type(article);
162      derived_word.set_language(word_desc->get_language());
163      derived_word.set_count(word_desc->get_count());
164      derived_word.set_gender(word_desc->get_gender());
165
166      if (word_desc->get_language() == L_english)
167      {
168          derived_word.set_string(eng);
169      }
170      else
171      {
172          if (word_desc->get_count() == C_plural)
173          {
174              derived_word.set_string(plural);
175          }
176          else if (current_gender == G_masculine)
177          {
178              derived_word.set_string(masc);
179          }
180          else if (current_gender == G_feminine)
181          {
182              derived_word.set_string(fem);
183          }
184          else
185          {
186              derived_word.set_string(neut);
187          }
188
189      }
190
191      return(derived_word);
192
193  }
194
195
196  //------------------------- Derived class ----------------------//
197
198          //--------------------------------------------//
```



```
199          // Constructor.                               //
200          //--------------------------------------------//
201
202  DictionaryEntry_noun::DictionaryEntry_noun(int init_index,
203                                             int init_know_level,
204                                             Gender init_gend,
205                                             char * init_g_sing, char * init_g_plural,
206                                             char * init_e_sing, char * init_e_plural) :
207                                 DictionaryEntry(init_index, init_know_level)
208  {
209  #ifdef DEBUG
210      cerr << "\t\t\t\t\tENTRY    DictionaryEntry_noun CONSTRUCTOR" << endl;
211  #endif
212
213      gend = init_gend;
214      g_sing = init_g_sing;
215      g_plural = init_g_plural;
216      e_sing = init_e_sing;
217      e_plural = init_e_plural;
218  );
219
220          //--------------------------------------------//
221          // Destructor.                                //
222          //--------------------------------------------//
223
224  DictionaryEntry_noun::~DictionaryEntry_noun()
225  {
226  #ifdef DEBUG
227      cerr << "\t\t\t\t\tENTRY    DictionaryEntry_noun DESTRUCTOR" << endl;
228  #endif
229  );
230
231          //--------------------------------------------//
232          // Member function derive_word().             //
233          //--------------------------------------------//
234
235  Word DictionaryEntry_noun::derive_word(WordDescriptor * word_desc)
236  {
237  #ifdef DEBUG
238      cerr << "\t\t\t\t\tENTRY DictionaryEntry_noun::derive_word" << endl;
239  #endif
240
241          //----------- Automatics - objects -------------//
242
243      Word   derived_word;
244
245          //----------- Automatics - other ---------------//
246
247          //----------- Code begins ----------------------//
248
249      derived_word.set_index(DictionaryEntry::get_index());
250      derived_word.set_type(noun);
251      derived_word.set_language(word_desc->get_language());
252      derived_word.set_count(word_desc->get_count());
253      derived_word.set_gender(gend);
254
255      if (word_desc->get_language() == L_english)
256      {
257          if (word_desc->get_count() == C_plural)
258          {
259              derived_word.set_string(e_plural);
260          }
261          else
262          {
263              derived_word.set_string(e_sing);
264          }
```

```
265       }
266       else
267       {
268           if (word_desc->get_count() == C_plural)
269           {
270               derived_word.set_string(g_plural);
271           }
272           else
273           {
274               derived_word.set_string(g_sing);
275           }
276       }
277       return(derived_word);
278   }
279
280   //------------------------- Derived class ----------------------//
281
282           //-----------------------------------------------//
283           // Constructor.                                  //
284           //-----------------------------------------------//
285
286   DictionaryEntry_adjective::DictionaryEntry_adjective(int init_index,
287                                   int init_know_level,
288                                   char * init_e, char * init_g) :
289                           DictionaryEntry(init_index, init_know_level)
290   {
291   #ifdef DEBUG
292       cerr << "\t\t\t\t\tENTRY    DictionaryEntry_adjective CONSTRUCTOR" << endl;
293   #endif
294
295       eng = init_e;
296       ger = init_g;
297   };
298
299           //-----------------------------------------------//
300           // Destructor.                                   //
301           //-----------------------------------------------//
302
303   DictionaryEntry_adjective::~DictionaryEntry_adjective()
304   {
305   #ifdef DEBUG
306       cerr << "\t\t\t\t\tENTRY    DictionaryEntry_adjective DESTRUCTOR" << endl;
307   #endif
308   };
309
310           //-----------------------------------------------//
311           // Member function derive_word().                //
312           //-----------------------------------------------//
313
314   Word DictionaryEntry_adjective::derive_word(WordDescriptor * word_desc)
315   {
316   #ifdef DEBUG
317       cerr << "\t\t\t\t\tENTRY DictionaryEntry_adjective::derive_word" << endl;
318   #endif
319
320           //----------- Automatics - objects -------------//
321
322       Word   derived_word;
323
324           //----------- Automatics - other ---------------//
325
326           //----------- Code begins ----------------------//
327
328       derived_word.set_index(DictionaryEntry::get_index());
329       derived_word.set_type(adjective);
330       derived_word.set_language(word_desc->get_language());
```



```
331       derived_word.set_count(C_none);
332       derived_word.set_gender(G_none);
333
334       if (word_desc->get_language() == L_english)
335       {
336           derived_word.set_string(eng);
337       }
338       else
339       {
340           derived_word.set_string(ger);
341       }
342       return(derived_word);
343   }
344
345   //------------------------- Derived class ----------------------//
346
347           //-----------------------------------------------//
348           // Constructor.                                  //
349           //-----------------------------------------------//
350
351   DictionaryEntry_verb::DictionaryEntry_verb(int init_index,
352                                   int init_know_level,
353                                   char * init_e, char * init_g) :
354                           DictionaryEntry(init_index, init_know_level)
355   {
356   #ifdef DEBUG
357       cerr << "\t\t\t\t\tENTRY    DictionaryEntry_verb CONSTRUCTOR" << endl;
358   #endif
359
360       eng = init_e;
361       ger = init_g;
362   };
363
364           //-----------------------------------------------//
365           // Destructor.                                   //
366           //-----------------------------------------------//
367
368   DictionaryEntry_verb::~DictionaryEntry_verb()
369   {
370   #ifdef DEBUG
371       cerr << "\t\t\t\t\tENTRY    DictionaryEntry_verb DESTRUCTOR" << endl;
372   #endif
373   };
374
375           //-----------------------------------------------//
376           // Member function derive_word().                //
377           //-----------------------------------------------//
378
379   Word DictionaryEntry_verb::derive_word(WordDescriptor * word_desc)
380   {
381   #ifdef DEBUG
382       cerr << "\t\t\t\t\tENTRY DictionaryEntry_verb::derive_word" << endl;
383   #endif
384
385           //----------- Automatics - objects -------------//
386
387       Word   derived_word;
388
389           //----------- Automatics - other ---------------//
390
391           //----------- Code begins ----------------------//
392
393       derived_word.set_index(DictionaryEntry::get_index());
394       derived_word.set_type(verb);
395       derived_word.set_language(word_desc->get_language());
396       derived_word.set_count(C_none);
```

```
397         derived_word.set_gender(G_none);
398
399         if (word_desc->get_language() == L_english)
400         {
401             derived_word.set_string(eng);
402         }
403         else
404         {
405             derived_word.set_string(ger);
406         }
407         return(derived_word);
408     }
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   dynamicworddescriptor.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the DynamicWordDescriptor class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include "dynamicworddescriptor.h"
22
23          //-------------------------------------------------//
24          // Constructor.                                    //
25          //-------------------------------------------------//
26
27  DynamicWordDescriptor::DynamicWordDescriptor()
28  {
29  #ifdef DEBUG
30      cerr << "\t\t\t\t\tENTRY    DynamicW..D.. CONSTRUCTOR" << endl;
31  #endif
32
33          //-------------------------------------------------//
34          // Initial values are the "not yet assigned"       //
35          // values.                                         //
36          //-------------------------------------------------//
37
38      count = C_DYNAMIC;
39      gender = G_DYNAMIC;
40      language = L_DYNAMIC;
41  };
42
43          //-------------------------------------------------//
44          // Destructor.                                     //
45          //-------------------------------------------------//
46
47  DynamicWordDescriptor::~DynamicWordDescriptor()
48  {
49  #ifdef DEBUG
50      cerr << "\t\t\t\t\tENTRY    DynamicW..D.. DESTRUCTOR" << endl;
51  #endif
52  };
53
54
55          //-------------------------------------------------//
56          // Member function get_count().                    //
57          //-------------------------------------------------//
58
59  Count DynamicWordDescriptor::get_count()
60  {
61  #ifdef DEBUG
62      cerr << "\t\t\t\t\tENTRY    DynamicW..D..::get_count" << endl;
63  #endif
64
65          //----------- Automatics - objects -------------//
66
```



```
67          //----------- Automatics - other ---------------//
68
69          //----------- Code begins ----------------------//
70
71
72      return(count);
73  };
74
75          //-------------------------------------------------//
76          // Member function get_gender().                   //
77          //-------------------------------------------------//
78
79  Gender DynamicWordDescriptor::get_gender()
80  {
81  #ifdef DEBUG
82      cerr << "\t\t\t\t\tENTRY    DynamicW..D..::get_gender" << endl;
83  #endif
84
85          //----------- Automatics - objects -------------//
86
87          //----------- Automatics - other ---------------//
88
89          //----------- Code begins ----------------------//
90
91
92      return(gender);
93  };
94
95
96          //-------------------------------------------------//
97          // Member function resolve_dynamics().             //
98          //-------------------------------------------------//
99
100 void DynamicWordDescriptor::resolve_dynamics(Word * in_word)
101 {
102 #ifdef DEBUG
103     cerr << "\t\t\t\t\tENTRY    DynamicW..D..::resolve_dynamics" << endl;
104
105     cerr << "\t\t\tDEBUG count, gender, lang before resolve are " <<
106                         count << " " << gender << " " << language << endl;
107 #endif
108
109         //----------- Automatics - objects -------------//
110
111         //----------- Automatics - other ---------------//
112
113         //----------- Code begins ----------------------//
114
115         //-------------------------------------------------//
116         // Set internal values based on the input          //
117         // word.                                           //
118         //-------------------------------------------------//
119
120     count = in_word->get_count();
121     gender = in_word->get_gender();
122     language = in_word->get_language();
123
124 #ifdef DEBUG
125     cerr << "\t\t\tDEBUG count, gender, lang after resolve are " <<
126 #endif
127                         count << " " << gender << " " << language << endl;
128 }
129
130         //-------------------------------------------------//
131         // Member function get_language().                 //
132         //-------------------------------------------------//
```

```
133
134   Language DynamicWordDescriptor::get_language()
135   {
136   #ifdef DEBUG
137       cerr << "\t\t\t\t\tENTRY    DynamicW..D..::get_language" << endl;
138   #endif
139
140           //----------- Automatics - objects -------------//
141
142           //----------- Automatics - other ---------------//
143
144           //----------- Code begins ----------------------//
145
146
147       return(language);
148   );
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   exit_tutor.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This function is called to terminate the current interactive
8     //              session, and save any run-time gathered data.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-------------------------------------------------------------------------------
18
19          //----------- Included files -------------------//
20
21    #include <iostream.h>
22    #include <stdlib.h>
23    #include <time.h>
24    #include "stdtype.h"
25    #include "log.h"
26    #include "statistics.h"
27    #include "userlist.h"
28
29          //----------- External Data --------------------//
30
31    extern  Log             log;
32    extern  Statistics      stats;
33    extern  time_t          session_start_time;
34    extern  UserList        users;
35
36          //----------- Calling Syntax -------------------//
37
38    void exit_tutor()
39    {
40
41          //----------- Automatics - objects -------------//
42
43          //----------- Automatics - other ---------------//
44
45        //time_t            session_end_time;
46        //time_t            tloc;
47        //double            duration;
48        //unsigned long     int_val;
49        //unsigned long     hours;
50        //unsigned long     minutes;
51        //unsigned long     seconds;
52
53          //----------- Code begins ----------------------//
54
55    #ifdef DEBUG
56        cerr << "\t\t\t\t\tENTRY exit_tutor()" << endl;
57    #endif
58
59          //------------------------------------------------//
60          // Print a final statistics summary for the      //
61          // user.                                         //
62          //------------------------------------------------//
63
64        cout << endl;
65        stats.print_summary();
66
```



```
67          //------------------------------------------------//
68          // Log final statistics, and write the log       //
69          // file.                                         //
70          //------------------------------------------------//
71
72        log.final_stats(stats);
73        log.save();
74
75          //------------------------------------------------//
76          // Save the user data file.                      //
77          //------------------------------------------------//
78
79        users.write();
80
81          //------------------------------------------------//
82          // Say goodbye...                                //
83          //------------------------------------------------//
84
85        cout << endl;
86        cout << "\tYour session with the German Language Tutor is complete." << endl
      ;
87        cout << endl;
88        exit(0);
89    }
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   hint.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the Hint class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------------
17
18            //----------- Included files -------------------//
19
20    #include <iostream.h>
21    #include <stdlib.h>
22    #include "hint.h"
23
24            //-----------------------------------------------//
25            // Constructor.                                  //
26            //-----------------------------------------------//
27
28    Hint::Hint(char * in_hint)
29    {
30    #ifdef DEBUG
31        cerr << "\t\t\t\t\tENTRY    Hint CONSTRUCTOR" << endl;
32    #endif
33
34        hint = in_hint;
35
36    };
37
38            //-----------------------------------------------//
39            // Destructor.                                   //
40            //-----------------------------------------------//
41
42    Hint::~Hint()
43    {
44    #ifdef DEBUG
45        cerr << "\t\t\t\t\tENTRY    Hint DESTRUCTOR" << endl;
46    #endif
47
48    };
49
50            //-----------------------------------------------//
51            // Member function print().                      //
52            //-----------------------------------------------//
53
54    void Hint::print()
55    {
56    #ifdef DEBUG
57        cerr << "\t\t\t\t\tENTRY Hint::print" << endl;
58    #endif
59
60            //----------- Automatics - objects -------------//
61
62            //----------- Automatics - other ---------------//
63
64            //----------- Code begins ----------------------//
65
66        cout << endl;
```



```
67        cout << "\t" << hint << endl;
68        cout << endl;
69
70    };
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   log.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the run-time Log class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include <fstream.h>
22  #include <stdlib.h>
23  #include <iomanip.h>
24  #include <string.h>
25  #include "log.h"
26  #include "logentry.h"
27  #include "statistics.h"
28  #include "word_and_phrase_defs.h"
29  #include "statistics_defs.h"
30  #include "userinterface.h"
31  #include "userinterface_defs.h"
32
33          //----------- External data --------------------//
34
35  extern UserInterface userint;
36
37          //------------------------------------------------//
38          // Constructor.                                   //
39          //------------------------------------------------//
40
41  Log::Log()
42  {
43  #ifdef DEBUG
44      cerr << "\t\t\t\t\tENTRY    Log CONSTRUCTOR" << endl;
45  #endif
46
47      current_username = NULL;
48      current_presentation_phrase = NULL;
49
50      current_entry_count = 0;
51      current_model_index = 0;
52
53      list = new LogEntry * [MAX_CURRENT_LOG_ENTRIES];
54  };
55
56          //------------------------------------------------//
57          // Destructor.                                    //
58          //------------------------------------------------//
59
60  Log::~Log()
61  {
62  #ifdef DEBUG
63      cerr << "\t\t\t\t\tENTRY    Log DESTRUCTOR" << endl;
64  #endif
65
66      for (int i = 0; i < current_entry_count; i++)
```



```
67      {
68          delete list[current_entry_count];
69      }
70      delete [] list;
71      delete [] current_username;
72      delete [] current_presentation_phrase;
73  };
74
75          //------------------------------------------------//
76          // Member function set_current_model_index().     //
77          //------------------------------------------------//
78
79  void Log::set_current_model_index(int in_index)
80  {
81  #ifdef DEBUG
82      cerr << "\t\t\t\t\tENTRY Log::set_current_model_index" << endl;
83  #endif
84
85          //----------- Automatics - objects -------------//
86
87          //----------- Automatics - other ---------------//
88
89          //----------- Code begins ----------------------//
90
91
92      cerr << "\t\t\tDEBUG setting current model index to " << in_index << endl;
93      current_model_index = in_index;
94  };
95
96          //------------------------------------------------//
97          // Member function at_leaf().                     //
98          //------------------------------------------------//
99
100 void Log::at_leaf(LE_at_leaf_type in_type, Phrase in_exp_response,
101                   Phrase in_user_response)
102 {
103 #ifdef DEBUG
104     cerr << "\t\t\t\t\tENTRY Log::at_leaf" << endl;
105 #endif
106
107         //----------- Automatics - objects -------------//
108
109         //----------- Automatics - other ---------------//
110
111     LogEntry_at_leaf *  entry_ptr;
112
113         //----------- Code begins ----------------------//
114
115
116         //------------------------------------------------//
117         // Create a new LogEntry object and add it to     //
118         // the current list.                              //
119         //------------------------------------------------//
120
121     entry_ptr = new LogEntry_at_leaf(current_username,
122                                      current_model_index,
123                                      current_presentation_phrase, in_type,
124                                      in_exp_response, in_user_response);
125
126     list[current_entry_count++] = entry_ptr;
127 };
128
129         //------------------------------------------------//
130         // Member function final_stats().                 //
131         //------------------------------------------------//
132
```

```
133  void Log::final_stats(Statistics in_stats)
134  {
135  #ifdef DEBUG
136      cerr << "\t\t\t\t\tENTRY Log::final_stats" << endl;
137  #endif
138
139          //----------- Automatics - objects -------------//
140
141          //----------- Automatics - other ---------------//
142
143      LogEntry_final_stats *      entry_ptr;
144
145          //----------- Code begins ----------------------//
146
147
148          //------------------------------------------------//
149          // Create a new LogEntry object and add it to   //
150          // the current list.                            //
151          //------------------------------------------------//
152
153      entry_ptr = new LogEntry_final_stats(current_username, in_stats);
154
155      list[current_entry_count++] = entry_ptr;
156  };
157
158          //------------------------------------------------//
159          // Member function save().                      //
160          //------------------------------------------------//
161
162  void Log::save()
163  {
164  #ifdef DEBUG
165      cerr << "\t\t\t\t\tENTRY Log::save" << endl;
166  #endif
167
168          //----------- Automatics - objects -------------//
169
170      ofstream outFile;
171
172          //----------- Automatics - other ---------------//
173
174          //----------- Code begins ----------------------//
175
176          //------------------------------------------------//
177          // Open the default log file to append the      //
178          // current data.                                //
179          //------------------------------------------------//
180
181      outFile.open("glt.log",ios::app);
182      if (!outFile)
183      {
184          cerr << "ERROR cannot open glt.log for append" << endl;
185      }
186      else
187      {
188
189          //------------------------------------------------//
190          // Loop through the current list of LogEntries, //
191          // and request that each print itself to the    //
192          // log file.                                    //
193          //------------------------------------------------//
194
195          for (int i=0; i<current_entry_count; i++)
196          {
197              list[i]->print(&outFile);
198          }
```



```
199
200      )
201  );
202
203          //------------------------------------------------//
204          // Member function set_current_user().          //
205          //------------------------------------------------//
206
207  void Log::set_current_user(char * in_user)
208  {
209  #ifdef DEBUG
210      cerr << "\t\t\t\t\tENTRY Log::set_current_user" << endl;
211  #endif
212
213          //----------- Automatics - objects -------------//
214
215          //----------- Automatics - other ---------------//
216
217          //----------- Code begins ----------------------//
218
219      current_username = new char[sizeof(in_user)];
220      strcpy(current_username,in_user);
221
222      cerr << "\t\t\tDEBUG current user was set to " << current_username << endl;
223  }
224
225          //------------------------------------------------//
226          // Member function set_current_presentation_    //
227          // phrase().                                    //
228          //------------------------------------------------//
229
230  void Log::set_current_presentation_phrase(Phrase in_pres_phrase)
231  {
232  #ifdef DEBUG
233      cerr << "\t\t\t\t\tENTRY Log::set_current_presentation_phrase" << endl;
234  #endif
235
236          //----------- Automatics - objects -------------//
237
238          //----------- Automatics - other ---------------//
239
240          //----------- Code begins ----------------------//
241
242
243      current_presentation_phrase = new char [MAX_PHRASE_STRING_LENGTH];
244
245      in_pres_phrase.get_string(current_presentation_phrase);
246
247      cerr << "\t\t\tDEBUG current pres phrase was set to " << current_presentatio
     n_phrase << endl;
248  );
249
250          //------------------------------------------------//
251          // Member function load().                      //
252          //------------------------------------------------//
253
254  void Log::load()
255  {
256  #ifdef DEBUG
257      cerr << "\t\t\t\t\tENTRY Log::load" << endl;
258  #endif
259
260          //----------- Automatics - objects -------------//
261
262      ifstream log_stream;
263
```

```
264         //----------- Automatics - other ---------------//
265
266     char        ch;
267     int         int_input;
268     LE_type     next_entry_type;
269     LogEntry_at_leaf * LE_at_leaf_ptr;
270     LogEntry_final_stats * LE_final_stats_ptr;
271
272         //----------- Code begins ----------------------//
273
274
275         //------------------------------------------------//
276         // Open the log file for input.  Read the       //
277         // Logentry type for each entry, create an      //
278         // appropriate LogEntry object, and ask it      //
279         // to read its data.                            //
280         //------------------------------------------------//
281
282     log_stream.open("glt.log", ios::in);
283     if (!log_stream)
284     {
285         cerr << "ERROR - could not open log file." << endl;
286     }
287
288     while (log_stream.get(ch))
289     {
290         log_stream.putback(ch);
291         log_stream >> int_input;
292         log_stream.get(ch);
293         next_entry_type = (LE_type) int_input;
294
295         switch(next_entry_type)
296         {
297             case LE_at_leaf:
298                 LE_at_leaf_ptr = new LogEntry_at_leaf(LE_at_leaf);
299                 LE_at_leaf_ptr->read(&log_stream);
300                 list[current_entry_count] = LE_at_leaf_ptr;
301                 current_entry_count++;
302                 break;
303             case LE_final_stats:
304                 LE_final_stats_ptr = new LogEntry_final_stats(LE_final_stats);
305                 LE_final_stats_ptr->read(&log_stream);
306                 list[current_entry_count] = LE_final_stats_ptr;
307                 current_entry_count++;
308                 break;
309             case LE_UNKNOWN_TYPE:
310             default:
311                 cerr << "ERROR: Bad log entry type read." << endl;
312                 break;
313         }
314
315     }
316
317     log_stream.close();
318 );
319
320         //------------------------------------------------//
321         // Member function view_at_leafs().             //
322         //------------------------------------------------//
323
324 void Log::view_at_leafs()
325 {
326 #ifdef DEBUG
327     cerr << "\t\t\t\t\tENTRY Log::view_at_leafs" << endl;
328 #endif
329
```



```
330     boolean             found = FALSE;
331     ControlCommand      choice;
332     LogEntry_at_leaf * cur_ptr;
333
334         //------------------------------------------------//
335         // Loop through log.  If appropriate entry      //
336         // is found, ask it to display itself, then     //
337         // present user with continuation menu.         //
338         //------------------------------------------------//
339
340     for (int i=0; i<current_entry_count; i++)
341     {
342         if (list[i]->get_type() == LE_at_leaf)
343         {
344             found = TRUE;
345             cur_ptr = (LogEntry_at_leaf *) list[i];
346             cur_ptr->display();
347
348             choice = userint.after_view();
349             if (choice == CC_quit)
350             {
351                 break;
352             }
353             else if (choice != CC_admin_find_next)
354             {
355                 cerr << "ERROR: bad choice returned" << endl;
356             }
357         }
358     }
359
360         //------------------------------------------------//
361         // If any were found, and we're now out of      //
362         // the loop, say no more, otherwise, say none   //
363         // to begin with.                               //
364         //------------------------------------------------//
365
366     if (found == TRUE)
367     {
368       if (choice != CC_quit)
369       {
370         cout << endl;
371         cout << "No further diagnostic failure entries found." << endl;
372         cout << endl;
373       }
374     }
375     else
376     {
377         cout << endl;
378         cout << "No diagnostic failure entries were found in the log." << endl;
379         cout << endl;
380     }
381
382
383 );
384
385         //------------------------------------------------//
386         // Member function view_ind_stats().            //
387         //------------------------------------------------//
388
389 void Log::view_ind_stats()
390 {
391 #ifdef DEBUG
392     cerr << "\t\t\t\t\tENTRY Log::view_ind_stats" << endl;
393 #endif
394
395     boolean             found=FALSE;
```

```
396       ControlCommand      choice;
397       LogEntry_final_stats * cur_ptr;
398
399           //-----------------------------------------------//
400           // Loop through log.  If appropriate entry      //
401           // is found, ask it to display itself, then     //
402           // present user with continuation menu.         //
403           //-----------------------------------------------//
404
405       for (int i=0; i<current_entry_count; i++)
406       {
407           if (list[i]->get_type() == LE_final_stats)
408           {
409               found = TRUE;
410               cur_ptr = (LogEntry_final_stats *) list[i];
411               cur_ptr->display();
412
413               choice = userint.after_view();
414               if (choice == CC_quit)
415               {
416                   break;
417               }
418               else if (choice != CC_admin_find_next)
419               {
420                   cerr << "ERROR: bad choice returned" << endl;
421               }
422           }
423       }
424
425           //-----------------------------------------------//
426           // If any were found, and we're now out of      //
427           // the loop, say no more, otherwise, say none   //
428           // to begin with.                               //
429           //-----------------------------------------------//
430
431       if (found == TRUE)
432       {
433         if (choice != CC_quit)
434         {
435           cout << endl;
436           cout << "No further statistics entries found." << endl;
437           cout << endl;
438         }
439       }
440       else
441       {
442           cout << endl;
443           cout << "No statistics entries were found in the log." << endl;
444           cout << endl;
445       }
446
447   };
448
449           //-----------------------------------------------//
450           // Member function view_cum_stats().            //
451           //-----------------------------------------------//
452
453   void Log::view_cum_stats()
454   {
455   #ifdef DEBUG
456       cerr << "\t\t\t\t\tENTRY Log::view_cum_stats" << endl;
457   #endif
458
459       ControlCommand      choice;
460       unsigned long       int_val;
461       unsigned long       hours;
```



```
462       unsigned long       minutes;
463       unsigned long       seconds;
464       int         num_stats=0;
465       long int    cum_duration=0;
466       long int    avg_duration;
467       int         cum_exercise_count=0;
468       int         cum_attempt_count=0;
469       int         cum_correct_count=0;
470       Statistics *        cur_stats;
471
472       LogEntry_final_stats * cur_ptr;
473
474           //-----------------------------------------------//
475           // Loop through log looking for individual      //
476           // statistics entries.  If found, gather the    //
477           // data.                                        //
478           //-----------------------------------------------//
479
480       for (int i=0; i<current_entry_count; i++)
481       {
482           if (list[i]->get_type() == LE_final_stats)
483           {
484               cur_ptr = (LogEntry_final_stats *) list[i];
485
486               num_stats++;
487               cur_stats = cur_ptr->get_stats();
488               cum_duration += (time_t) cur_stats->get_prev_duration();
489               cum_exercise_count += cur_stats->get_exercise_count();
490               cum_attempt_count += cur_stats->get_attempt_count();
491               cum_correct_count += cur_stats->get_correct_count();
492           }
493       }
494
495           //-----------------------------------------------//
496           // If stats entries were found, calculate,      //
497           // format, and display data to user.            //
498           //-----------------------------------------------//
499
500       if (num_stats > 0)
501       {
502
503           cout << endl;
504           cout << "Cumulative statistics from log file." << endl;
505
506           cout << "\tNumber of individual sessions counted - " << num_stats << end
      l;
507
508           int_val = (unsigned long) cum_duration;
509           hours = int_val/NUM_SECONDS_PER_HOUR;
510           int_val = int_val - (hours * NUM_SECONDS_PER_HOUR);
511           minutes = int_val/NUM_SECONDS_PER_MINUTE;
512           seconds = int_val-(minutes * NUM_SECONDS_PER_MINUTE);
513
514           cout << "\tTotal duration of sessions - ";
515           if (hours < 1)
516           {
517               cout << minutes << " minutes, " << seconds << " seconds." << endl;
518           }
519           else
520           {
521               cout << hours << " hours, " << minutes << " minutes." << endl;
522           }
523
524           avg_duration = cum_duration/num_stats;
525           int_val = (unsigned long) avg_duration;
526           hours = int_val/NUM_SECONDS_PER_HOUR;
```

```
527             int_val = int_val - (hours * NUM_SECONDS_PER_HOUR);
528             minutes = int_val/NUM_SECONDS_PER_MINUTE;
529             seconds = int_val-(minutes * NUM_SECONDS_PER_MINUTE);
530
531             cout << "\tAverage duration of sessions - ";
532             if (hours < 1)
533             {
534                 cout << minutes << " minutes, " << seconds << " seconds." << endl;
535             }
536             else
537             {
538                 cout << hours << " hours, " << minutes << " minutes." << endl;
539             }
540             cout << "\tTotal number of phrases presented by system - " << cum_exerci
    se_count << endl;
541             cout << "\tTotal number of student responses - " << cum_attempt_count <<
     endl;
542             cout << "\tTotal number of correct translations - " << cum_correct_count
     << endl;
543
544         }
545         else
546         {
547             cout << endl;
548             cout << "\tNo statistics entries found in log file." << endl;
549             cout << endl;
550         }
551
552         choice = userint.after_cum();
553         if (choice != CC_quit)
554         {
555             cerr << "ERROR: bad choice returned" << endl;
556         }
557     };
```

```
1     //--------------------------------------------------------------------------------
2     //
3     //  Filename:   logentry.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the LogEntry class and its derived
8     //              classes.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //--------------------------------------------------------------------------------
18
19            //----------- Included files -------------------//
20
21    #include <iostream.h>
22    #include <fstream.h>
23    #include <string.h>
24    #include <time.h>
25    #include "logentry.h"
26    #include "statistics.h"
27    #include "word_and_phrase_defs.h"
28
29    //------------------------- Base class --------------------------//
30
31            //-------------------------------------------//
32            // Constructor 1 of 2.                       //
33            //-------------------------------------------//
34
35    LogEntry::LogEntry()
36    {
37    #ifdef DEBUG
38        cerr << "\t\t\t\t\tENTRY    LogEntry CONSTRUCTOR 1 of 2" << endl;
39        cerr << "\t\t\t\t\tENTRY    (automatics to be read)" << endl;
40    #endif
41
42            //-------------------------------------------//
43            // Expect that internal data will be set by  //
44            // a derived class read().                   //
45            //-------------------------------------------//
46
47    );
48            //-------------------------------------------//
49            // Constructor 2 of 2.                       //
50            //-------------------------------------------//
51
52    LogEntry::LogEntry(char * in_username, int in_index,
53                       LE_type in_type)
54    {
55    #ifdef DEBUG
56        cerr << "\t\t\t\t\tENTRY    LogEntry CONSTRUCTOR 2 of 2" << endl;
57        cerr << "\t\t\t\t\tENTRY    (for newly logged entries)" << endl;
58    #endif
59
60            //-------------------------------------------//
61            // Set internal data based on input, and     //
62            // stamp the new entry with the current date //
63            // and time.                                 //
64            //-------------------------------------------//
65
66
```



```
67        time_t      tloc;
68
69        username = new char[sizeof(in_username)];
70        strcpy(username,in_username);
71        model_index = in_index;
72        record_type = in_type;
73        date_time_stamp = time(&tloc);
74    );
75
76            //-------------------------------------------//
77            // Destructor.                               //
78            //-------------------------------------------//
79
80    LogEntry::~LogEntry()
81    {
82    #ifdef DEBUG
83        cerr << "\t\t\t\t\tENTRY    LogEntry DESTRUCTOR" << endl;
84    #endif
85
86        delete [] username;
87    );
88
89
90            //-------------------------------------------//
91            // Member function get_type().               //
92            //-------------------------------------------//
93
94    LE_type LogEntry::get_type()
95    {
96    #ifdef DEBUG
97        cerr << "\t\t\t\t\tENTRY LogEntry::get_type" << endl;
98    #endif
99
100       return(record_type);
101
102   );
103
104   //------------------------- Derived class ----------------------//
105
106           //-------------------------------------------//
107           // Constructor 1 of 2.                       //
108           //-------------------------------------------//
109
110   LogEntry_at_leaf::LogEntry_at_leaf(LE_type in_type)
111   {
112   #ifdef DEBUG
113       cerr << "\t\t\t\t\tENTRY    LogEntry_at_leaf CONSTRUCTOR 1 of 2" << endl;
114       cerr << "\t\t\t\t\tENTRY    (for automatics to be read)" << endl;
115   #endif
116
117       record_type = in_type;
118
119   );
120
121           //-------------------------------------------//
122           // Constructor 2 of 2.                       //
123           //-------------------------------------------//
124
125   LogEntry_at_leaf::LogEntry_at_leaf(char * in_username, int in_index,
126           char * in_pres_phrase,
127           LE_at_leaf_type in_type, Phrase in_exp_response,
128           Phrase in_user_response) : LogEntry(in_username, in_index, LE_at_leaf)
129   {
130   #ifdef DEBUG
131       cerr << "\t\t\t\t\tENTRY    LogEntry_at_leaf CONSTRUCTOR 2 of 2" << endl;
132       cerr << "\t\t\t\t\tENTRY    (for run-time data entries)" << endl;
```

```
133  #endif
134
135          //-----------------------------------------------//
136          // Set entry-specific data.                      //
137          //-----------------------------------------------//
138
139      expected_response = new char [MAX_PHRASE_STRING_LENGTH];
140      user_response = new char [MAX_PHRASE_STRING_LENGTH];
141      presentation_phrase = new char [MAX_PHRASE_STRING_LENGTH];
142
143      in_exp_response.get_string(expected_response);
144      in_user_response.get_string(user_response);
145      strcpy(presentation_phrase, in_pres_phrase);
146
147  #ifdef DEBUG
148      cerr << "\t\t\tDEBUG expected response is " << expected_response << endl;
149      cerr << "\t\t\tDEBUG actual response is " << user_response << endl;
150      cerr << "\t\t\tDEBUG presentation phrase is ' << presentation_phrase << endl
     ;
151  #endif
152
153      type = in_type;
154  ];
155
156          //-----------------------------------------------//
157          // Destructor.                                   //
158          //-----------------------------------------------//
159
160  LogEntry_at_leaf::~LogEntry_at_leaf()
161  {
162  #ifdef DEBUG
163      cerr << "\t\t\t\t\tENTRY    LogEntry_at_leaf DESTRUCTOR" << endl;
164  #endif
165
166      delete [] presentation_phrase;
167      delete [] expected_response;
168      delete [] user_response;
169  ];
170
171
172          //-----------------------------------------------//
173          // Member function print().                      //
174          //-----------------------------------------------//
175
176  void LogEntry_at_leaf::print(ofstream * log_stream)
177  {
178  #ifdef DEBUG
179      cerr << "\t\t\t\t\tENTRY LogEntry_at_leaf::print" << endl;
180  #endif
181
182          //----------- Automatics - objects -------------//
183
184          //----------- Automatics - other ---------------//
185
186          //----------- Code begins ----------------------//
187
188      *log_stream << record_type << endl;
189      *log_stream << type << endl;
190      *log_stream << username << endl;
191      *log_stream << date_time_stamp << endl;
192      *log_stream << model_index << endl;
193      *log_stream << presentation_phrase << endl;
194      *log_stream << expected_response << endl;
195      *log_stream << user_response << endl;
196
197  ];
```



```
198
199          //-----------------------------------------------//
200          // Member function read().                       //
201          //-----------------------------------------------//
202
203  void LogEntry_at_leaf::read(ifstream * log_stream)
204  {
205      int          in_int;
206      char         in_ch;
207      char         input_line[MAX_PHRASE_STRING_LENGTH];
208      int          ch_cnt;
209      boolean      end_of_line;
210
211          //-----------------------------------------------//
212          // Read individual data entries from specified   //
213          // I/O stream.                                   //
214          //-----------------------------------------------//
215
216      *log_stream >> in_int;
217      type = (LE_at_leaf_type) in_int;
218      log_stream->get(in_ch);
219
220      end_of_line = FALSE;
221      for (ch_cnt = 0; !end_of_line && log_stream->get(in_ch); ch_cnt++)
222      {
223          if (in_ch == '\n')
224          {
225              end_of_line = TRUE;
226          }
227          else
228          {
229              cerr << "\t\t\tDEBUG int read is -" << (int) in_ch << "-" << endl;
230              input_line[ch_cnt] = in_ch;
231          }
232      }
233      input_line[ch_cnt-1] = '\0';
234      cerr << "\t\t\tDEBUG length of input line is " << strlen(input_line) << endl
     ;
235
236      username = new char[ch_cnt];
237      strcpy(username, input_line);
238      cerr << "\t\t\tDEBUG length of username is " << strlen(username) << endl;
239
240      cerr << "\t\t\tDEBUG username as read from leaf record: " << endl;
241      cerr << "\t\t\tDEBUG    " << username << endl;
242
243      *log_stream >> in_int;
244      date_time_stamp = (time_t) in_int;
245      log_stream->get(in_ch);
246
247      *log_stream >> model_index;
248      log_stream->get(in_ch);
249
250      end_of_line = FALSE;
251      for (ch_cnt = 0; !end_of_line && log_stream->get(in_ch); ch_cnt++)
252      {
253          if (in_ch == '\n')
254          {
255              end_of_line = TRUE;
256          }
257          else
258          {
259              cerr << "\t\t\tDEBUG int read is -" << (int) in_ch << "-" << endl;
260              input_line[ch_cnt] = in_ch;
261          }
262      }
```

```
263       input_line[ch_cnt-1] = '\0';
264
265       presentation_phrase = new char[ch_cnt];
266       strcpy(presentation_phrase, input_line);
267
268
269       cerr << "\t\t\tDEBUG pres phrase as read from leaf record: " << endl;
270       cerr << "\t\t\tDEBUG    " << presentation_phrase << endl;
271
272
273
274       end_of_line = FALSE;
275       for (ch_cnt = 0; !end_of_line && log_stream->get(in_ch); ch_cnt++)
276       {
277           if (in_ch == '\n')
278           {
279               end_of_line = TRUE;
280           }
281           else
282           {
283               input_line[ch_cnt] = in_ch;
284           }
285       }
286       input_line[ch_cnt-1] = '\0';
287
288       expected_response = new char[ch_cnt];
289       strcpy(expected_response, input_line);
290
291
292       end_of_line = FALSE;
293       for (ch_cnt = 0; !end_of_line && log_stream->get(in_ch); ch_cnt++)
294       {
295           if (in_ch == '\n')
296           {
297               end_of_line = TRUE;
298           }
299           else
300           {
301               input_line[ch_cnt] = in_ch;
302           }
303       }
304       input_line[ch_cnt-1] = '\0';
305
306       user_response = new char[ch_cnt];
307       strcpy(user_response, input_line);
308
309   );
310
311           //-------------------------------------------------//
312           // Member function display().                      //
313           //-------------------------------------------------//
314
315   void LogEntry_at_leaf::display()
316   {
317       cerr << "\t\t\t\tENTRY LogEntry_at_leaf::display" << endl;
318
319           //------------ Automatics - objects --------------//
320
321           //------------ Automatics - other ----------------//
322
323       struct tm *         my_ptr;
324       char                formatted_date_time_stamp[MAX_DATE_STAMP_SIZE];
325
326           //------------ Code begins -----------------------//
327
328       cout << endl;
```



```
329       if (type == LE_at_leaf_hint)
330       {
331           cout << "Leaf node encountered while looking for additional hints." << e
      ndl;
332       }
333       else
334       {
335           cout << "Leaf node encountered while trying to diagnose." << endl;
336       }
337
338           //-------------------------------------------------//
339           // Format the date/time stamp, and output all    //
340           // entry data to the specified log file.         //
341           //-------------------------------------------------//
342
343       my_ptr = localtime( &date_time_stamp );
344       (void) strftime(    formatted_date_time_stamp,
345                           MAX_DATE_STAMP_SIZE,
346                           "%a %b %d %Y",
347                           my_ptr  );
348
349       cout << "\tDate - " << formatted_date_time_stamp << "    Username - " << use
      rname << endl;
350       cout << "\tIndex of model chosen - " << model_index << endl;
351       cout << "\tPhrase which was to be translated - " << presentation_phrase << e
      ndl;
352       cout << "\tExpected response - " << expected_response << endl;
353       cout << "\tActual response - " << user_response << endl;
354       cout << endl;
355
356   );
357   //------------------------- Derived class -------------------------//
358
359           //-------------------------------------------------//
360           // Constructor 1 of 2.                           //
361           //-------------------------------------------------//
362
363   LogEntry_final_stats::LogEntry_final_stats(LE_type in_type)
364   {
365   #ifdef DEBUG
366       cerr << "\t\t\t\tENTRY    LogEntry_final_stats CONSTRUCTOR 1 of 2" << endl
      ;
367       cerr << "\t\t\t\tENTRY    (for automatics to be read)" << endl;
368   #endif
369
370       record_type = in_type;
371
372   );
373
374           //-------------------------------------------------//
375           // Constructor 2 of 2.                           //
376           //-------------------------------------------------//
377
378   LogEntry_final_stats::LogEntry_final_stats(char * in_username,
379           Statistics in_stats) : LogEntry(in_username, 0, LE_final_stats)
380   {
381   #ifdef DEBUG
382       cerr << "\t\t\t\tENTRY    LogEntry_final_stats CONSTRUCTOR 2 of 2" << endl
      ;
383       cerr << "\t\t\t\tENTRY    (for run-time created entries)" << endl;
384   #endif
385
386       current_stats = in_stats;
387
388   );
389
```

```
390         //-----------------------------------------------//
391         // Destructor.                                   //
392         //-----------------------------------------------//
393
394   LogEntry_final_stats::~LogEntry_final_stats()
395   {
396   #ifdef DEBUG
397       cerr << "\t\t\t\t\tENTRY    LogEntry_final_stats DESTRUCTOR" << endl;
398   #endif
399
400   };
401
402         //-----------------------------------------------//
403         // Member function print().                      //
404         //-----------------------------------------------//
405
406   void LogEntry_final_stats::print(ofstream * log_stream)
407   {
408   #ifdef DEBUG
409       cerr << "\t\t\t\t\tENTRY LogEntry_final_stats::print" << endl;
410   #endif
411
412         //----------- Automatics - objects -------------//
413
414         //----------- Automatics - other ---------------//
415
416         //----------- Code begins ----------------------//
417
418       *log_stream << record_type << endl;
419       *log_stream << username << endl;
420       *log_stream << date_time_stamp << endl;
421
422       current_stats.print_to_file(log_stream);
423
424   };
425
426         //-----------------------------------------------//
427         // Member function read().                       //
428         //-----------------------------------------------//
429
430   void LogEntry_final_stats::read(ifstream * log_stream)
431   {
432   #ifdef DEBUG
433       cerr << "\t\t\t\t\tENTRY LogEntry_final_stats::read" << endl;
434   #endif
435
436         //----------- Automatics - objects -------------//
437
438         //----------- Automatics - other ---------------//
439
440       char in_ch;
441       int in_int;
442       char         input_line[MAX_PHRASE_STRING_LENGTH];
443       int          ch_cnt;
444       boolean      end_of_line;
445
446         //----------- Code begins ----------------------//
447
448         //-----------------------------------------------//
449         // Read each individual data entry from the      //
450         // specified I/O stream.                         //
451         //-----------------------------------------------//
452
453       end_of_line = FALSE;
454       for (ch_cnt = 0; !end_of_line && log_stream->get(in_ch); ch_cnt++)
455       {
```



```
456           if (in_ch == '\n')
457           {
458               end_of_line = TRUE;
459           }
460           else
461           {
462               cerr << "\t\t\tDEBUG int read is -" << (int) in_ch << "-" << endl;
463               input_line[ch_cnt] = in_ch;
464           }
465       }
466       input_line[ch_cnt-1] = '\0';
467       cerr << "\t\t\tDEBUG length of input line is " << strlen(input_line) << endl
    ;
468
469       username = new char[ch_cnt];
470       strcpy(username, input_line);
471       cerr << "\t\t\tDEBUG length of username is " << strlen(username) << endl;
472
473       cerr << "\t\t\tDEBUG username as read from stats record: " << endl;
474       cerr << "\t\t\tDEBUG    " << username << endl;
475
476       *log_stream >> in_int;
477       date_time_stamp = (time_t) in_int;
478       log_stream->get(in_ch);
479
480       current_stats.read(log_stream);
481
482   };
483
484         //-----------------------------------------------//
485         // Member function display().                    //
486         //-----------------------------------------------//
487
488   void LogEntry_final_stats::display()
489   {
490       cerr << "\t\t\t\t\tENTRY LogEntry_final_stats::display" << endl;
491
492           //----------- Automatics - objects -------------//
493
494           //----------- Automatics - other ---------------//
495
496       struct tm *          my_ptr;
497       char                 formatted_date_time_stamp[MAX_DATE_STAMP_SIZE];
498
499           //----------- Code begins ----------------------//
500
501           //-----------------------------------------------//
502           // Format the date/time stamp and output all     //
503           // entry data.                                   //
504           //-----------------------------------------------//
505
506       my_ptr = localtime( &date_time_stamp );
507       (void) strftime(     formatted_date_time_stamp,
508                            MAX_DATE_STAMP_SIZE,
509                            "%a %b %d %Y",
510                            my_ptr  );
511
512       cout << endl;
513       cout << endl << "Statistics after a student session." << endl;
514       cout << "\tDate - " << formatted_date_time_stamp << "    Username - " << use
    rname << endl;
515
516       current_stats.display();
517
518   };
519
```

```
520         //-------------------------------------------//
521         // Member function get_stats().              //
522         //-------------------------------------------//
523
524   Statistics * LogEntry_final_stats::get_stats()
525   {
526   #ifdef DEBUG
527       cerr << "\t\t\t\t\tENTRY LogEntry_final_stats::get_stats" << endl;
528   #endif
529
530       return(&current_stats);
531
532   };
```

```
1   //--------------------------------------------------------------------------------
2   //
3   //  Filename:   patternnode.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the PatternNode class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //--------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include "hint.h"
22  #include "patternnode.h"
23  #include "compoundstring.h"
24  #include "userinterface.h"
25  #include "log.h"
26
27          //----------- External data --------------------//
28
29  extern UserInterface userint;
30  extern Log log;
31
32          //------------------------------------------//
33          // Constructor.                             //
34          //------------------------------------------//
35
36  PatternNode::PatternNode(Pattern (*in_func)(Phrase &) , PatternNode * in_mat,
37                                      PatternNode * in_nomat, Hint * in_hint)
38  {
39  #ifdef DEBUG
40      cerr << "\t\t\t\t\tENTRY    PatternNode CONSTRUCTOR" << endl;
41  #endif
42
43      hint = in_hint;
44      pattern_constructor = in_func;
45      match_path_pattern = in_mat;
46      no_match_path_pattern = in_nomat;
47  );
48
49          //------------------------------------------//
50          // Destructor.                              //
51          //------------------------------------------//
52
53  PatternNode::~PatternNode()
54  {
55  #ifdef DEBUG
56      cerr << "\t\t\t\t\tENTRY    PatternNode DESTRUCTOR" << endl;
57  #endif
58  };
59
60          //------------------------------------------//
61          // Member function process_response().      //
62          //------------------------------------------//
63
64  boolean PatternNode::process_response(Phrase expected_response,
65                                      Phrase user_response)
66  {
```



```
67  #ifdef DEBUG
68      cerr << "\t\t\t\t\tENTRY PatternNode::process_response()" << endl;
69  #endif
70
71          //----------- Automatics - objects -------------//
72
73      Pattern wildcard_pattern = (*pattern_constructor)(expected_response);
74
75          //----------- Automatics - other ---------------//
76
77      ControlCommand      choice1;
78      boolean             try_again;
79
80          //----------- Code begins ----------------------//
81
82          //------------------------------------------//
83          // The first thing a PatternNode does to    //
84          // process a user response is to attempt    //
85          // to match it against its own generated    //
86          // Pattern.                                 //
87          //------------------------------------------//
88
89      if (wildcard_pattern.match(user_response))
90      {
91          //------------------------------------------//
92          // If the Pattern matched, it means that an //
93          // incorrect response has been recognized.  //
94          // Offer the user options.                  //
95          //------------------------------------------//
96
97          choice1 = userint.wrong_answer();
98  #ifdef DEBUG
99          cerr << "\t\t\tDEBUG choice returned to pattern is " << choice1 << endl;
100 #endif
101         if (choice1==CC_correct_answer)
102         //------------------------------------------//
103         // If the user requests the correct answer, //
104         // set the "try again" indicator to FALSE, so //
105         // that the TranslationTree will stop       //
106         // diagnosing.                              //
107         //------------------------------------------//
108         {
109 #ifdef DEBUG
110         cerr << "\t\t\tDEBUG taking correct answer action." << endl;
111 #endif
112             try_again = FALSE;
113         }
114         else if (choice1==CC_hint)
115         //------------------------------------------//
116         // If the user requests a hint, pass this   //
117         // Pattern's hint to the UserInterface object //
118         // for printing, and additional user options. //
119         //------------------------------------------//
120         {
121 #ifdef DEBUG
122         cerr << "\t\t\tDEBUG taking hint action." << endl;
123 #endif
124             choice1 = userint.after_hint(hint);
125             if (choice1==CC_hint)
126             {
127         //------------------------------------------//
128         // If the user requests an additional hint, //
129         // try to move down the Match Path for more //
130         // detailed diagnosis.  (If this is not     //
131         // possible, make a log entry indicating that //
132         // a leaf node has been hit while attempting //
```

```
133          // to give more detailed hints.                  //
134          //--------------------------------------------//
135                  if (match_path_pattern == NULL)
136                  {
137                      log.at_leaf(LE_at_leaf_hint, expected_response,
138                                  user_response);
139                      choice1 = userint.cant_diagnose_further();
140                      if (choice1 == CC_try_again)
141                      {
142                          try_again = TRUE;
143                      }
144                      else
145                      {
146                          try_again = FALSE;
147                      }
148                  }
149                  else
150                  {
151                      try_again = match_path_pattern->more_hints(
152                                  expected_response, user_response);
153                      if (try_again)
154                      {
155                          try_again = TRUE;
156                      }
157                      else
158                      {
159                          try_again = FALSE;
160                      }
161                  }
162              }
163              else if (choice1==CC_correct_answer)
164              {
165                  try_again = FALSE;
166              }
167              else
168              {
169                  try_again = TRUE;
170              }
171          }
172          else
173          {
174  #ifdef DEBUG
175          cerr << "\t\t\tDEBUG taking default action." << endl;
176  #endif
177              try_again = TRUE;
178          }
179      }
180      else
181      {
182          //--------------------------------------------//
183          // If the Pattern at this node does not match,  //
184          // try to move down the no-Match Path in an     //
185          // effort to find a match.  (If this is not     //
186          // possible, make a log entry indicating that   //
187          // a leaf node has been hit while attempting    //
188          // a diagnosis.                                 //
189          //--------------------------------------------//
190          if (no_match_path_pattern == NULL)
191          {
192              log.at_leaf(LE_at_leaf_diag, expected_response,
193                          user_response);
194              choice1 = userint.cant_diagnose();
195              if (choice1 == CC_try_again)
196              {
197                  try_again = TRUE;
198              }
```



```
199              else
200              {
201                  try_again = FALSE;
202              }
203          }
204          else
205          {
206              try_again = no_match_path_pattern->process_response(
207                                  expected_response, user_response);
208          }
209      }
210      return(try_again);
211  }
212
213          //--------------------------------------------//
214          // Member function more_hints().               //
215          //--------------------------------------------//
216
217  boolean PatternNode::more_hints(Phrase expected_response,
218                                  Phrase user_response)
219  {
220  #ifdef DEBUG
221      cerr << "\t\t\t\t\tENTRY PatternNode::more_hints()" << endl;
222  #endif
223
224          //---------- Automatics - objects -------------//
225
226      Pattern wildcard_pattern = (*pattern_constructor)(expected_response);
227
228          //---------- Automatics - other ---------------//
229
230      boolean             try_again;
231      ControlCommand      choice1;
232
233          //---------- Code begins ----------------------//
234
235          //--------------------------------------------//
236          // The first thing a PatternNode does when      //
237          // attempting to provide more detailed hints,   //
238          // is to attempt to match the current response  //
239          // against its own generated Pattern.           //
240          //--------------------------------------------//
241
242      if (wildcard_pattern.match(user_response))
243      {
244          //--------------------------------------------//
245          // If the Pattern matched, it means that the    //
246          // problem with the response has been further   //
247          // isolated.  Pass this Pattern's hint to the   //
248          // Userinterface object for printing, and       //
249          // additional user options.                     //
250          //--------------------------------------------//
251          choice1 = userint.after_hint(hint);
252          if (choice1==CC_hint)
253          {
254          //--------------------------------------------//
255          // If the user requests an additional hint,     //
256          // try to move down the Match Path for more     //
257          // detailed diagnosis.  (If this is not         //
258          // possible, make a log entry indicating that   //
259          // a leaf node has been hit while attempting    //
260          // to give more detailed hints.                 //
261          //--------------------------------------------//
262
263              if (match_path_pattern == NULL)
264              {
```

```
265                     log.at_leaf(LE_at_leaf_hint, expected_response,
266                                 user_response);
267                     choice1 = userint.cant_diagnose_further();
268                     if (choice1 == CC_try_again)
269                     {
270                         try_again = TRUE;
271                     }
272                     else
273                     {
274                         try_again = FALSE;
275                     }
276                 }
277                 else
278                 {
279                     try_again = match_path_pattern->more_hints(
280                                 expected_response, user_response);
281                 }
282             }
283             else if (choice1==CC_correct_answer)
284             {
285                 try_again = FALSE;
286             }
287             else
288             {
289                 try_again = TRUE;
290             }
291         }
292         else
293         {
294
295 //          //-----------------------------------------------//
296 //          // If the Pattern at this node does not match,  //
297 //          // then make a log entry indicating that        //
298 //          // a leaf node has been hit while attempting    //
299 //          // to give more detailed hints.                 //
300 //          //-----------------------------------------------//
301 //
302 //          log.at_leaf(LE_at_leaf_hint, expected_response,
303 //                                  user_response);
304 //          choice1 = userint.cant_diagnose_further();
305 //          if (choice1 == CC_try_again)
306 //          {
307 //              try_again = TRUE;
308 //          }
309 //          else
310 //          {
311 //              try_again = FALSE;
312 //          }
313
314             //-----------------------------------------------//
315             // If the Pattern at this node does not macth,  //
316             // attempt to head down the no-Match Path       //
317             // (still processing as "more hints".           //
318             //-----------------------------------------------//
319
320             if (no_match_path_pattern == NULL)
321             {
322                 log.at_leaf(LE_at_leaf_hint, expected_response,
323                                 user_response);
324                 choice1 = userint.cant_diagnose_further();
325                 if (choice1 == CC_try_again)
326                 {
327                     try_again = TRUE;
328                 }
329                 else
330                 {
```



```
331                     try_again = FALSE;
332                 }
333             }
334             else
335             {
336                 try_again = no_match_path_pattern->more_hints(
337                                 expected_response, user_response);
338             }
339         }
340         return(try_again);
341     }
```

```
1     //-------------------------------------------------------------------------
2     //
3     //  Filename:   phrasedescriptor.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the PhraseDescriptor class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //-------------------------------------------------------------------------
17
18            //----------- Included files --------------------//
19
20    #include <iostream.h>
21    #include "word.h"
22    #include "dictionary.h"
23    #include "worddescriptor.h"
24    #include "phrasedescriptor.h"
25    #include "dynamicworddescriptor.h"
26
27            //----------- External data --------------------//
28
29    extern Dictionary dictionary;
30
31
32            //-----------------------------------------------//
33            // Constructor.                                  //
34            //-----------------------------------------------//
35
36    PhraseDescriptor::PhraseDescriptor(int list_size, int * in_order,
37                            WordDescriptor * wd1,
38                            WordDescriptor * wd2, WordDescriptor * wd3,
39                            WordDescriptor * wd4, WordDescriptor * wd5,
40                            WordDescriptor * wd6, WordDescriptor * wd7,
41                            WordDescriptor * wd8, WordDescriptor * wd9,
42                            WordDescriptor * wd10)
43    {
44    #ifdef DEBUG
45        cerr << "\t\t\t\t\tENTRY    PhraseDescriptor CONSTRUCTOR" << endl;
46    #endif
47
48            //-----------------------------------------------//
49            // Create any array which will hold pointers     //
50            // to the individual WordDescriptors.            //
51            //-----------------------------------------------//
52
53        descriptors = new WordDescriptor * [list_size];
54
55        size = list_size;
56        fillin_order = in_order;
57
58            //-----------------------------------------------//
59            // Set any descriptors which were specified.     //
60            //-----------------------------------------------//
61
62        switch(list_size)
63        {
64            case 10:
65                    descriptors[9] = wd10;
66            case 9:
```



```
67                  descriptors[8] = wd9;
68            case 8:
69                  descriptors[7] = wd8;
70            case 7:
71                  descriptors[6] = wd7;
72            case 6:
73                  descriptors[5] = wd6;
74            case 5:
75                  descriptors[4] = wd5;
76            case 4:
77                  descriptors[3] = wd4;
78            case 3:
79                  descriptors[2] = wd3;
80            case 2:
81                  descriptors[1] = wd2;
82            case 1:
83                  descriptors[0] = wd1;
84                  break;
85            default:
86                  cerr << "\t\t\tERROR PhraseDescriptor invalid length specified"
      << endl;
87    #ifdef DEBUG
88                  cout << "\t\t\tERROR PhraseDescriptor invalid length specified"
      << endl;
89    #endif
90        }
91
92    };
93
94            //-----------------------------------------------//
95            // Destructor.                                   //
96            //-----------------------------------------------//
97
98    PhraseDescriptor::~PhraseDescriptor()
99    {
100   #ifdef DEBUG
101       cerr << "\t\t\t\t\tENTRY    PhraseDescriptor DESTRUCTOR" << endl;
102   #endif
103
104       delete [] descriptors;
105   };
106
107           //-----------------------------------------------//
108           // Member function buildphrase().                //
109           //-----------------------------------------------//
110
111
112   Phrase PhraseDescriptor::build_phrase(BasePhrase& in_bp)
113   {
114   #ifdef DEBUG
115       cerr << "\t\t\t\t\tENTRY PhraseDescriptor::build_phrase" << endl;
116   #endif
117
118           //----------- Automatics - objects -------------//
119
120       Word        words[MAX_PHRASE_LENGTH];       // not obj, but filled in below.
121       Phrase      new_phrase(size);
122
123           //----------- Automatics - other ---------------//
124
125       int         order_index;
126       int         word_index;
127
128           //----------- Code begins ----------------------//
129
130   #ifdef DEBUG
```

```
131             cerr << "\t\t\tDEBUG Echoing the word indices for the" << endl;
132             cerr << "\t\t\tDEBUG BasePhrase input to _D::build_phrase." << endl;
133             int temp = in_bp.get_index(1);
134             temp = in_bp.get_index(2);
135             temp = in_bp.get_index(3);
136             temp = in_bp.get_index(4);
137             temp = in_bp.get_index(5);
138     #endif
139
140             //-----------------------------------------------//
141             // Loop based on the size of the Phrase          //
142             // Descriptor, but process each word based       //
143             // on the fill-in order (necessary to ensure     //
144             // that words which must agree with each other   //
145             // are obtained in the proper order).            //
146             //-----------------------------------------------//
147
148         for (int size_count = 0; size_count < size; size_count++)
149         {
150             order_index = fillin_order[size_count];
151     #ifdef DEBUG
152             cerr << "\t\t\tDEBUG build_phrase order index used is ";
153             cerr << order_index << endl;
154     #endif
155
156             //-----------------------------------------------//
157             // Get the appropriate word_index and Word       //
158             // Descriptor - the two things needed in order   //
159             // to obtain an actual word from the dictionary.//
160             //-----------------------------------------------//
161
162             word_index = in_bp.get_index(order_index);
163     #ifdef DEBUG
164             cerr << "\t\t\tDEBUG got word index " << word_index << "get WD_" << endl
    ;
165     #endif
166             WordDescriptor word_desc(*(descriptors[order_index-1]));
167
168             //-----------------------------------------------//
169             // The current WordDescriptor may contain        //
170             // attributes which are dynamic (held over to    //
171             // agree with earlier words).  While this is     //
172             // true, loop back through already processed     //
173             // words until all dynamic attributes are        //
174             // resolved.                                     //
175             //-----------------------------------------------//
176
177     #ifdef DEBUG
178             cerr << "\t\t\tDEBUG got WD_, work on dynamics" << endl;
179     #endif
180             DynamicWordDescriptor dyna_WD;
181             for (int past_cnt = size_count-1; past_cnt >= 0 &&
182                                 !word_desc.no_dynamics(); past_cnt--)
183             {
184                 dyna_WD.resolve_dynamics(&words[fillin_order[past_cnt]-1]);
185                 word_desc.resolve_dynamics(dyna_WD);
186             }
187
188             //-----------------------------------------------//
189             // Get the current word from the dictionary,     //
190             // and store it.                                 //
191             //-----------------------------------------------//
192
193             words[order_index-1] = dictionary.get_word(word_index,&word_desc);
194
195         }
```



```
196
197             //-----------------------------------------------//
198             // Compile the obtained words into a new         //
199             // phrase - now syntactically and semantically   //
200             // correct - and return it to the caller.        //
201             //-----------------------------------------------//
202
203         for (int word_count = 0; word_count < size; word_count++)
204         {
205             new_phrase.set_word(word_count+1,words[word_count]);
206         }
207
208         return(new_phrase);
209
210     }
```

```
1     //--------------------------------------------------------------------------------
2     //
3     //  Filename:   phrasemodel.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This file implements the PhraseModel class.
8     //
9     //  Notes:      The theory and design of this system have been documented
10    //              in the accompanying thesis report.
11    //
12    //  Revisions   By              Reason
13    //  ---------   --              ------
14    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15    //
16    //--------------------------------------------------------------------------------
17
18            //----------- Included files --------------------//
19
20    #include <iostream.h>
21    #include "phrasemodel.h"
22    #include "log.h"
23    #include "statistics.h"
24
25    #include "dictionary.h"                                   // test
26    #include "worddescriptor.h"                               // test
27    #include "word.h"                                         // test
28
29            //----------- External Data --------------------//
30
31    extern Log log;
32    extern Statistics stats;
33
34            //-------------------------------------------//
35            // Constructor.                              //
36            //-------------------------------------------//
37
38    PhraseModel::PhraseModel(int in_know_level,
39                             BasePhrasePool& pool, PhraseDescriptor& in_desc,
40                             TranslationTree& in_tree)
41    {
42    #ifdef DEBUG
43        cerr << "\t\t\t\t\tENTRY    PhraseModel CONSTRUCTOR" << endl;
44        cerr << "\t\t\tDEBUG Addr of object " << (unsigned long) this << " size " <<
       sizeof(*this) << endl;
45    #endif
46
47        knowledge_level = in_know_level;
48        phrases = &pool;
49        fixed_phrase_desc = &in_desc;
50        trans_tree = &in_tree;
51    );
52
53            //-------------------------------------------//
54            // Destructor.                               //
55            //-------------------------------------------//
56
57    PhraseModel::~PhraseModel()
58    {
59    #ifdef DEBUG
60        cerr << "\t\t\t\t\tENTRY    PhraseModel DESTRUCTOR" << endl;
61    #endif
62    );
63
64            //-------------------------------------------//
65            // Member function do_exercise().            //
```



```
66            //-------------------------------------------//
67
68    extern Dictionary  dictionary;                                    // test
69
70    boolean PhraseModel::do_exercise()
71    {
72    #ifdef DEBUG
73        cerr << "\t\t\t\t\tENTRY PhraseModel::do_exercise" << endl;
74    #endif
75
76            //----------- Automatics - objects -------------//
77
78            //----------- Automatics - other ---------------//
79
80        BasePhrase *        chosen_BasePhrase;
81        boolean             able_to_do_exercise;
82        boolean             try_again;
83    #ifdef DEBUG
84            cerr << "\t\t\tDEBUG Addr of auto boolean try_again in PhraseModel " <<
      (unsigned long) &try_again << endl;
85    #endif
86
87            //----------- Code begins ----------------------//
88
89
90            //-------------------------------------------//
91            // Draw a BasePhrase from this model's pool.    //
92            // If none can be found (indicating that all    //
93            // valid selections have been exhausted during  //
94            // this session), indicate to caller that this  //
95            // exercise was not done.                       //
96            //-------------------------------------------//
97
98        chosen_BasePhrase = phrases->get_BasePhrase();
99        if (chosen_BasePhrase == NULL)
100       {
101           able_to_do_exercise = FALSE;
102       }
103       else
104       {
105           try_again = TRUE;
106
107           //-------------------------------------------//
108           // Create a Phrase to be presented to the user  //
109           // for translation, and request that this       //
110           // models PhraseDescriptor build the            //
111           // syntactically and semantically correct       //
112           // Phrase.                                      //
113           //-------------------------------------------//
114
115           Phrase presentation_phrase(chosen_BasePhrase->get_size());
116           presentation_phrase = fixed_phrase_desc->build_phrase(*chosen_BasePhrase
      );
117
118           //-------------------------------------------//
119           // Save the presentation phrase for possible    //
120           // use by the Log, and indicate that another    //
121           // exercise has been started.                   //
122           //-------------------------------------------//
123
124           log.set_current_presentation_phrase(presentation_phrase);
125           stats.increment_exercise_count();
126
127           //-------------------------------------------//
128           // While the user wishes to try again (or until //
129           // the correct response is diagnosed), ask the  //
```

```
130         // user to translate the current presentation   //
131         // phrase.                                      //
132         //-----------------------------------------------//
133
134         while (try_again)
135         {
136 #ifdef DEBUG
137             cerr << "Translate:" << endl;
138 #endif
139             cout << "Translate:" << endl;
140             cout << "\t\t";
141             presentation_phrase.print();
142             cout << endl;
143 #ifdef DEBUG
144             cerr << "to German:" << endl;
145             cerr << "\t\t" << endl;
146 #endif
147             cout << "to German:" << endl;
148             cout << "\t\t";
149
150         //-----------------------------------------------//
151         // Feed the response to the TranslationTree     //
152         // for diagnosis.                               //
153         //-----------------------------------------------//
154
155             try_again = trans_tree->check_response(*chosen_BasePhrase);
156             able_to_do_exercise = TRUE;
157         }
158     }
159     return(able_to_do_exercise);
160 );
161
162         //-----------------------------------------------//
163         // Member function reset_depletions().          //
164         //-----------------------------------------------//
165
166 boolean PhraseModel::reset_depletions(int in_level)
167 {
168 #ifdef DEBUG
169     cerr << "\t\t\t\t\tENTRY PhraseModel::reset_depletions" << endl;
170 #endif
171
172         //----------- Automatics - objects -------------//
173
174         //----------- Automatics - other ---------------//
175
176         //----------- Code begins ----------------------//
177
178         //-----------------------------------------------//
179         // If this PhraseModel falls within the current //
180         // user's knowledge level, request that the     //
181         // BasePhrasePool reset its selections based    //
182         // on this knowledge level.                     //
183         //-----------------------------------------------//
184
185     if (knowledge_level <= in_level)
186     {
187         if (phrases->reset_selections(in_level) == TRUE)
188         {
189 #ifdef DEBUG
190     cerr << "\t\t\tDEBUG BPP reports it has active lists" << endl;
191 #endif
192             return(TRUE);
193         }
194         else
195         {
```



```
196 #ifdef DEBUG
197     cerr << "\t\t\tDEBUG BPP reports it does not have active lists" << endl;
198 #endif
199             return(FALSE);
200         }
201     }
202     else
203     {
204         return(FALSE);
205     }
206 );
207
208         //-----------------------------------------------//
209         // Member function set_level().                 //
210         //-----------------------------------------------//
211
212 boolean PhraseModel::set_level(int in_level)
213 {
214 #ifdef DEBUG
215     cerr << "\t\t\t\t\tENTRY PhraseModel::set_level" << endl;
216 #endif
217
218         //-----------------------------------------------//
219         // Same functionality as reset_depletions().    //
220         // Used during system initialization to set     //
221         // the system's available data to the current   //
222         // user's knowledge level.                      //
223         //-----------------------------------------------//
224
225     return(this->reset_depletions(in_level));
226 );
```

```
1   //--------------------------------------------------------------------------
2   //
3   //  Filename:   phrasemodellist.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the PhraseModelList class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //--------------------------------------------------------------------------
17
18          //----------- Included files --------------------//
19
20  #include <iostream.h>
21  #include <stdlib.h>
22  #include "phrasemodellist.h"
23  #include "phrasemodel.h"
24  #include "log.h"
25
26          //----------- External data ---------------------//
27
28  extern Log log;
29
30          //-------------------------------------------//
31          // Constructor.                              //
32          //-------------------------------------------//
33
34  PhraseModelList::PhraseModelList(int in_size, PhraseModel * list)
35  {
36  #ifdef DEBUG
37      cerr << "\t\t\t\t\tENTRY    PhraseModelList CONSTRUCTOR" << endl;
38  #endif
39
40      size = in_size;
41      model_list = list;
42      depleted = new boolean[size];
43
44      for (int i=0; i<size; i++)
45      {
46          depleted[i] = FALSE;
47      }
48  };
49
50          //-------------------------------------------//
51          // Destructor.                               //
52          //-------------------------------------------//
53
54  PhraseModelList::~PhraseModelList()
55  {
56  #ifdef DEBUG
57      cerr << "\t\t\t\t\tENTRY    PhraseModelList DESTRUCTOR" << endl;
58  #endif
59
60      delete [] depleted;
61  };
62
63          //-------------------------------------------//
64          // Member function do_exercise.              //
65          //-------------------------------------------//
66
```



```
67  void PhraseModelList::do_exercise()
68  {
69  #ifdef DEBUG
70      cerr << "\t\t\t\t\tENTRY PhraseModelList::do_exercise" << endl;
71  #endif
72
73          //----------- Automatics - objects -------------//
74
75          //----------- Automatics - other ---------------//
76
77      boolean exercise_done;
78      int number_active_models = 0;
79
80          //----------- Code begins ----------------------//
81
82
83          //-------------------------------------------//
84          // Attempt to find a PhraseModel which has not  //
85          // yet been exhausted (i.e. all its available   //
86          // vocabulary selections presented during this  //
87          // session).                                    //
88          //-------------------------------------------//
89
90      exercise_done = this->find_and_try();
91
92          //-------------------------------------------//
93          // If no exercise was possible, assume that the //
94          // system has depleted all available selections //
95          // for the current knowledge level, and reset   //
96          // them.  Then try again to do an exercise.     //
97          //-------------------------------------------//
98
99      if (!exercise_done)
100     {
101         for (int i=0; i<size; i++)
102         {
103             if (model_list[i].reset_depletions(current_level) == TRUE)
104             {
105                 depleted[i] = FALSE;
106                 number_active_models++;
107             }
108             else
109             {
110                 depleted[i] = TRUE;
111             }
112         }
113         exercise_done = this->find_and_try();
114     }
115 };
116
117         //-------------------------------------------//
118         // Member function find_and_try().           //
119         //-------------------------------------------//
120
121 boolean PhraseModelList::find_and_try()
122 {
123 #ifdef DEBUG
124     cerr << "\t\t\t\t\tENTRY PhraseModelList::find_and_try" << endl;
125 #endif
126
127         //----------- Automatics - objects -------------//
128
129         //----------- Automatics - other ---------------//
130
131     boolean     exercise_done = FALSE;
132     boolean     still_trying = TRUE;
```

```
133         int         num_undepleted = 0;
134         int         rand_val;
135         int         chosen_count;
136         int         chosen_index;
137
138             //------------ Code begins ----------------------//
139
140             //------------------------------------------------//
141             // Determine how many PhraseModels have not yet //
142             // exhausted all of their vocabulary selections.//
143             //------------------------------------------------//
144
145         for (int i=0; i<size; i++)
146         {
147             if (depleted[i] == FALSE)
148             {
149                 num_undepleted++;
150             }
151         }
152
153             //------------------------------------------------//
154             // While an exercise has not yet been done,     //
155             // try the different PhraseModels.              //
156             //------------------------------------------------//
157
158         while (still_trying)
159         {
160
161             if (num_undepleted == 0)
162             {
163                 still_trying = FALSE;
164             }
165             else
166             {
167             //------------------------------------------------//
168             // Get a pseudo-random number in the range of   //
169             // undepleted PhraseModels.                     //
170             //------------------------------------------------//
171
172                 rand_val = rand();
173                 chosen_count = rand_val % num_undepleted;
174     #ifdef DEBUG
175                 cerr << "\t\t\tDEBUG number of undepleted entries is " << num_undepl
    eted << endl;
176                 cerr << "\t\t\tDEBUG random value obtained is " << rand_val << endl;
177                 cerr << "\t\t\tDEBUG chosen count is modulo above 2 and is " << chos
    en_count << endl;
178     #endif
179                 chosen_count++;
180     #ifdef DEBUG
181                 cerr << "\t\t\tDEBUG chosen count final value is " << chosen_count <
    < endl;
182     #endif
183             //------------------------------------------------//
184             // Use the chosen random index to step into     //
185             // the undepleted PhraseModels.                 //
186             //------------------------------------------------//
187
188                 for (chosen_index=0; chosen_count > 0; chosen_index++)
189                 {
190                     if (depleted[chosen_index] == FALSE)
191                     {
192                         chosen_count--;
193                     }
194                 }
195                 chosen_index--;
```



```
196
197     #ifdef DEBUG
198                 cerr << "\t\t\tDEBUG using the model at array location [";
199                 cerr << chosen_index << "]" << endl;
200     #endif
201
202             //------------------------------------------------//
203             // Ask the PhraseModel to do an exercise.  If   //
204             // it reports that no exercise could be done,   //
205             // mark it as depleted, and try another of the  //
206             // previously undepleted models.                //
207             //------------------------------------------------//
208
209                 log.set_current_model_index(chosen_index);
210                 exercise_done = model_list[chosen_index].do_exercise();
211                 if (exercise_done)
212                 {
213                     still_trying= FALSE;
214                 }
215                 else
216                 {
217                     num_undepleted--;
218                     depleted[chosen_index] = TRUE;
219                     still_trying = TRUE;
220                 }
221             }
222         }
223         return (exercise_done);
224     };
225
226             //------------------------------------------------//
227             // Member function set_current_knowledge_level. //
228             //------------------------------------------------//
229
230     void PhraseModelList::set_current_knowledge_level(int in_level)
231     {
232     #ifdef DEBUG
233         cerr << "\t\t\t\t\tENTRY PhraseModelList::set_cur_know_level" << endl;
234     #endif
235
236             //------------ Automatics - objects -------------//
237
238             //------------ Automatics - other ---------------//
239
240         int number_active_models = 0;
241
242             //------------ Code begins ----------------------//
243
244             //------------------------------------------------//
245             // Loop through list, asking each PhraseModel   //
246             // to set its knowledge level to match the      //
247             // current user's.                              //
248             //------------------------------------------------//
249
250         current_level = in_level;
251
252         for (int i=0; i<size; i++)
253         {
254             if (model_list[i].set_level(current_level) == TRUE)
255             {
256                 depleted[i] = FALSE;
257                 number_active_models++;
258             }
259             else
260             {
261                 depleted[i] = TRUE;
```

```
262             }
263         }
264     #ifdef DEBUG
265         cerr << "\t\t\tDEBUG Number of active models is " << number_active_models <<
        endl;
266     #endif
267     };
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   session.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file represents the single Agent which was left after
8   //              analysis and design.  It is the "main" program of the system,
9   //              and coordinates the major objects.
10  //
11  //  Notes:      The theory and design of this system have been documented
12  //              in the accompanying thesis report.
13  //
14  //                      **********************
15  //                      *  SPECIAL NOTE!!!   *
16  //                      **********************
17  //
18  //              If you are reading this comment, you are
19  //              probably looking at the code in some amount of detail.  You
20  //              should be warned that I was learning the language as I used
21  //              it - not the ideal situation for the program which is
22  //              being written!  As I learned C++ better, I did go back and
23  //              correct some glaring errors in usage, which no doubt would
24  //              have manisfested themselves as subtle bugs later.  However,
25  //              I am sure I didn't get them all, not to mention the myriad
26  //              style issues which I could not even begin to address given
27  //              time constraints.  I would really liked to have gone back
28  //              and reviewed and updated the code strictly for the use of
29  //              const and references, copy constructors, operator=, etc.  You
30  //              will probably run into some ugly code!  If you are planning
31  //              to enhance, or otherwise use this program, you should be
32  //              aware that it is very much a prototype, and it would be
33  //              wise to look at the thesis report, especially the
34  //              conclusions chapter.  I may also have made enhancements
35  //              myself (just for fun :-), so feel free to drop me a line
36  //              (assuming I am still at the same e-mail address) to ask.
37  //              Good luck!
38  //                      Last known e-mail address: staffan@serum.kodak.com.
39  //
40  //  Revisions   By              Reason
41  //  ---------   --              ------
42  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
43  //
44  //-------------------------------------------------------------------------------
45
46          //----------- Included files -------------------//
47
48  #include <iostream.h>
49  #include "dictionary.h"
50  #include "stdtype.h"
51  #include "session.h"
52  #include "phrasemodellist.h"
53  #include <stdlib.h>
54  #include <sys/types.h>
55  #include <sys/time.h>
56  #include "statistics.h"
57  #include "userinterface.h"
58  #include "log.h"
59  #include "userlist.h"
60
61  typedef  long  time_t;
62
63          //----------- Global Data ----------------------//
64
65  UserInterface   userint;
66  Log             log;
```



```
67  Statistics      stats;
68  UserList        users;
69  time_t          session_start_time;
70
71          //----------- External Data --------------------//
72
73  extern  PhraseModelList models;
74  extern  Dictionary      dictionary;
75
76          //----------- External Functions ---------------//
77
78  extern  void  exit_tutor();
79
80
81          //-----------------------------------------------//
82          // Main entry point for the system.              //
83          //-----------------------------------------------//
84
85  main()
86  {
87          //----------- Automatics - objects -------------//
88
89
90          //----------- Automatics - other ---------------//
91
92      time_t              long_seed;
93      time_t              tloc;
94      unsigned int        int_seed;
95      Usertype            user_type;
96      int                 exercise_count;
97      boolean             user_wants_to_continue;
98      ControlCommand      choice;
99
100         //----------- Code begins ----------------------//
101
102 #ifdef DEBUG
103     cerr << "\t\t\t\t\tENTRY session - main()" << endl;
104 #endif
105
106         //-----------------------------------------------//
107         // Note the session start time, for statistics  //
108         // purposes.                                     //
109         //-----------------------------------------------//
110
111     session_start_time = time(&tloc);
112
113 #ifdef DEBUG
114     cerr << "\t\t\tDEBUG session start time is " << session_start_time << endl;
115 #endif
116
117         //-----------------------------------------------//
118         // Print header and introductory instructions.  //
119         //-----------------------------------------------//
120
121     cout << "\n\n\t\t\tGerman Language Tutor, version 1.0" << endl << endl << en
    dl;
122     cout << "If you are not sure how to begin, type \"?\"." << endl << endl;
123 #ifdef DEBUG
124     cerr << "\n\n\tGerman Language Tutor, version 1.0" << endl << endl;
125     cerr << "If you are not sure how to begin, type \"?\"." << endl << endl;
126 #endif
127
128         //-----------------------------------------------//
129         // Seed the pseudo-random number generator for  //
130         // selection of models and vocabulary.  NOTE:   //
131         // during the system demonstration at the       //
```

```
132             // thesis defense, pseudo-random selection did  //
133             // not seem to be completely working.  The      //
134             // order of phrases presented was shuffled,     //
135             // but it _always_ started with the same        //
136             // phrase.  That was on the RIT system, and     //
137             // here on the Kodak system, it seems to work   //
138             // fine.  May have been a portability issue...  //
139             //----------------------------------------------//
140
141         long_seed = time(&tloc);
142     #ifdef DEBUG
143         cerr << "\n\t\t\tDEBUG long seed value of " << long_seed << endl;
144     #endif
145         int_seed = (unsigned int) long_seed;
146     #ifdef DEBUG
147         cerr << "\n\t\t\tDEBUG int seed value of " << int_seed << endl;
148     #endif
149         srand(int_seed);
150
151             //----------------------------------------------//
152             // Read the data file which indicates valid     //
153             // usernames.  Request the object to verify     //
154             // that the current user is valid.              //
155             //----------------------------------------------//
156
157         users.read();
158
159         user_type = users.establish_connection();
160
161     #ifdef DEBUG
162         cerr << "\n\t\t\tDEBUG connection established w/user" << endl;
163     #endif
164
165         switch (user_type)
166         {
167             case STUDENT:
168
169             //----------------------------------------------//
170             // If student user, set current knowledge       //
171             // level and begin exercises.                   //
172             //----------------------------------------------//
173
174     #ifdef DEBUG
175                 cerr << "\t\t\tDEBUG taking action for STUDENT" << endl;
176     #endif
177                 user_wants_to_continue = TRUE;
178                 exercise_count = 0;
179
180                 cout << endl;
181                 cout << "\tYou will now be asked to translate various phrases and" <
    < endl;
182                 cout << "\tsentences.  You may stop at any time - the important comm
    and" << endl;
183                 cout << "\tto remember is \"?\"   Type a question mark at anytime to
    " << endl;
184                 cout << "\tbe told your options and be given the opportunity to leav
    e"  << endl;
185                 cout << endl;
186
187                 models.set_current_knowledge_level(6);
188
189                 while (user_wants_to_continue)
190                 {
191     #ifdef DEBUG
192                     cerr << "\t\t\tDEBUG invoking models to do an exercise" << endl;
193     #endif
```



```
194                     models.do_exercise();
195                     exercise_count++;
196
197                     if ((exercise_count % MAJOR_INTERVENE_MAX) == 0)
198                     {
199             //----------------------------------------------//
200             // If the user has reached a predetermined      //
201             // number of exercises, intervene to give the   //
202             // student an explicit opportunity to stop.     //
203             //----------------------------------------------//
204     #ifdef DEBUG
205                         cerr << "\t\t\tDEBUG taking major intervene action" << endl;
206                         cout << "==================================================
    =" << endl;
207                         cout << "                              NOTE" << endl;
208                         cout << "==================================================
    =" << endl << endl;
209     #endif
210                         cout << "You have been working for a particularly long" << e
    ndl;
211                         cout << "time.  A user's effectiveness deteriorates over" <<
     endl;
212                         cout << "time, so it is strongly encouraged that you take" <
    < endl;
213                         cout << "a break and return to the system later (however," <
    < endl;
214                         cout << "you may continue if desired).  The following" << en
    dl;
215                         cout << "summarizes your session so far:" << endl << endl;
216
217                         stats.print_summary();
218                         cout << endl;
219
220                         choice = userint.after_stats();
221                         if (choice == CC_try_again)
222                         {
223                             user_wants_to_continue = TRUE;
224                         }
225                         else
226                         {
227                             user_wants_to_continue = FALSE;
228                         }
229
230                     }
231                     else if ((exercise_count % MINOR_INTERVENE_MAX) == 0)
232                     {
233             //----------------------------------------------//
234             // If the user has reached a predetermined      //
235             // number of exercises, intervene to give the   //
236             // student an explicit opportunity to stop.     //
237             // This intervention is less stern than the     //
238             // MAJOR_INTERVENE action above.                //
239             //----------------------------------------------//
240     #ifdef DEBUG
241                         cerr << "\t\t\tDEBUG taking minor intervene action" << endl;
242     #endif
243                         cout << "==================================================
    =" << endl;
244                         cout << "                              NOTE" << endl;
245                         cout << "==================================================
    =" << endl << endl;
246                         cout << "When working with this type of system, it is easy"
    << endl;
247                         cout << "to lose track of time.  The following summarizes" <
    < endl;
248                         cout << "your session so far:" << endl << endl;
```

```
249
250                         stats.print_summary();
251                         cout << endl;
252
253                         choice = userint.after_stats();
254                         if (choice == CC_try_again)
255                         {
256                             user_wants_to_continue = TRUE;
257                         }
258                         else
259                         {
260                             user_wants_to_continue = FALSE;
261                         }
262
263                     }
264
265                     user_wants_to_continue = TRUE;
266
267                 } // end while
268
269                 break;
270
271             case SYSTEM:
272     #ifdef DEBUG
273                 cerr << "\t\t\tDEBUG taking action for system designer" << endl;
274     #endif
275             //-------------------------------------------------//
276             // System administrator features.                  //
277             //-------------------------------------------------//
278
279                 log.load();
280
281     //          test code append it back - file should double.
282     // .        log.save();
283
284                 for(;;)
285                 {
286                 choice = userint.sysadmin();
287                 switch(choice)
288                 {
289                     case CC_admin_write:
290                         cout << "The system administrator command to write the" << e
    ndl;
291                         cout << "current log entries to the default log file has" <<
     endl;
292                         cout << "not yet been enabled.  It will be done automaticall
    y" << endl;
293                         cout << "at the end of the session." << endl;
294                         break;
295                     case CC_admin_delete:
296                         cout << "The system administrator command to delete the" <<
    endl;
297                         cout << "current log file has not yet been enabled." << endl
    ;
298                         break;
299                     case CC_admin_save:
300                         cout << "The system administrator command to save the curren
    t" << endl;
301                         cout << "log file under a new name has not yet been enabled.
    " << endl;
302                         break;
303                     case CC_admin_diag_fail:
304                         log.view_at_leafs();
305                         break;
306                     case CC_admin_report_diag_fail:
307                         cout << "The system administrator command to generate a" <<
```



```
    endl;
308                         cout << "report of all system diagnosis failures from the "
    << endl;
309                         cout << "log file has not yet been enabled.  This informatio
    n" << endl;
310                         cout << "can be viewed manually by exiting the tutoring syst
    em" << endl;
311                         cout << "and looking at the file glt.log" << endl;
312                         break;
313                     case CC_admin_stats:
314                         log.view_ind_stats();
315                         break;
316                     case CC_admin_report_stats:
317                         cout << "The system administrator command to generate a" <<
    endl;
318                         cout << "report of all system session statistics from the "
    << endl;
319                         cout << "log file has not yet been enabled.  This informatio
    n" << endl;
320                         cout << "can be viewed manually by exiting the tutoring syst
    em" << endl;
321                         cout << "and looking at the file glt.log." << endl;
322                         break;
323                     case CC_admin_cum_stats:
324                         log.view_cum_stats();
325                         break;
326                     case CC_admin_report_cum_stats:
327                         cout << "The system administrator command to generate a" <<
    endl;
328                         cout << "report containing cumulative statistics from " << e
    ndl;
329                         cout << "individual statistics entries in the log has" << en
    dl;
330                         cout << "not yet been enabled.  This information can be" <<
    endl;
331                         cout << "derived manually by exiting the tutoring system" <<
     endl;
332                         cout << "and looking at the file glt.log." << endl;
333                         break;
334                     case CC_admin_report_full:
335                         cout << "The system administrator command to generate a" <<
    endl;
336                         cout << "full report from all log entries has not yet been"
    << endl;
337                         cout << "enabled.  This information can be viewed manually b
    y" << endl;
338                         cout << "exiting the tutoring system and viewing the file" <
    < endl;
339                         cout << "glt.log" << endl;
340                         break;
341                     case CC_NONE:
342                     case CC_try_again:
343                     case CC_hint:
344                     case CC_correct_answer:
345                     case CC_quit:
346                     case CC_exit:
347                     case CC_help:
348                     case CC_admin_find_next:
349                     default:
350                         cerr << "ERROR: bad admin command returned " << endl;
351                         break;
352                 }
353                 }
354                 break;
355             default:
356                 break;
```

```
357         }
358
359         exit_tutor();
360     }
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   statistics.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the run-time Statistics class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include <fstream.h>
22  #include <time.h>
23  #include "statistics.h"
24  #include "statistics_defs.h"
25
26          //----------- External data --------------------//
27
28  extern  time_t  session_start_time;
29
30          //------------------------------------------------//
31          // Constructor.                                   //
32          //------------------------------------------------//
33
34  Statistics::Statistics()
35  {
36  #ifdef DEBUG
37      cerr << "\t\t\t\t\tENTRY    Statistics CONSTRUCTOR" << endl;
38  #endif
39
40      exercise_count=0;
41      attempt_count=0;
42      correct_count=0;
43  };
44
45          //------------------------------------------------//
46          // Destructor.                                    //
47          //------------------------------------------------//
48
49  Statistics::~Statistics()
50  {
51  #ifdef DEBUG
52      cerr << "\t\t\t\t\tENTRY    Statistics DESTRUCTOR" << endl;
53  #endif
54
55  );
56
57          //------------------------------------------------//
58          // Member function increment_exercise_count().  //
59          //------------------------------------------------//
60
61  void Statistics::increment_exercise_count()
62  {
63  #ifdef DEBUG
64      cerr << "\t\t\t\t\tENTRY Statistics::increment_exercise_count" << endl;
65  #endif
66
```



```
67          //----------- Automatics - objects -------------//
68
69          //----------- Automatics - other ---------------//
70
71          //----------- Code begins ----------------------//
72
73
74      exercise_count++;
75  );
76
77          //------------------------------------------------//
78          // Member function increment_attempt_count().   //
79          //------------------------------------------------//
80
81  void Statistics::increment_attempt_count()
82  {
83  #ifdef DEBUG
84      cerr << "\t\t\t\t\tENTRY Statistics::increment_attempt_count" << endl;
85  #endif
86
87          //----------- Automatics - objects -------------//
88
89          //----------- Automatics - other ---------------//
90
91          //----------- Code begins ----------------------//
92
93
94      attempt_count++;
95  );
96
97          //------------------------------------------------//
98          // Member function increment_correct_count().   //
99          //------------------------------------------------//
100
101 void Statistics::increment_correct_count()
102 {
103 #ifdef DEBUG
104     cerr << "\t\t\t\t\tENTRY Statistics::increment_correct_count" << endl;
105 #endif
106
107         //----------- Automatics - objects -------------//
108
109         //----------- Automatics - other ---------------//
110
111         //----------- Code begins ----------------------//
112
113
114     correct_count++;
115 );
116
117         //------------------------------------------------//
118         // Member function print_summary().             //
119         //------------------------------------------------//
120
121 void Statistics::print_summary()
122 {
123 #ifdef DEBUG
124     cerr << "\t\t\t\t\tENTRY Statistics::print_summary" << endl;
125 #endif
126
127         //----------- Automatics - objects -------------//
128
129         //----------- Automatics - other ---------------//
130
131     time_t              session_current_time;
132     time_t              tloc;
```

```
133     double              duration;
134     unsigned long       int_val;
135     unsigned long       hours;
136     unsigned long       minutes;
137     unsigned long       seconds;
138
139         //----------- Code begins ----------------------//
140
141         //------------------------------------------------//
142         // Get the current time, and calculate session  //
143         // duration since start time.  Format and       //
144         // print duration.                              //
145         //------------------------------------------------//
146
147     session_current_time = time(&tloc);
148
149     duration = session_current_time - session_start_time;
150
151     if (duration >= MAX_SESSION_DURATION)       // 1 day
152     {
153         cerr << "ERROR - time out of range" << endl;
154     }
155     else
156     {
157         int_val = (unsigned long) duration;
158         hours = int_val/NUM_SECONDS_PER_HOUR;
159         int_val = int_val - (hours * NUM_SECONDS_PER_HOUR);
160         minutes = int_val/NUM_SECONDS_PER_MINUTE;
161         seconds = int_val-(minutes * NUM_SECONDS_PER_MINUTE);
162
163         cout << "\tThis session has been active for ";
164         if (hours < 1)
165         {
166             cout << minutes << " minutes, " << seconds << " seconds." << endl;
167         }
168         else
169         {
170             cout << hours << " hours, " << minutes << " minutes." << endl;
171         }
172     }
173         //------------------------------------------------//
174         // Summarize other information.                 //
175         //------------------------------------------------//
176
177     cout << "\tNumber of different phrases asked to translate - " << exercise_co
    unt << endl;
178     cout << "\tNumber of correct translations - " << correct_count << endl;
179     cout << "\tNumber of total responses - " << attempt_count << endl;
180
181 }
182
183         //------------------------------------------------//
184         // Member function print_to_file().             //
185         //------------------------------------------------//
186
187 void Statistics::print_to_file(ofstream * stats_stream)
188 {
189 #ifdef DEBUG
190     cerr << "\t\t\t\t\tENTRY Statistics::print_to_file" << endl;
191 #endif
192
193         //----------- Automatics - objects -------------//
194
195         //----------- Automatics - other ---------------//
196
197     time_t              session_current_time;
```



```
198     time_t              tloc;
199     double              duration;
200 //    unsigned long     int_val;
201 //    unsigned long     hours;
202 //    unsigned long     minutes;
203 //    unsigned long     seconds;
204
205         //----------- Code begins ----------------------//
206
207         //------------------------------------------------//
208         // Essentially the same functionality as        //
209         // print_summary(), except that it prints to    //
210         // a specified statistics file instead of to    //
211         // the screen.                                  //
212         //------------------------------------------------//
213
214     session_current_time = time(&tloc);
215
216     duration = session_current_time - session_start_time;
217
218     if (duration >= MAX_SESSION_DURATION)       // 1 day
219     {
220         cerr << "ERROR - time out of range" << endl;
221     }
222 //    else
223 //    {
224 //      int_val = (unsigned long) duration;
225 //      hours = int_val/NUM_SECONDS_PER_HOUR;
226 //      int_val = int_val - (hours * NUM_SECONDS_PER_HOUR);
227 //      minutes = int_val/NUM_SECONDS_PER_MINUTE;
228 //      seconds = int_val-(minutes * NUM_SECONDS_PER_MINUTE);
229 //
230 //      *stats_stream << "\tThis session has been active for ";
231 //      if (hours < 1)
232 //      {
233 //          *stats_stream << minutes << " minutes, " << seconds << " seconds." <
    < endl;
234 //      }
235 //      else
236 //      {
237 //          *stats_stream << hours << " hours, " << minutes << " minutes." << en
    dl;
238 //      }
239 //    }
240 //    *stats_stream << "\tNumber of different phrases asked to translate - " <<
    exercise_count << endl;
241 //    *stats_stream << "\tNumber of correct translations - " << correct_count <<
     endl;
242 //    *stats_stream << "\tNumber of total responses - " << attempt_count << endl
    ;
243
244     *stats_stream << duration << endl;
245     *stats_stream << exercise_count << endl;
246     *stats_stream << correct_count << endl;
247     *stats_stream << attempt_count << endl;
248
249 };
250
251         //------------------------------------------------//
252         // Member function read().                      //
253         //------------------------------------------------//
254
255 void Statistics::read(ifstream * stats_stream)
256 {
257 #ifdef DEBUG
258     cerr << "\t\t\t\t\tENTRY Statistics::read" << endl;
```

```
259  #endif
260
261      char ch;
262
263      *stats_stream >> prev_duration;
264      stats_stream->get(ch);
265      *stats_stream >> exercise_count;
266      stats_stream->get(ch);
267      *stats_stream >> correct_count;
268      stats_stream->get(ch);
269      *stats_stream >> attempt_count;
270      stats_stream->get(ch);
271
272  );
273
274          //-------------------------------------------------//
275          // Member function display().                      //
276          //-------------------------------------------------//
277
278  void Statistics::display()
279  {
280  #ifdef DEBUG
281      cerr << "\t\t\t\t\tENTRY Statistics::display" << endl;
282  #endif
283
284          //----------- Automatics - objects -------------//
285
286          //----------- Automatics - other ---------------//
287
288      unsigned long       int_val;
289      unsigned long       hours;
290      unsigned long       minutes;
291      unsigned long       seconds;
292
293          //----------- Code begins ----------------------//
294
295          int_val = (unsigned long) prev_duration;
296          hours = int_val/NUM_SECONDS_PER_HOUR;
297          int_val = int_val - (hours * NUM_SECONDS_PER_HOUR);
298          minutes = int_val/NUM_SECONDS_PER_MINUTE;
299          seconds = int_val-(minutes * NUM_SECONDS_PER_MINUTE);
300
301          cout << "\tThis session lasted ";
302          if (hours < 1)
303          {
304              cout << minutes << " minutes, " << seconds << " seconds." << endl;
305          }
306          else
307          {
308              cout << hours << " hours, " << minutes << " minutes." << endl;
309          }
310
311          //-------------------------------------------------//
312          // Summarize other information.                    //
313          //-------------------------------------------------//
314
315      cout << "\tNumber of different phrases asked to translate - " << exercise_co
     unt << endl;
316      cout << "\tNumber of correct translations - " << correct_count << endl;
317      cout << "\tNumber of total responses - " << attempt_count << endl;
318
319  );
320
321
322          //-------------------------------------------------//
323          // Member function get_exercise_count().           //
```



```
324          //-------------------------------------------------//
325
326  int Statistics::get_exercise_count()
327  {
328  #ifdef DEBUG
329      cerr << "\t\t\t\t\tENTRY Statistics::get_exercise_count" << endl;
330  #endif
331
332      return(exercise_count);
333
334  );
335
336          //-------------------------------------------------//
337          // Member function get_attempt_count().            //
338          //-------------------------------------------------//
339
340  int Statistics::get_attempt_count()
341  {
342  #ifdef DEBUG
343      cerr << "\t\t\t\t\tENTRY Statistics::get_attempt_count" << endl;
344  #endif
345
346      return(attempt_count);
347
348  );
349
350          //-------------------------------------------------//
351          // Member function get_correct_count().            //
352          //-------------------------------------------------//
353
354  int Statistics::get_correct_count()
355  {
356  #ifdef DEBUG
357      cerr << "\t\t\t\t\tENTRY Statistics::get_correct_count" << endl;
358  #endif
359
360      return(correct_count);
361
362  );
363          //-------------------------------------------------//
364          // Member function get_prev_duration().            //
365          //-------------------------------------------------//
366
367  int Statistics::get_prev_duration()
368  {
369  #ifdef DEBUG
370      cerr << "\t\t\t\t\tENTRY Statistics::get_prev_duration" << endl;
371  #endif
372
373      return(prev_duration);
374
375  );
```

```
1     //-----------------------------------------------------------------------------
2     //
3     //  Filename:   sys_exit_tutor.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This function is called to terminate the current interactive
8     //              session, and save any run-time gathered data.
9     //
10    //  Notes:      The theory and design of this system have been documented
11    //              in the accompanying thesis report.
12    //
13    //  Revisions   By              Reason
14    //  ---------   --              ------
15    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
16    //
17    //-----------------------------------------------------------------------------
18
19            //------------ Included files --------------------//
20
21    #include <iostream.h>
22    #include <stdlib.h>
23    #include <time.h>
24    #include "stdtype.h"
25    #include "log.h"
26    #include "statistics.h"
27    #include "userlist.h"
28
29            //------------ External Data ---------------------//
30
31    extern  Log             log;
32    extern  Statistics      stats;
33    extern  time_t          session_start_time;
34    extern  UserList        users;
35
36            //------------ Calling Syntax --------------------//
37
38    void sys_exit_tutor()
39    {
40
41            //------------ Automatics - objects --------------//
42
43            //------------ Automatics - other ----------------//
44
45        //time_t            session_end_time;
46        //time_t            tloc;
47        //double            duration;
48        //unsigned long     int_val;
49        //unsigned long     hours;
50        //unsigned long     minutes;
51        //unsigned long     seconds;
52
53            //------------ Code begins -----------------------//
54
55    #ifdef DEBUG
56        cerr << "\t\t\t\t\tENTRY exit_tutor()" << endl;
57    #endif
58
59            //-------------------------------------------//
60            // Say goodbye...                            //
61            //-------------------------------------------//
62
63        cout << endl;
64        cout << "\tYour session with the German Language Tutor is complete." << endl
      ;
65        cout << endl;
```



```
66        exit(0);
67    }
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   translationtree.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the TranslationTree class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include "translationtree.h"
22  #include "patternnode.h"
23  #include "statistics.h"
24  #include "word_and_phrase_defs.h"
25
26          //----------- External data --------------------//
27
28  extern Statistics stats;
29
30          //------------------------------------------------//
31          // Constructor.                                   //
32          //------------------------------------------------//
33
34  TranslationTree::TranslationTree(BasePhrase (*in_bpt) (BasePhrase &),
35                                   PatternNode * in_root,
36                                   PhraseDescriptor * in_fixed_desc)
37  {
38  #ifdef DEBUG
39      cerr << "\t\t\t\t\tENTRY    TranslationTree CONSTRUCTOR" << endl;
40  #endif
41
42      in_phrase_translator = in_bpt;
43      root_pattern = in_root;
44      fixed_phrase_desc = in_fixed_desc;
45  );
46
47          //------------------------------------------------//
48          // Destructor.                                    //
49          //------------------------------------------------//
50
51  TranslationTree::~TranslationTree()
52  {
53  #ifdef DEBUG
54      cerr << "\t\t\t\t\tENTRY    TranslationTree DESTRUCTOR" << endl;
55  #endif
56
57  };
58
59          //------------------------------------------------//
60          // Member function check_response().              //
61          //------------------------------------------------//
62
63  boolean TranslationTree::check_response(BasePhrase in_phrase)
64  {
65  #ifdef DEBUG
66      cerr << "\t\t\t\t\tENTRY TranslationTree::check_response" << endl;
```



```
67  #endif
68
69          //----------- Automatics - objects -------------//
70
71      BasePhrase translated_bp = in_phrase_translator(in_phrase);
72      Phrase expected_response(in_phrase.get_size());
73      expected_response = fixed_phrase_desc->build_phrase(translated_bp);
74      Pattern correct_pattern(expected_response);
75
76          //----------- Automatics - other ---------------//
77
78      boolean user_wants_to_try_again = TRUE;
79      Phrase user_response(MAX_PHRASE_LENGTH);
80
81          //----------- Code begins ----------------------//
82
83  #ifdef DEBUG
84      cerr << "\t\t\tDEBUG Expected response (translation phrase) is:" << endl;
85      expected_response.debug_print();
86  #endif
87
88          //------------------------------------------------//
89          // Read the user's translation attempt.           //
90          // (Increment attempt count in Statistics).       //
91          //------------------------------------------------//
92
93      user_response.read();
94      stats.increment_attempt_count();
95
96  #ifdef DEBUG
97      cerr << "\t\t\tDEBUG Actual user response (input) is:" << endl;
98      user_response.debug_print();
99  #endif
100
101         //------------------------------------------------//
102         // Attempt to match this against the expected     //
103         // response.  If it matches, tell the user,       //
104         // tally the correct answer in the Statistics,    //
105         // and set the return value to indicate that      //
106         // it is not necessary to try again on this       //
107         // exercise.                                      //
108         //------------------------------------------------//
109
110     if (correct_pattern.match(user_response))
111     {
112         user_wants_to_try_again = FALSE;
113 #ifdef DEBUG
114         cerr << endl;
115         cerr << "\tYes! That is the expected response." << endl;
116         cerr << endl;
117 #endif
118         cout << endl;
119         cout << "\tYes! That is the expected response." << endl;
120         cout << endl;
121         stats.increment_correct_count();
122     )
123     else
124     {
125         //------------------------------------------------//
126         // If the expected response was not found,        //
127         // pass the user's response to the root           //
128         // PatternNode of the diagnosis tree.             //
129         //------------------------------------------------//
130
131         user_wants_to_try_again = root_pattern->process_response(
132                                             expected_response, user_response);
```

```
133             if (user_wants_to_try_again == FALSE)
134             (
135             //---------------------------------------------//
136             // If the user chooses not to try again after   //
137             // diagnosis, print the expected response.      //
138             //---------------------------------------------//
139
140                 cout << endl;
141                 cout << "\tThe system was expecting:" << endl << endl;
142                 cout << "\t\t";
143                 expected_response.print();
144                 cout << endl << endl;
145             )
146         )
147         return(user_wants_to_try_again);
148     }
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   user.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the User class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //              The get_string() member function returns a pointer to the
13  //              internal string.  This should probably be improved, but for
14  //              the time being, it is important for the caller to realize
15  //              that the pointer can only be used while this object is in
16  //              scope.
17  //
18  //  Revisions   By              Reason
19  //  ---------   --              ------
20  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
21  //
22  //-------------------------------------------------------------------------------
23
24          //----------- Included files -------------------//
25
26  #include <fstream.h>
27  #include <iostream.h>
28  #include <string.h>
29  #include "user.h"
30  #include "user_defs.h"
31
32          //-----------------------------------------------//
33          // Constructor.                                  //
34          //-----------------------------------------------//
35
36  User::User()
37  {
38  #ifdef DEBUG
39      cerr << "\t\t\t\t\tENTRY    User CONSTRUCTOR" << endl;
40  #endif
41
42      type = NO_USERTYPE;
43      last_known_level = 0;
44      login_name = new char[MAX_LOGIN_NAME_STRING];
45      full_name = new char[MAX_FULL_NAME_STRING];
46  };
47
48          //-----------------------------------------------//
49          // Destructor.                                   //
50          //-----------------------------------------------//
51
52  User::~User()
53  {
54  #ifdef DEBUG
55      cerr << "\t\t\t\t\tENTRY    User DESTRUCTOR" << endl;
56  #endif
57
58      delete [] login_name;
59      delete [] full_name;
60  };
61
62          //-----------------------------------------------//
63          // Member function match_user().                 //
64          //-----------------------------------------------//
65
66  Usertype  User::match_user(char * in_string)
```



```
67  {
68  #ifdef DEBUG
69      cerr << "\t\t\t\t\tENTRY User::match_user" << endl;
70  #endif
71
72          //----------- Automatics - objects -------------//
73
74          //----------- Automatics - other ---------------//
75
76          //----------- Code begins ----------------------//
77
78          //-----------------------------------------------//
79          // If the input string matches the current       //
80          // User object's username string - return        //
81          // this user's type.                             //
82          //-----------------------------------------------//
83
84      if (strcmp(in_string,login_name) == 0)
85      {
86          return(type);
87      }
88      else
89      {
90          return(NO_USERTYPE);
91      }
92  }
93
94          //-----------------------------------------------//
95          // Member function get_string().                 //
96          //-----------------------------------------------//
97
98  char * User::get_string()
99  {
100 #ifdef DEBUG
101     cerr << "\t\t\t\t\tENTRY User::get_string" << endl;
102 #endif
103
104         //----------- Automatics - objects -------------//
105
106         //----------- Automatics - other ---------------//
107
108         //----------- Code begins ----------------------//
109
110
111     return(login_name);
112 }
113
114         //-----------------------------------------------//
115         // Member function read().                       //
116         //-----------------------------------------------//
117
118 boolean User::read(ifstream * user_stream)
119 {
120 #ifdef DEBUG
121     cerr << "\t\t\t\t\tENTRY User::read" << endl;
122 #endif
123
124     int user_type;
125
126         //-----------------------------------------------//
127         // Read all internal data from the specified     //
128         // user data file.  If a valid user object       //
129         // is found, return TRUE.                        //
130         //-----------------------------------------------//
131
132     if (*user_stream >> user_type)
```

```
133         {
134             type = (Usertype) user_type;
135     #ifdef DEBUG
136         cerr << "\t\t\tDEBUG user type read is " << (int) type << endl;
137     #endif
138             *user_stream >> last_known_level;
139     #ifdef DEBUG
140         cerr << "\t\t\tDEBUG last knowledge level read is " << last_known_level << e
        ndl;
141     #endif
142             *user_stream >> login_name;
143     #ifdef DEBUG
144         cerr << "\t\t\tDEBUG login name read is " << login_name << endl;
145     #endif
146             *user_stream >> full_name;
147     #ifdef DEBUG
148         cerr << "\t\t\tDEBUG full name read is " << full_name << endl;
149     #endif
150             return(TRUE);
151         }
152         else
153         {
154     #ifdef DEBUG
155         cerr << "\t\t\tDEBUG User::read() found end-of-file" << endl;
156     #endif
157             return(FALSE);
158         }
159
160     );
161
162             //-----------------------------------------------//
163             // Member function set_current_level().          //
164             //-----------------------------------------------//
165
166     boolean User::set_current_level(int in_level)
167     {
168     #ifdef DEBUG
169         cerr << "\t\t\t\tENTRY User::set_current_level" << endl;
170     #endif
171
172         boolean change_made;
173
174             //-----------------------------------------------//
175             // If the input knowledge level is different     //
176             // from the previously known value, set the      //
177             // new value, and flag that a change has been    //
178             // made so that the user data file can be        //
179             // updated.                                      //
180             //-----------------------------------------------//
181
182         if( last_known_level != in_level )
183         {
184     #ifdef DEBUG
185         cerr << "\t\t\tDEBUG current level " << last_known_level << " new level " <<
        in_level << endl;
186         cerr << "\t\t\tDEBUG    returning TRUE" << endl;
187     #endif
188             last_known_level = in_level;
189             change_made = TRUE;
190         }
191         else
192         {
193     #ifdef DEBUG
194         cerr << "\t\t\tDEBUG current level " << last_known_level << " new level " <<
        in_level << endl;
195         cerr << "\t\t\tDEBUG    returning FALSE" << endl;
```



```
196     #endif
197             change_made = FALSE;
198         }
199
200         return(change_made);
201
202     };
203
204             //-----------------------------------------------//
205             // Member function get_current_level().          //
206             //-----------------------------------------------//
207
208     int User::get_current_level()
209     {
210     #ifdef DEBUG
211         cerr << "\t\t\t\tENTRY User::get_current_level" << endl;
212     #endif
213
214         return( last_known_level );
215
216     };
217
218             //-----------------------------------------------//
219             // Member function write().                      //
220             //-----------------------------------------------//
221
222     void User::write(ofstream * user_stream)
223     {
224     #ifdef DEBUG
225         cerr << "\t\t\t\tENTRY User::write" << endl;
226     #endif
227
228             //-----------------------------------------------//
229             // Write the current User object's data to the   //
230             // specified user data file.                     //
231             //-----------------------------------------------//
232
233         *user_stream << (int) type << endl;
234         *user_stream << last_known_level << endl;
235         *user_stream << login_name << endl;
236         *user_stream << full_name << endl;
237
238     };
239
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   userinterface.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the UserInterface class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //              This object should really be modularized a better -
13  //              pulling out the menu printing, etc. from the individual
14  //              member functions.  As it was, new member functions were
15  //              added in a rapid-prototype fashion by just copying the
16  //              previous and modifying it as needed.  It should have been
17  //              cleaned up, but as time grew short, the user interface
18  //              fell by the wayside.
19  //
20  //  Revisions   By              Reason
21  //  ---------   --              ------
22  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
23  //
24  //-------------------------------------------------------------------------------
25
26          //----------- Included files -------------------//
27
28  #include <iostream.h>
29  #include <stdlib.h>
30  #include "stdtype.h"
31  #include "userinterface.h"
32
33          //----------- External functions ---------------//
34
35  extern void exit_tutor();
36  extern void sys_exit_tutor();
37
38          //------------------------------------------------//
39          // Constructor.                                   //
40          //------------------------------------------------//
41
42  UserInterface::UserInterface()
43  {
44  #ifdef DEBUG
45      cerr << "\t\t\t\t\tENTRY    UserInterface CONSTRUCTOR" << endl;
46  #endif
47
48  );
49
50          //------------------------------------------------//
51          // Destructor.                                    //
52          //------------------------------------------------//
53
54  UserInterface::~UserInterface()
55  {
56  #ifdef DEBUG
57      cerr << "\t\t\t\t\tENTRY    UserInterface DESTRUCTOR" << endl;
58  #endif
59
60  );
61
62          //------------------------------------------------//
63          // Member function wrong_answer().                //
64          //------------------------------------------------//
65
66  ControlCommand UserInterface::wrong_answer()
```



```
67  {
68  #ifdef DEBUG
69      cerr << "\t\t\t\t\tENTRY UserInterface::wrong_answer" << endl;
70  #endif
71
72          //----------- Automatics - objects -------------//
73
74          //----------- Automatics - other ---------------//
75
76      boolean             not_satisfied = TRUE;
77      ControlCommand      choice;
78      int                 int_choice;
79      char                in_char;
80
81          //----------- Code begins ----------------------//
82
83          //------------------------------------------------//
84          // Print message indicating that the expected    //
85          // response was not input.                        //
86          //------------------------------------------------//
87
88      cout << endl;
89      cout << "\tThat is not the expected translation.  Enter the number" << endl;
90      cout << "\tindicating the action you would like to take:" << endl;
91      cout << endl;
92
93          //------------------------------------------------//
94          // Print menu of options and read the user's     //
95          // choice.                                        //
96          //------------------------------------------------//
97
98      while (not_satisfied)
99      {
100
101         cout << "\t" << CC_try_again << " - Try again." << endl;
102         cout << "\t" << CC_hint << " - Have the system give you a hint." << endl
    ;
103         cout << "\t" << CC_correct_answer << " - Have the system give you its an
    swer." << endl;
104         cout << "\t" << CC_quit << " - Quit this exercise and go on to next." <<
     endl;
105         cout << "\t" << CC_exit << " - Exit the tutoring system." << endl;
106         cout << "\t" << CC_help << " - Help." << endl;
107         cout << endl;
108
109         cout << "Choice? ";
110         cin.get(in_char);
111         if (in_char == '?')
112         {
113             choice = CC_help;
114         }
115         else
116         {
117             cin.putback(in_char);
118             cin >> int_choice;
119             choice = (ControlCommand) int_choice;
120         }
121
122 #ifdef DEBUG
123         cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
124         cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
125 #endif
126         //------------------------------------------------//
127         // Parse choices - some are handled right here,  //
128         // others are returned to the caller.            //
129         //------------------------------------------------//
```

```
130           switch(choice)
131           {
132               case CC_try_again:
133               case CC_hint:
134               case CC_correct_answer:
135                   not_satisfied = FALSE;
136                   break;
137               case CC_quit:
138                   choice = CC_correct_answer;
139                   not_satisfied = FALSE;
140                   break;
141               case CC_exit:
142                   not_satisfied = FALSE;
143                   exit_tutor();
144                   break;
145               case CC_help:
146                   cout << endl;
147                   cout << "\tThis is a menu of the options that you have at this p
    oint." << endl;
148                   cout << "\tType the number of the option which you wish to take.
    " << endl;
149                   cout << "\tYour choices at this point are to try answering the e
    xercise" << endl;
150                   cout << "\tagain, have the system give you a hint about what was
     wrong," << endl;
151                   cout << "\thave the system give you the correct answer, go on to
     a new" << endl;
152                   cout << "\texercise, exit the system altogether, or view this he
    lp message." << endl;
153                   cout << endl;
154                   not_satisfied = TRUE;
155                   break;
156               case CC_NONE:
157               case CC_admin_write:
158               case CC_admin_delete:
159               case CC_admin_save:
160               case CC_admin_diag_fail:
161               case CC_admin_report_diag_fail:
162               case CC_admin_stats:
163               case CC_admin_report_stats:
164               case CC_admin_cum_stats:
165               case CC_admin_report_cum_stats:
166               case CC_admin_report_full:
167               case CC_admin_find_next:
168               default:
169                   cout << endl;
170                   cout << "\tInvalid choice, please choose again:" << endl;
171                   not_satisfied = FALSE;
172                   break;
173           }
174       }
175       return(choice);
176   };
177
178           //-----------------------------------------------//
179           // Member function after_hint().                 //
180           //-----------------------------------------------//
181
182   ControlCommand UserInterface::after_hint(Hint * in_hint)
183   {
184   #ifdef DEBUG
185       cerr << "\t\t\t\t\tENTRY UserInterface::after_hint" << endl;
186   #endif
187
188           //------------ Automatics - objects -------------//
189
```



```
190           //------------ Automatics - other ---------------//
191
192       boolean             not_satisfied = TRUE;
193       ControlCommand      choice;
194       int                 int_choice;
195       char                in_char;
196
197           //------------ Code begins ----------------------//
198
199           //-----------------------------------------------//
200           // Request that the input Hint print itself.     //
201           //-----------------------------------------------//
202
203       in_hint->print();
204
205           //-----------------------------------------------//
206           // Print menu of options and read the user's     //
207           // choice.                                       //
208           //-----------------------------------------------//
209
210       while (not_satisfied)
211       {
212
213           cout << "\t" << CC_try_again << " - Try again." << endl;
214           cout << "\t" << CC_hint << " - Have the system give you another hint." <
    < endl;
215           cout << "\t" << CC_correct_answer << " - Have the system give you its an
    swer." << endl;
216           cout << "\t" << CC_quit << " - Quit this exercise and go on to next." <<
     endl;
217           cout << "\t" << CC_exit << " - Exit the tutoring system." << endl;
218           cout << "\t" << CC_help << " - Help." << endl;
219           cout << endl;
220
221           cout << "Choice? ";
222           cin.get(in_char);
223           if (in_char == '?')
224           {
225               choice = CC_help;
226           }
227           else
228           {
229               cin.putback(in_char);
230               cin >> int_choice;
231               choice = (ControlCommand) int_choice;
232           }
233
234   #ifdef DEBUG
235           cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
236           cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
237   #endif
238           //-----------------------------------------------//
239           // Parse choices - some are handled right here, //
240           // others are returned to the caller.            //
241           //-----------------------------------------------//
242
243           switch(choice)
244           {
245               case CC_try_again:
246               case CC_hint:
247               case CC_correct_answer:
248                   not_satisfied = FALSE;
249                   break;
250               case CC_quit:
251                   choice = CC_correct_answer;
252                   not_satisfied = FALSE;
```

```
253                 break;
254             case CC_exit:
255                 not_satisfied = FALSE;
256                 exit_tutor();
257                 break;
258             case CC_help:
259                 cout << endl;
260                 cout << "\tThis is a menu of the options that you have at this p
    oint." << endl;
261                 cout << "\tType the number of the option which you wish to take.
    " << endl;
262                 cout << "\tYour choices at this point are to try answering the e
    xercise" << endl;
263                 cout << "\tagain, have the system give you another hint about wh
    at was wrong," << endl;
264                 cout << "\thave the system give you the correct answer, go on to
     a new" << endl;
265                 cout << "\texercise, exit the system altogether, or view this he
    lp message." << endl;
266                 cout << endl;
267                 not_satisfied = TRUE;
268                 break;
269             case CC_NONE:
270             case CC_admin_write:
271             case CC_admin_delete:
272             case CC_admin_save:
273             case CC_admin_diag_fail:
274             case CC_admin_report_diag_fail:
275             case CC_admin_stats:
276             case CC_admin_report_stats:
277             case CC_admin_cum_stats:
278             case CC_admin_report_cum_stats:
279             case CC_admin_report_full:
280             case CC_admin_find_next:
281             default:
282                 cout << endl;
283                 cout << "\tInvalid choice, please choose again:" << endl;
284                 not_satisfied = FALSE;
285                 break;
286         }
287     }
288     return(choice);
289 };
290
291         //-------------------------------------------------//
292         // Member function cant_diagnose().                //
293         //-------------------------------------------------//
294
295 ControlCommand UserInterface::cant_diagnose()
296 {
297 #ifdef DEBUG
298     cerr << "\t\t\t\t\tENTRY UserInterface::cant_diagnose" << endl;
299 #endif
300
301         //----------- Automatics - objects -------------//
302
303         //----------- Automatics - other ---------------//
304
305     boolean             not_satisfied = TRUE;
306     ControlCommand      choice;
307     int                 int_choice;
308     char                in_char;
309
310         //----------- Code begins ----------------------//
311
312         //-------------------------------------------------//
```



```
313         // Print message indicating that the system     //
314         // could not match the expected response at all.//
315         //-------------------------------------------------//
316
317     cout << endl;
318     cout << "\tThat response is incorrect, but the system is unable" << endl;
319     cout << "\tto isolate the specific problem." << endl;
320     cout << endl;
321
322         //-------------------------------------------------//
323         // Print menu of options and read the user's    //
324         // choice.                                      //
325         //-------------------------------------------------//
326
327
328     while (not_satisfied)
329     {
330
331         cout << "\t" << CC_try_again << " - Try again." << endl;
332         cout << "\t" << CC_correct_answer << " - Have the system give you its an
    swer." << endl;
333         cout << "\t" << CC_quit << " - Quit this exercise and go on to next." <<
     endl;
334         cout << "\t" << CC_exit << " - Exit the tutoring system." << endl;
335         cout << "\t" << CC_help << " - Help." << endl;
336         cout << endl;
337
338         cout << "Choice? ";
339         cin.get(in_char);
340         if (in_char == '?')
341         {
342             choice = CC_help;
343         }
344         else
345         {
346             cin.putback(in_char);
347             cin >> int_choice;
348             choice = (ControlCommand) int_choice;
349         }
350
351 #ifdef DEBUG
352         cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
353         cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
354 #endif
355
356         //-------------------------------------------------//
357         // Parse choices - some are handled right here, //
358         // others are returned to the caller.           //
359         //-------------------------------------------------//
360
361         switch(choice)
362         {
363             case CC_try_again:
364             case CC_correct_answer:
365                 not_satisfied = FALSE;
366                 break;
367             case CC_quit:
368                 choice = CC_correct_answer;
369                 not_satisfied = FALSE;
370                 break;
371             case CC_exit:
372                 not_satisfied = FALSE;
373                 exit_tutor();
374                 break;
375             case CC_help:
376                 cout << endl;
```

```
377                 cout << "\tThis is a menu of the options that you have at this p
    oint." << endl;
378                 cout << "\tType the number of the option which you wish to take.
    " << endl;
379                 cout << "\tYour choices at this point are to try answering the e
    xercise" << endl;
380                 cout << "\tagain, have the system give you a hint about what was
     wrong," << endl;
381                 cout << "\thave the system give you the correct answer, go on to
     a new" << endl;
382                 cout << "\texercise, exit the system altogether, or view this he
    lp message." << endl;
383                 cout << endl;
384                 not_satisfied = TRUE;
385                 break;
386             case CC_NONE:
387             case CC_hint:
388             case CC_admin_write:
389             case CC_admin_delete:
390             case CC_admin_save:
391             case CC_admin_diag_fail:
392             case CC_admin_report_diag_fail:
393             case CC_admin_stats:
394             case CC_admin_report_stats:
395             case CC_admin_cum_stats:
396             case CC_admin_report_cum_stats:
397             case CC_admin_report_full:
398             case CC_admin_find_next:
399             default:
400                 cout << endl;
401                 cout << "\tInvalid choice, please choose again:" << endl;
402                 not_satisfied = FALSE;
403                 break;
404         )
405     )
406     return(choice);
407 );
408
409         //------------------------------------------------//
410         // Member function cant_diagnose_further().      //
411         //------------------------------------------------//
412
413 ControlCommand UserInterface::cant_diagnose_further()
414 {
415 #ifdef DEBUG
416     cerr << "\t\t\t\t\tENTRY UserInterface::cant_diagnose_further" << endl;
417 #endif
418
419         //----------- Automatics - objects -------------//
420
421         //----------- Automatics - other ---------------//
422
423     boolean             not_satisfied = TRUE;
424     ControlCommand      choice;
425     int                 int_choice;
426     char                in_char;
427
428         //----------- Code begins ----------------------//
429
430         //------------------------------------------------//
431         // Print message indicating that the system      //
432         // cannot isolate the problem any further        //
433         // than it already has.                          //
434         //------------------------------------------------//
435
436
```



```
437     cout << endl;
438     cout << "\tThe system cannot diagnose this any further." << endl;
439     cout << endl;
440
441         //------------------------------------------------//
442         // Print menu of options and read the user's     //
443         // choice.                                       //
444         //------------------------------------------------//
445
446
447     while (not_satisfied)
448     {
449
450         cout << "\t" << CC_try_again << " - Try again." << endl;
451         cout << "\t" << CC_correct_answer << " - Have the system give you its an
    swer." << endl;
452         cout << "\t" << CC_quit << " - Quit this exercise and go on to next." <<
     endl;
453         cout << "\t" << CC_exit << " - Exit the tutoring system." << endl;
454         cout << "\t" << CC_help << " - Help." << endl;
455         cout << endl;
456
457         cout << "Choice? ";
458         cin.get(in_char);
459         if (in_char == '?')
460         {
461             choice = CC_help;
462         }
463         else
464         {
465             cin.putback(in_char);
466             cin >> int_choice;
467             choice = (ControlCommand) int_choice;
468         }
469
470 #ifdef DEBUG
471         cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
472         cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
473 #endif
474
475         //------------------------------------------------//
476         // Parse choices - some are handled right here, //
477         // others are returned to the caller.           //
478         //------------------------------------------------//
479
480         switch(choice)
481         {
482             case CC_try_again:
483             case CC_correct_answer:
484                 not_satisfied = FALSE;
485                 break;
486             case CC_quit:
487                 choice = CC_correct_answer;
488                 not_satisfied = FALSE;
489                 break;
490             case CC_exit:
491                 not_satisfied = FALSE;
492                 exit_tutor();
493                 break;
494             case CC_help:
495                 cout << endl;
496                 cout << "\tThis is a menu of the options that you have at this p
    oint." << endl;
497                 cout << "\tType the number of the option which you wish to take.
    " << endl;
498                 cout << "\tYour choices at this point are to try answering the e
```

```
    xercise" << endl;
499                 cout << "\tagain, have the system give you a hint about what was
     wrong," << endl;
500                 cout << "\thave the system give you the correct answer, go on to
     a new" << endl;
501                 cout << "\texercise, exit the system altogether, or view this he
    lp message." << endl;
502                 cout << endl;
503                 not_satisfied = TRUE;
504                 break;
505             case CC_NONE:
506             case CC_hint:
507             case CC_admin_write:
508             case CC_admin_delete:
509             case CC_admin_save:
510             case CC_admin_diag_fail:
511             case CC_admin_report_diag_fail:
512             case CC_admin_stats:
513             case CC_admin_report_stats:
514             case CC_admin_cum_stats:
515             case CC_admin_report_cum_stats:
516             case CC_admin_report_full:
517             case CC_admin_find_next:
518             default:
519                 cout << endl;
520                 cout << "\tInvalid choice, please choose again:" << endl;
521                 not_satisfied = FALSE;
522                 break;
523         }
524     }
525     return(choice);
526 );
527
528         //-------------------------------------------------//
529         // Member function after_stats().                  //
530         //-------------------------------------------------//
531
532 ControlCommand UserInterface::after_stats()
533 {
534 #ifdef DEBUG
535     cerr << "\t\t\t\t\tENTRY UserInterface::after_stats" << endl;
536 #endif
537
538         //----------- Automatics - objects -------------//
539
540         //----------- Automatics - other ---------------//
541
542     boolean             not_satisfied = TRUE;
543     ControlCommand      choice;
544     int                 int_choice;
545     char                in_char;
546
547         //----------- Code begins ----------------------//
548
549
550     cout << endl;
551
552         //-------------------------------------------------//
553         // Print menu of options and read the user's       //
554         // choice.                                         //
555         //-------------------------------------------------//
556
557
558     while (not_satisfied)
559     {
560
```



```
561         cout << "\t" << CC_try_again << " - Continue (more exercises)." << endl;
562         cout << "\t" << CC_exit << " - Exit the tutoring system." << endl;
563         cout << "\t" << CC_help << " - Help." << endl;
564         cout << endl;
565
566         cout << "Choice? ";
567         cin.get(in_char);
568         if (in_char == '?')
569         {
570             choice = CC_help;
571         }
572         else
573         {
574             cin.putback(in_char);
575             cin >> int_choice;
576             choice = (ControlCommand) int_choice;
577         }
578
579 #ifdef DEBUG
580         cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
581         cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
582 #endif
583
584         //-------------------------------------------------//
585         // Parse choices - some are handled right here, //
586         // others are returned to the caller.              //
587         //-------------------------------------------------//
588
589         switch(choice)
590         {
591             case CC_try_again:
592                 not_satisfied = FALSE;
593                 break;
594             case CC_exit:
595                 not_satisfied = FALSE;
596                 exit_tutor();
597                 break;
598             case CC_help:
599                 cout << endl;
600                 cout << "\tThis is a menu of the options that you have at this p
    oint." << endl;
601                 cout << "\tType the number of the option which you wish to take.
    " << endl;
602                 cout << "\tYour choices at this point are to continue with more"
     << endl;
603                 cout << "\texercises, or to exit the system." << endl;
604                 cout << endl;
605                 not_satisfied = TRUE;
606                 break;
607             case CC_NONE:
608             case CC_hint:
609             case CC_correct_answer:
610             case CC_quit:
611             case CC_admin_write:
612             case CC_admin_delete:
613             case CC_admin_save:
614             case CC_admin_diag_fail:
615             case CC_admin_report_diag_fail:
616             case CC_admin_stats:
617             case CC_admin_report_stats:
618             case CC_admin_cum_stats:
619             case CC_admin_report_cum_stats:
620             case CC_admin_report_full:
621             case CC_admin_find_next:
622             default:
623                 cout << endl;
```

```
624                 cout << "\tInvalid choice, please choose again:" << endl;
625                 not_satisfied = FALSE;
626                 break;
627             }
628         }
629         return(choice);
630     };
631
632             //-----------------------------------------------//
633             // Member function sysadmin().                   //
634             //-----------------------------------------------//
635
636     ControlCommand UserInterface::sysadmin()
637     {
638     #ifdef DEBUG
639         cerr << "\t\t\t\t\tENTRY UserInterface::sysadmin" << endl;
640     #endif
641
642             //----------- Automatics - objects -------------//
643
644             //----------- Automatics - other ---------------//
645
646         boolean             not_satisfied = TRUE;
647         ControlCommand      choice;
648         int                 int_choice;
649         char                in_char;
650
651             //----------- Code begins ----------------------//
652
653         cout << endl;
654         cout << "\tSelect desired function from menu below." << endl;
655         cout << endl;
656
657             //-----------------------------------------------//
658             // Print menu of options and read the user's     //
659             // choice.                                       //
660             //-----------------------------------------------//
661
662
663         while (not_satisfied)
664         {
665
666             cout << "\t" << CC_exit << "  - Exit the tutoring system." << endl;
667             cout << "\t" << CC_help << "  - Help." << endl;
668             cout << "\t" << CC_admin_write << "  - Write current log entries to file
    ." << endl;
669             cout << "\t" << CC_admin_delete << "  - Delete log file." << endl;
670             cout << "\t" << CC_admin_save << "  - Save log file to new name." << end
    l;
671             cout << "\t" << CC_admin_diag_fail << " - View system diagnosis failures
    ." << endl;
672             cout << "\t" << CC_admin_report_diag_fail << " - Generate report, system
     diagnosis failures." << endl;
673             cout << "\t" << CC_admin_stats << " - View individual past session stati
    stics." << endl;
674             cout << "\t" << CC_admin_report_stats << " - Generate report, individual
     past session statistics." << endl;
675             cout << "\t" << CC_admin_cum_stats << " - View cumulative past session s
    tatistics." << endl;
676             cout << "\t" << CC_admin_report_cum_stats << " - Generate report, cumula
    tive past session statistics." << endl;
677             cout << "\t" << CC_admin_report_full << " - Generate report, all informa
    tion available." << endl;
678             cout << endl;
679
680             cout << "Choice? ";
```



```
681             cin.get(in_char);
682             if (in_char == '?')
683             {
684                 choice = CC_help;
685             }
686             else
687             {
688                 cin.putback(in_char);
689                 cin >> int_choice;
690                 choice = (ControlCommand) int_choice;
691             }
692
693     #ifdef DEBUG
694             cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
695             cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
696     #endif
697
698             //-----------------------------------------------//
699             // Parse choices - some are handled right here, //
700             // others are returned to the caller.           //
701             //-----------------------------------------------//
702
703             switch(choice)
704             {
705                 case CC_admin_write:
706                 case CC_admin_delete:
707                 case CC_admin_save:
708                 case CC_admin_diag_fail:
709                 case CC_admin_report_diag_fail:
710                 case CC_admin_stats:
711                 case CC_admin_report_stats:
712                 case CC_admin_cum_stats:
713                 case CC_admin_report_cum_stats:
714                 case CC_admin_report_full:
715                     not_satisfied = FALSE;
716                     break;
717                 case CC_exit:
718                     not_satisfied = FALSE;
719                     sys_exit_tutor();
720                     break;
721                 case CC_help:
722                     cout << endl;
723                     cout << "\tThis is a menu of the options that you have at this p
    oint." << endl;
724                     cout << "\tType the number of the option which you wish to take.
    " << endl;
725                     cout << "\tYour choices at this point are to exit the system," <
    < endl;
726                     cout << "\tor to select one of the log viewing or report" << end
    l;
727                     cout << "\tgeneration functions." << endl;
728                     cout << endl;
729                     not_satisfied = TRUE;
730                     break;
731                 case CC_NONE:
732                 case CC_try_again:
733                 case CC_hint:
734                 case CC_correct_answer:
735                 case CC_quit:
736                 case CC_admin_find_next:
737                 default:
738                     cout << endl;
739                     cout << "\tInvalid choice, please choose again:" << endl;
740                     not_satisfied = FALSE;
741                     break;
742             }
```

```
743         )
744         return(choice);
745     );
746
747             //-----------------------------------------------//
748             // Member function after_view().                 //
749             //-----------------------------------------------//
750
751     ControlCommand UserInterface::after_view()
752     {
753     #ifdef DEBUG
754         cerr << "\t\t\t\t\tENTRY UserInterface::after_view" << endl;
755     #endif
756
757             //----------- Automatics - objects -------------//
758
759             //----------- Automatics - other ---------------//
760
761         boolean             not_satisfied = TRUE;
762         ControlCommand      choice;
763         int                 int_choice;
764         char                in_char;
765
766             //----------- Code begins ----------------------//
767
768         cout << endl;
769
770             //-----------------------------------------------//
771             // Print menu of options and read the user's     //
772             // choice.                                       //
773             //-----------------------------------------------//
774
775
776         while (not_satisfied)
777         {
778
779             cout << "\t" << CC_exit << "  - Exit the tutoring system." << endl;
780             cout << "\t" << CC_help << "  - Help." << endl;
781             cout << "\t" << CC_quit << "  - Return to main system administrator menu
     " << endl;
782             cout << "\t" << CC_admin_find_next << " - View next entry." << endl;
783             cout << endl;
784
785             cout << "Choice? ";
786             cin.get(in_char);
787             if (in_char == '?')
788             {
789                 choice = CC_help;
790             }
791             else
792             {
793                 cin.putback(in_char);
794                 cin >> int_choice;
795                 choice = (ControlCommand) int_choice;
796             }
797
798     #ifdef DEBUG
799             cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
800             cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
801     #endif
802
803             //-----------------------------------------------//
804             // Parse choices - some are handled right here, //
805             // others are returned to the caller.            //
806             //-----------------------------------------------//
807
```



```
808             if (choice == CC_admin_find_next || choice == CC_quit)
809             {
810                 not_satisfied = FALSE;
811             }
812             else if (choice == CC_exit)
813             {
814                 not_satisfied = FALSE;
815                 sys_exit_tutor();
816             }
817             else if (choice == CC_help)
818             {
819                 cout << endl;
820                 cout << "\tThis is a menu of the options that you have at this point
     " << endl;
821                 cout << "\tType the number of the option which you wish to take." <<
     endl;
822                 cout << "\tYour choices at this point are to exit the system," << en
    dl;
823                 cout << "\tview the next log entry, or quit viewing entries and" <<
    endl;
824                 cout << "\treturn to the main system administrator menu." << endl;
825                 cout << endl;
826                 not_satisfied = TRUE;
827             }
828             else
829             {
830                 cout << endl;
831                 cout << "\tInvalid choice, please choose again:" << endl;
832                 not_satisfied = FALSE;
833             }
834         }
835         return(choice);
836     );
837
838             //-----------------------------------------------//
839             // Member function after_cum().                  //
840             //-----------------------------------------------//
841
842     ControlCommand UserInterface::after_cum()
843     {
844     #ifdef DEBUG
845         cerr << "\t\t\t\t\tENTRY UserInterface::after_cum" << endl;
846     #endif
847
848             //----------- Automatics - objects -------------//
849
850             //----------- Automatics - other ---------------//
851
852         boolean             not_satisfied = TRUE;
853         ControlCommand      choice;
854         int                 int_choice;
855         char                in_char;
856
857             //----------- Code begins ----------------------//
858
859         cout << endl;
860
861             //-----------------------------------------------//
862             // Print menu of options and read the user's     //
863             // choice.                                       //
864             //-----------------------------------------------//
865
866
867         while (not_satisfied)
868         {
869
```

```
870             cout << "\t" << CC_exit << "  - Exit the tutoring system." << endl;
871             cout << "\t" << CC_help << "  - Help." << endl;
872             cout << "\t" << CC_quit << "  - Return to main system administrator menu
     " << endl;
873             cout << endl;
874
875             cout << "Choice? ";
876             cin.get(in_char);
877             if (in_char == '?')
878             {
879                 choice = CC_help;
880             }
881             else
882             {
883                 cin.putback(in_char);
884                 cin >> int_choice;
885                 choice = (ControlCommand) int_choice;
886             }
887
888     #ifdef DEBUG
889             cerr << "\t\t\tDEBUG int choice is " << int_choice << endl;
890             cerr << "\t\t\tDEBUG ControlCommand choice is " << choice << endl;
891     #endif
892
893             //---------------------------------------------//
894             // Parse choices - some are handled right here, //
895             // others are returned to the caller.          //
896             //---------------------------------------------//
897
898             if (choice == CC_quit)
899             {
900                 not_satisfied = FALSE;
901             }
902             else if (choice == CC_exit)
903             {
904                 not_satisfied = FALSE;
905                 sys_exit_tutor();
906             }
907             else if (choice == CC_help)
908             {
909                 cout << endl;
910                 cout << "\tThis is a menu of the options that you have at this point
     " << endl;
911                 cout << "\tType the number of the option which you wish to take." <<
     endl;
912                 cout << "\tYour choices at this point are to exit the system," << en
     dl;
913                 cout << "\tor return to the main system administrator menu." << endl
     ;
914                 cout << endl;
915                 not_satisfied = TRUE;
916             }
917             else
918             {
919                 cout << endl;
920                 cout << "\tInvalid choice, please choose again:" << endl;
921                 not_satisfied = FALSE;
922             }
923         }
924         return(choice);
925     };
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   userlist.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the UserList class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <fstream.h>
21  #include <iostream.h>
22  #include <stdlib.h>
23  #include <string.h>
24  #include "stdtype.h"
25  #include "userlist.h"
26  #include "user.h"
27  #include "log.h"
28  #include "user_defs.h"
29
30          //----------- External data --------------------//
31
32  extern  Log     log;
33
34          //------------------------------------------------//
35          // Constructor.                                   //
36          //------------------------------------------------//
37
38  UserList::UserList()
39  {
40  #ifdef DEBUG
41      cerr << "\t\t\t\t\tENTRY    UserList CONSTRUCTOR" << endl;
42  #endif
43
44      size = 0;
45      current_user = 0;
46      users = new User * [MAX_NUMBER_USERS];
47      list_modified = FALSE;
48  };
49
50          //------------------------------------------------//
51          // Destructor.                                    //
52          //------------------------------------------------//
53
54  UserList::~UserList()
55  {
56  #ifdef DEBUG
57      cerr << "\t\t\t\t\tENTRY    UserList DESTRUCTOR" << endl;
58  #endif
59
60      for (int i=0; i<size; i++)
61      {
62          delete users[i];
63      }
64      delete [] users;
65  };
66
```



```
67          //------------------------------------------------//
68          // Member function establish_connection.          //
69          //------------------------------------------------//
70
71  Usertype  UserList::establish_connection()
72  {
73  #ifdef DEBUG
74      cerr << "\t\t\t\t\tENTRY UserList::establish_connection" << endl;
75  #endif
76
77          //----------- Automatics - objects -------------//
78
79          //----------- Automatics - other ---------------//
80
81      char        ch;
82      char        in_name[MAX_LOGIN_NAME_STRING];
83      Usertype    type;
84      int         in_index;
85      int         search_index;
86      boolean     still_trying = TRUE;
87      int         attempts = 0;
88
89          //----------- Code begins ----------------------//
90
91          //------------------------------------------------//
92          // Prompt the user for a login name and read.     //
93          //------------------------------------------------//
94
95      while (still_trying)
96      {
97          cout << "Please login: ";
98  #ifdef DEBUG
99          cerr << "Please login: " << endl;
100 #endif
101         in_index = 0;
102
103         while (cin.get(ch))
104         {
105             if (ch == '\n')
106             {
107                 in_name[in_index++] = '\0';
108                 break;
109             }
110             else
111             {
112                 in_name[in_index++] = ch;
113             }
114         }
115
116 #ifdef DEBUG
117         cerr << "\t\t\tDEBUG login name attempt read is " << in_name << endl;
118 #endif
119
120         //------------------------------------------------//
121         // Loop through the current list of valid         //
122         // User objects, asking each to attempt a match   //
123         // against the name which was read.               //
124         //------------------------------------------------//
125
126         for (search_index=0; search_index<size; search_index++)
127         {
128             type = users[search_index]->match_user(in_name);
129             if (type != NO_USERTYPE)
130             {
131                 break;
132             }
```

```
133         }
134
135         //-----------------------------------------------//
136         // If no match is found, handle possible        //
137         // alternatives.                                 //
138         //-----------------------------------------------//
139         if (search_index == size)
140         {
141             if (strcmp("?",in_name)==0)
142             {
143         //-----------------------------------------------//
144         // If user requested help, supply it.            //
145         //-----------------------------------------------//
146
147                 still_trying = TRUE;
148                 cout << endl;
149                 cout << "\tIn order the use this system, you must have an author
    ized" << endl;
150                 cout << "\tusername.  If you have not received a username yet, s
    ee" << endl;
151                 cout << "\tyour system administrator.  If you do have a valid us
    ername," << endl;
152                 cout << "\ttype it in now to gain access to the tutoring system.
    " << endl;
153                 cout << endl;
154             }
155             else if (attempts >= 2)
156             {
157         //-----------------------------------------------//
158         // If this is the third bad try, exit the       //
159         // program.                                      //
160         //-----------------------------------------------//
161
162                 still_trying = FALSE;
163                 cout << endl;
164                 cout << "\tDiscontinuing attempted login after three tries." <<
    endl;
165                 cout << "\tPlease verify that you are using a correct username,"
     << endl;
166                 cout << "\tor see the system administrator." << endl;
167                 cout << endl;
168                 exit(0);
169             }
170             else
171             {
172         //-----------------------------------------------//
173         // If this is a bad attempt, but the above      //
174         // limit has not been reached, ask the user     //
175         // to try again.                                 //
176         //-----------------------------------------------//
177
178                 attempts++;
179                 still_trying = TRUE;
180                 cout << endl;
181                 cout << "\tThat is not a correct username, please verify that" <
    < endl;
182                 cout << "\tyou typed what you thought you did, and that you have
     not" << endl;
183                 cout << "\tused upper or lower case when not expected." << endl;
184                 cout << endl;
185             }
186         }
187         else
188         {
189         //-----------------------------------------------//
190         // If valid username has been entered, log      //
```



```
191         // the current user, query current knowledge    //
192         // level, and set if different from previously  //
193         // known level.                                  //
194         //-----------------------------------------------//
195
196             still_trying = FALSE;
197             current_user = search_index;
198             log.set_current_user((users[current_user])->get_string());
199
200             int in_level;
201             int cur_level;
202             if (type == STUDENT)
203             {
204                 cur_level = users[current_user]->get_current_level();
205                 cout << endl;
206
207         //-----------------------------------------------//
208         // Prompt the user for his/her current          //
209         // knowledge level.  Make sure it is valid.     //
210         //-----------------------------------------------//
211
212                 boolean level_needed = TRUE;
213
214                 while (level_needed)
215                 {
216                     cout << "Enter highest chapter completed (currently " <<
217                                                         cur_level << ")? ";
218                     cin >> in_level;
219                     if (in_level < MIN_KNOW_LEVEL)
220                     {
221                         cout << "Chapter must be greater than or" <<
222                                     " equal to the minimum level, " <<
223                                     MIN_KNOW_LEVEL << endl;
224                     }
225                     else if (in_level > MAX_KNOW_LEVEL)
226                     {
227                         cout << "The maximum chapter supported by" <<
228                                     " the system is " <<
229                                     MAX_KNOW_LEVEL << endl;
230                         cout << "Please enter that number if you wish to" <<
231                                     " select the maximum" << endl;
232                     }
233                     else
234                     {
235                         level_needed = FALSE;
236                     }
237                 }
238
239                 if (users[current_user]->set_current_level(in_level) == TRUE)
240                 {
241                     list_modified = TRUE;
242                 }
243                 cout << endl;
244                 cout << "\tExercises will be limited to chapter " << in_level <<
    " and earlier." << endl;
245             }
246         }
247     }
248     return(type);
249
250 };
251
252         //-----------------------------------------------//
253         // Member function read.                        //
254         //-----------------------------------------------//
255
```

```
256  void UserList::read()
257  {
258  #ifdef DEBUG
259      cerr << "\t\t\t\t\tENTRY UserList::read" << endl;
260  #endif
261
262      ifstream user_stream;
263
264          //-----------------------------------------------//
265          // Open the user data file for input.  Loop      //
266          // and ask User objects to read data.            //
267          //-----------------------------------------------//
268
269      user_stream.open("users.data", ios::in);
270      if (!user_stream)
271      {
272          cerr << "ERROR - could not open user data file." << endl;
273      }
274  #ifdef DEBUG
275      else
276      {
277          cerr << "\t\t\tDEBUG users.data successfully opened." << endl;
278      }
279  #endif
280
281      do
282      {
283          users[size] = new User;
284          size++;
285      }
286      while (users[size-1]->read(&user_stream));
287
288      user_stream.close();
289  );
290
291          //-----------------------------------------------//
292          // Member function get_current_level().          //
293          //-----------------------------------------------//
294
295  int UserList::get_current_level()
296  {
297  #ifdef DEBUG
298      cerr << "\t\t\t\t\tENTRY UserList::get_current_level" << endl;
299  #endif
300
301      return( users[current_user]->get_current_level() );
302  };
303
304          //-----------------------------------------------//
305          // Member function write().                      //
306          //-----------------------------------------------//
307
308  void UserList::write()
309  {
310  #ifdef DEBUG
311      cerr << "\t\t\t\t\tENTRY UserList::write" << endl;
312  #endif
313
314      ofstream user_stream;
315
316          //-----------------------------------------------//
317          // If user data has not changed, do nothing.     //
318          // Otherwise, open the user data file for        //
319          // output and loop through the user list         //
320          // requesting that each write itself out to the  //
321          // file.                                         //
```



```
322          //-----------------------------------------------//
323
324      if (list_modified == TRUE)
325      {
326          user_stream.open("users.data", ios::out);
327
328          if (!user_stream)
329          {
330              cerr << "ERROR - could not open user data file." << endl;
331          }
332  #ifdef DEBUG
333          else
334          {
335              cerr << "\t\t\tDEBUG users.data successfully opened." << endl;
336          }
337  #endif
338
339          for (int i=0; i<(size-1); i++)
340          {
341              users[i]->write(&user_stream);
342          }
343
344          user_stream.close();
345      }
346  #ifdef DEBUG
347      else
348      {
349          cerr << "\t\t\tDEBUG list not modified" << endl;
350      }
351  #endif
352  );
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   utility_gender1_var.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This function takes a Gender attribute, and returns the
8   //              Gender attribute which is referred to as the "first gender
9   //              variation".  This is merely the next Gender attribute
10  //              (numerically) modulo the valid number of choices.  It
11  //              relies on the Gender enumeration.
12  //
13  //  Notes:      The theory and design of this system have been documented
14  //              in the accompanying thesis report.
15  //
16  //  Revisions   By              Reason
17  //  ---------   --              ------
18  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
19  //
20  //-------------------------------------------------------------------------
21
22          //----------- Included files -------------------//
23
24  #include <iostream.h>
25  #include "worddescriptor.h"
26
27          //----------- External Data --------------------//
28
29          //----------- Calling syntax -------------------//
30
31  Gender utility_gender1_var(Gender in_gender)
32  {
33
34          //----------- Automatics - objects -------------//
35
36          //----------- Automatics - other ---------------//
37
38      int         zero_based_input;
39      int         variation;
40      int         one_based_output;
41
42          //----------- Code begins ----------------------//
43
44  #ifdef DEBUG
45      cerr << "\t\t\t\t\tENTRY utility_gender1_var()" << endl;
46  #endif
47
48          //------------------------------------------------//
49          // Convert input gender to a zero-based          //
50          // integer value for math.                       //
51          //------------------------------------------------//
52
53      zero_based_input = ((int) in_gender) - 1;
54
55          //------------------------------------------------//
56          // The enumerated value G_none is one more than //
57          // the highest attribute value, so (G_none-1)   //
58          // is the number of possible values.  Calculate //
59          // the variation, modulo this number of         //
60          // possible values.                             //
61          //------------------------------------------------//
62
63      variation = (zero_based_input + 1) % (G_none-1);
64
65          //------------------------------------------------//
66          // Return the resulting number as a Gender      //
```



```
67          // attribute.                                   //
68          //------------------------------------------------//
69
70      one_based_output = variation + 1;
71      return( (Gender) one_based_output );
72  }
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   utility_gender2_var.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This function takes a Gender attribute, and returns the
8   //              Gender attribute which is referred to as the "second gender
9   //              variation".  This is merely the Gender attribute
10  //              (numerically) plus 2, modulo the valid number of choices.  It
11  //              relies on the Gender enumeration.
12  //
13  //  Notes:      The theory and design of this system have been documented
14  //              in the accompanying thesis report.
15  //
16  //  Revisions   By              Reason
17  //  ---------   --              ------
18  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
19  //
20  //-------------------------------------------------------------------------
21
22          //----------- Included files -------------------//
23
24  #include <iostream.h>
25  #include "worddescriptor.h"
26
27          //----------- External Data --------------------//
28
29          //----------- Calling syntax -------------------//
30
31  Gender utility_gender2_var(Gender in_gender)
32  {
33
34      .   //----------- Automatics - objects -------------//
35
36          //----------- Automatics - other ---------------//
37
38      int         zero_based_input;
39      int         variation;
40      int         one_based_output;
41
42          //----------- Code begins ----------------------//
43
44  #ifdef DEBUG
45      cerr << "\t\t\t\t\tENTRY utility_gender2_var()" << endl;
46  #endif
47
48          //------------------------------------------------//
49          // Convert input gender to a zero-based          //
50          // integer value for math.                       //
51          //------------------------------------------------//
52
53      zero_based_input = ((int) in_gender) - 1;
54
55          //------------------------------------------------//
56          // The enumerated value G_none is one more than //
57          // the highest attribute value, so (G_none-1)   //
58          // is the number of possible values.  Calculate //
59          // the variation, modulo this number of         //
60          // possible values.                             //
61          //------------------------------------------------//
62
63      variation = (zero_based_input + 2) % (G_none-1);
64
65          //------------------------------------------------//
66          // Return the resulting number as a Gender      //
```



```
67          // attribute.                                   //
68          //------------------------------------------------//
69
70      one_based_output = variation + 1;
71      return( (Gender) one_based_output );
72  }
```

```
1     //-------------------------------------------------------------------------------
2     //
3     //  Filename:   utility_not_capitalized.C
4     //
5     //  Project:    German Language Tutor, RIT Master's Thesis
6     //
7     //  Purpose:    This function takes a Word, and returns the same word
8     //              with the first letter lower-case.  Note that it assumes
9     //              that the first letter is already upper-case.  This is the
10    //              responsiblity of the caller.
11    //
12    //  Notes:      The theory and design of this system have been documented
13    //              in the accompanying thesis report.
14    //
15    //  Revisions   By              Reason
16    //  ---------   --              ------
17    //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
18    //
19    //-------------------------------------------------------------------------------
20
21            //------------ Included files --------------------//
22
23    #include <iostream.h>
24    #include <string.h>
25    #include "word.h"
26    #include "PC_defs.h"
27
28            //------------ External Data ---------------------//
29
30            //------------ Calling syntax --------------------//
31
32    Word utility_not_capitalized(Word in_word)
33    {
34
35            //------------ Automatics - objects --------------//
36
37        Word new_word;
38
39            //------------ Automatics - other ----------------//
40
41        char  new_word_string[MAX_WORD_LENGTH];
42
43            //------------ Code begins -----------------------//
44
45    #ifdef DEBUG
46        cerr << "\t\t\t\t\tENTRY utility_not_capitalized()" << endl;
47    #endif
48
49            //-------------------------------------------------//
50            // Obtain the current word string, and convert  //
51            // the first character to lower case.           //
52            //-------------------------------------------------//
53
54        strncpy(new_word_string, in_word.get_string(), MAX_WORD_LENGTH);
55        new_word_string[0] = new_word_string[0] + CASE_CONVERSION_OFFSET;
56
57            //-------------------------------------------------//
58            // Set this string for the new word, and return //
59            // this word to the caller.                     //
60            //-------------------------------------------------//
61
62        new_word.set_string(new_word_string);
63        return( new_word );
64
65    }
```

```
1   //-------------------------------------------------------------------------
2   //
3   //  Filename:   word.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the Word class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //              The get_string() member function returns a pointer to the
13  //              internal string.  This should probably be improved, but for
14  //              the time being, it is important for the caller to realize
15  //              that the pointer can only be used while this object is in
16  //              scope.
17  //
18  //  Revisions   By              Reason
19  //  ---------   --              ------
20  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
21  //
22  //-------------------------------------------------------------------------
23
24          //----------- Included files -------------------//
25
26  #include <iostream.h>
27  #include "word.h"
28  #include <string.h>
29
30          //-------------------------------------------//
31          // Constructor 1 of 3.                       //
32          //-------------------------------------------//
33
34  Word::Word()
35  {
36  #ifdef DEBUG
37      cerr << "\t\t\t\t\tENTRY    Word CONSTRUCTOR 1" << endl;
38      cerr << "\t\t\t\t\tENTRY    (for automatics)" << endl;
39  #endif
40
41          //-------------------------------------------//
42          // Initialize all internal data to default or   //
43          // "unassigned" values.                      //
44          //-------------------------------------------//
45
46      index = 0;
47      word = new char[MAX_WORD_LENGTH];
48      type = WD_UNKNOWN_TYPE;
49      count = C_none;
50      gender = G_none;
51      language = L_none;
52  );
53
54          //-------------------------------------------//
55          // Constructor 2 of 3.                       //
56          //-------------------------------------------//
57
58  Word::Word(char * in_static, WD_Type in_type, Count in_count, Gender in_gender,
59                               Language in_lang)
60  {
61  #ifdef DEBUG
62      cerr << "\t\t\t\t\tENTRY    Word CONSTRUCTOR 2" << endl;
63      cerr << "\t\t\t\t\tENTRY    (for knowledge base)" << endl;
64  #endif
65
66          //-------------------------------------------//
```



```
67          // Set internal data based on input.         //
68          //-------------------------------------------//
69
70      index = 0;
71      word = in_static;
72      type = in_type;
73      count = in_count;
74      gender = in_gender;
75      language = in_lang;
76  );
77
78          //-------------------------------------------//
79          // Constructor 3 of 3.                       //
80          //-------------------------------------------//
81
82  Word::Word(const Word & in_word)
83  {
84  #ifdef DEBUG
85      cerr << "\t\t\t\t\tENTRY    Word CONSTRUCTOR 3" << endl;
86      cerr << "\t\t\t\t\tENTRY    (for copies)" << endl;
87  #endif
88
89          //-------------------------------------------//
90          // Copy constructor.  Initialize this word   //
91          // based on input.                           //
92          //-------------------------------------------//
93
94      word = new char[MAX_WORD_LENGTH];
95      strcpy( word, in_word.word );
96
97      index = in_word.index;
98      type = in_word.type;
99      count = in_word.count;
100     gender = in_word.gender;
101     language = in_word.language;
102
103 );
104
105         //-------------------------------------------//
106         // Destructor.                               //
107         //-------------------------------------------//
108
109 Word::~Word()
110 {
111 #ifdef DEBUG
112     cerr << "\t\t\t\t\tENTRY    Word DESTRUCTOR" << endl;
113 #endif
114
115     delete [] word;
116 );
117
118         //-------------------------------------------//
119         // Assignment operator.                      //
120         //-------------------------------------------//
121
122 Word& Word::operator=(Word & in_word)
123 {
124 #ifdef DEBUG
125     cerr << "\t\t\t\t\tENTRY    Word::operator=" << endl;
126 #endif
127
128         //-------------------------------------------//
129         // If assignment to self, do nothing.        //
130         // Otherwise, clear any current data, and    //
131         // create new data to match the Word on the  //
132         // right-hand side of the assignment.        //
```

```
133          //-----------------------------------------------//
134
135      if (&in_word != this)
136      {
137          delete [] word;
138          word = new char(strlen(in_word.word)+1];
139          strcpy(word,in_word.word);
140          index = in_word.index;
141          type = in_word.type;
142          count = in_word.count;
143          gender = in_word.gender;
144          language = in_word.language;
145      }
146      return *this;
147  };
148
149          //-----------------------------------------------//
150          // Member function set_string().                 //
151          //-----------------------------------------------//
152
153  void Word::set_string(char * in_string)
154  {
155  #ifdef DEBUG
156      cerr << "\t\t\t\t\tENTRY Word::set_string" << endl;
157  #endif
158
159          //----------- Automatics - objects -------------//
160
161          //----------- Automatics - other ---------------//
162
163          //----------- Code begins ----------------------//
164
165          //-----------------------------------------------//
166          // Copy the contents of the input string to the //
167          // internal string.                              //
168          //-----------------------------------------------//
169
170      strncpy(word,in_string,MAX_WORD_LENGTH);
171  }
172
173          //-----------------------------------------------//
174          // Member function read().                       //
175          //-----------------------------------------------//
176
177  boolean Word::read()
178  {
179  #ifdef DEBUG
180      cerr << "\t\t\t\t\tENTRY Word::read" << endl;
181  #endif
182
183          //----------- Automatics - objects -------------//
184
185          //----------- Automatics - other ---------------//
186
187      char        ch;
188      int         ch_index = 0;
189      boolean     end_of_line = FALSE;
190      boolean     end_of_word = FALSE;
191
192          //----------- Code begins ----------------------//
193
194          //-----------------------------------------------//
195          // Read characters from the input stream        //
196          // until end-of-word is encountered.            //
197          //-----------------------------------------------//
198
```



```
199      while (!end_of_word && cin.get(ch) )
200      {
201  #ifdef DEBUG
202          cerr << "\t\t\tDEBUG char just read was -" << ch << "-" << endl;
203  #endif
204          switch (ch)
205          {
206          //-----------------------------------------------//
207          // White space indicates end-of-word.           //
208          //-----------------------------------------------//
209              case '\t':
210              case ' ':
211  #ifdef DEBUG
212                  cerr << "\t\t\tDEBUG whitespace" << endl;
213  #endif
214                  end_of_word = TRUE;
215                  break;
216          //-----------------------------------------------//
217          // A newline indicates both end-of-word, and    //
218          // end-of-line, which will be indicated to the  //
219          // caller.                                      //
220          //-----------------------------------------------//
221              case '\n':
222  #ifdef DEBUG
223                  cerr << "\t\t\tDEBUG newline" << endl;
224  #endif
225                  end_of_word = TRUE;
226                  end_of_line = TRUE;
227                  break;
228              default:
229          //-----------------------------------------------//
230          // If not white space or newline, store as      //
231          // character of word.                           //
232          //-----------------------------------------------//
233  #ifdef DEBUG
234                  cerr << "\t\t\tDEBUG valid char for word" << endl;
235  #endif
236                  word[ch_index++] = ch;
237                  break;
238          }
239          word[ch_index] = '\0';
240      }
241      return( end_of_line );
242  }
243
244          //-----------------------------------------------//
245          // Member function get_string().                 //
246          //-----------------------------------------------//
247
248  char * Word::get_string()
249  {
250  #ifdef DEBUG
251      cerr << "\t\t\t\t\tENTRY Word::get_string" << endl;
252  #endif
253
254          //----------- Automatics - objects -------------//
255
256          //----------- Automatics - other ---------------//
257
258          //----------- Code begins ----------------------//
259
260
261      return(word);
262  }
263
264          //-----------------------------------------------//
```

```
265         // Member function print().                    //
266         //-------------------------------------------//
267
268   void Word::print()
269   {
270   #ifdef DEBUG
271       cerr << "\t\t\t\t\tENTRY Word::print" << endl;
272
273   #endif
274
275         //----------- Automatics - objects -------------//
276
277         //----------- Automatics - other ---------------//
278
279         //----------- Code begins ----------------------//
280
281
282   #ifdef DEBUG
283       cerr << word;
284   #endif
285       cout << word;
286   }
287
288   #ifdef DEBUG
289         //-------------------------------------------//
290         // Member function debug_print().             //
291         //-------------------------------------------//
292
293   void Word::debug_print()
294   {
295       cerr << "\t\t\t\t\tENTRY Word::debug_print" << endl;
296
297           //----------- Automatics - objects -------------//
298
299           //----------- Automatics - other ---------------//
300
301           //----------- Code begins ----------------------//
302
303
304           //-------------------------------------------//
305           // Because this is debug_print(), send the word //
306           // to the standard error stream (that's where   //
307           // all debug output of the program goes.        //
308           //-------------------------------------------//
309
310       cerr << word;
311   }
312   #endif
313
314         //-------------------------------------------//
315         // Member function set_type().                //
316         //-------------------------------------------//
317
318   void Word::set_type(WD_Type in_type)
319   {
320   #ifdef DEBUG
321       cerr << "\t\t\t\t\tENTRY Word::set_type" << endl;
322   #endif
323
324         //----------- Automatics - objects -------------//
325
326         //----------- Automatics - other ---------------//
327
328         //----------- Code begins ----------------------//
329
330
```



```
331       type = in_type;
332   }
333
334         //-------------------------------------------//
335         // Member function set_index().               //
336         //-------------------------------------------//
337
338   void Word::set_index(int in_index)
339   {
340   #ifdef DEBUG
341       cerr << "\t\t\t\t\tENTRY Word::set_index" << endl;
342   #endif
343
344         //----------- Automatics - objects -------------//
345
346         //----------- Automatics - other ---------------//
347
348         //----------- Code begins ----------------------//
349
350
351       index = in_index;
352   }
353
354         //-------------------------------------------//
355         // Member function set_language().            //
356         //-------------------------------------------//
357
358   void Word::set_language(Language in_lang)
359   {
360   #ifdef DEBUG
361       cerr << "\t\t\t\t\tENTRY Word::set_language" << endl;
362   #endif
363
364         //----------- Automatics - objects -------------//
365
366         //----------- Automatics - other ---------------//
367
368         //----------- Code begins ----------------------//
369
370
371       language = in_lang;
372   }
373
374         //-------------------------------------------//
375         // Member function set_gender().              //
376         //-------------------------------------------//
377
378   void Word::set_gender(Gender in_gend)
379   {
380   #ifdef DEBUG
381       cerr << "\t\t\t\t\tENTRY Word::set_gender" << endl;
382   #endif
383
384         //----------- Automatics - objects -------------//
385
386         //----------- Automatics - other ---------------//
387
388         //----------- Code begins ----------------------//
389
390
391       gender = in_gend;
392   }
393
394         //-------------------------------------------//
395         // Member function set_count().               //
396         //-------------------------------------------//
```

```
397
398   void Word::set_count(Count in_count)
399   {
400   #ifdef DEBUG
401       cerr << "\t\t\t\t\tENTRY Word::set_count" << endl;
402   #endif
403
404           //----------- Automatics - objects -------------//
405
406           //----------- Automatics - other ---------------//
407
408           //----------- Code begins ----------------------//
409
410
411       count = in_count;
412   }
413
414           //-----------------------------------------------//
415           // Member function get_language().               //
416           //-----------------------------------------------//
417
418   Language Word::get_language()
419   {
420   #ifdef DEBUG
421       cerr << "\t\t\t\t\tENTRY Word::get_language" << endl;
422   #endif
423
424           //----------- Automatics - objects -------------//
425
426           //----------- Automatics - other ---------------//
427
428           //----------- Code begins ----------------------//
429
430
431       return(language);
432   }
433
434           //-----------------------------------------------//
435           // Member function get_index().                  //
436           //-----------------------------------------------//
437
438   int Word::get_index()
439   {
440   #ifdef DEBUG
441       cerr << "\t\t\t\t\tENTRY Word::get_index" << endl;
442   #endif
443
444           //----------- Automatics - objects -------------//
445
446           //----------- Automatics - other ---------------//
447
448           //----------- Code begins ----------------------//
449
450
451       return(index);
452   }
453
454           //-----------------------------------------------//
455           // Member function get_type().                   //
456           //-----------------------------------------------//
457
458   WD_Type Word::get_type()
459   {
460   #ifdef DEBUG
461       cerr << "\t\t\t\t\tENTRY Word::get_type" << endl;
462   #endif
```



```
463
464           //----------- Automatics - objects -------------//
465
466           //----------- Automatics - other ---------------//
467
468           //----------- Code begins ----------------------//
469
470
471       return(type);
472   }
473
474           //-----------------------------------------------//
475           // Member function get_gender().                 //
476           //-----------------------------------------------//
477
478   Gender Word::get_gender()
479   {
480   #ifdef DEBUG
481       cerr << "\t\t\t\t\tENTRY Word::get_gender" << endl;
482   #endif
483
484           //----------- Automatics - objects -------------//
485
486           //----------- Automatics - other ---------------//
487
488           //----------- Code begins ----------------------//
489
490
491       return(gender);
492   }
493
494           //-----------------------------------------------//
495           // Member function get_count().                  //
496           //-----------------------------------------------//
497
498   Count Word::get_count()
499   {
500   #ifdef DEBUG
501       cerr << "\t\t\t\t\tENTRY Word::get_count" << endl;
502   #endif
503
504           //----------- Automatics - objects -------------//
505
506           //----------- Automatics - other ---------------//
507
508           //----------- Code begins ----------------------//
509
510
511       return(count);
512   }
```

```
1   //-------------------------------------------------------------------------------
2   //
3   //  Filename:   worddescriptor.C
4   //
5   //  Project:    German Language Tutor, RIT Master's Thesis
6   //
7   //  Purpose:    This file implements the WordDescriptor class.
8   //
9   //  Notes:      The theory and design of this system have been documented
10  //              in the accompanying thesis report.
11  //
12  //  Revisions   By              Reason
13  //  ---------   --              ------
14  //  26-Apr-93   Ken Staffan     v1.0 release with final thesis report.
15  //
16  //-------------------------------------------------------------------------------
17
18          //----------- Included files -------------------//
19
20  #include <iostream.h>
21  #include "word.h"
22  #include "worddescriptor.h"
23  #include "dynamicworddescriptor.h"
24
25          //------------------------------------------------//
26          // Constructor 1.                                 //
27          //------------------------------------------------//
28
29  WordDescriptor::WordDescriptor(Count in_count, Gender in_gend, Language
30                                                  in_lang, WD_Type in_type)
31  {
32  #ifdef DEBUG
33      cerr << "\t\t\t\t\tENTRY    WordDescriptor CONSTRUCTOR 1" << endl;
34      cerr << "\t\t\t\t\tENTRY    (for KB - all fields specified)" << endl;
35  #endif
36
37      type = in_type;
38      count = in_count;
39      gender = in_gend;
40      language = in_lang;
41  };
42
43          //------------------------------------------------//
44          // Constructor 2.                                 //
45          //------------------------------------------------//
46
47  WordDescriptor::WordDescriptor(Word & in_word)
48  {
49  #ifdef DEBUG
50      cerr << "\t\t\t\t\tENTRY    WordDescriptor CONSTRUCTOR 2" << endl;
51      cerr << "\t\t\t\t\tENTRY    (made from an existing Word)" << endl;
52  #endif
53
54      type = in_word.get_type();
55      count = in_word.get_count();
56      gender = in_word.get_gender();
57      language = in_word.get_language();
58  };
59
60          //------------------------------------------------//
61          // Destructor.                                    //
62          //------------------------------------------------//
63
64  WordDescriptor::~WordDescriptor()
65  {
66  #ifdef DEBUG
```



```
67      cerr << "\t\t\t\t\tENTRY    WordDescriptor DESTRUCTOR" << endl;
68  #endif
69  };
70
71          //------------------------------------------------//
72          // Member function set_gender().                  //
73          //------------------------------------------------//
74
75  void WordDescriptor::set_gender(Gender in_gender)
76  {
77  #ifdef DEBUG
78      cerr << "\t\t\t\t\tENTRY WordDescriptor::set_gender" << endl;
79  #endif
80
81          //----------- Automatics - objects -------------//
82
83          //----------- Automatics - other ---------------//
84
85          //----------- Code begins ----------------------//
86
87
88      gender = in_gender;
89  };
90
91          //------------------------------------------------//
92          // Member function set_language().                //
93          //------------------------------------------------//
94
95  void WordDescriptor::set_language(Language in_language)
96  {
97  #ifdef DEBUG
98      cerr << "\t\t\t\t\tENTRY WordDescriptor::set_language" << endl;
99  #endif
100
101         //----------- Automatics - objects -------------//
102
103         //----------- Automatics - other ---------------//
104
105         //----------- Code begins ----------------------//
106
107
108     language = in_language;
109 };
110
111         //------------------------------------------------//
112         // Member function set_count().                   //
113         //------------------------------------------------//
114
115 void WordDescriptor::set_count(Count in_count)
116 {
117 #ifdef DEBUG
118     cerr << "\t\t\t\t\tENTRY WordDescriptor::set_count" << endl;
119 #endif
120
121         //----------- Automatics - objects -------------//
122
123         //----------- Automatics - other ---------------//
124
125         //----------- Code begins ----------------------//
126
127
128     count = in_count;
129 };
130
131         //------------------------------------------------//
132         // Member function get_count().                   //
```

```
133         //-----------------------------------------------//
134
135 Count WordDescriptor::get_count()
136 {
137 #ifdef DEBUG
138     cerr << "\t\t\t\t\tENTRY WordDescriptor::get_count" << endl;
139 #endif
140
141         //----------- Automatics - objects -------------//
142
143         //----------- Automatics - other ---------------//
144
145         //----------- Code begins ----------------------//
146
147
148     return(count);
149 );
150
151         //-----------------------------------------------//
152         // Member function get_gender().                 //
153         //-----------------------------------------------//
154
155 Gender WordDescriptor::get_gender()
156 {
157 #ifdef DEBUG
158     cerr << "\t\t\t\t\tENTRY WordDescriptor::get_gender" << endl;
159 #endif
160
161         //----------- Automatics - objects -------------//
162
163         //----------- Automatics - other ---------------//
164
165         //----------- Code begins ----------------------//
166
167
168     return(gender);
169 );
170
171         //-----------------------------------------------//
172         // Member function get_language().               //
173         //-----------------------------------------------//
174
175 Language WordDescriptor::get_language()
176 {
177 #ifdef DEBUG
178     cerr << "\t\t\t\t\tENTRY WordDescriptor::get_language" << endl;
179 #endif
180
181         //----------- Automatics - objects -------------//
182
183         //----------- Automatics - other ---------------//
184
185         //----------- Code begins ----------------------//
186
187
188     return(language);
189 );
190
191         //-----------------------------------------------//
192         // Member function no_dynamics().                //
193         //-----------------------------------------------//
194
195 boolean WordDescriptor::no_dynamics()
196 {
197 #ifdef DEBUG
198     cerr << "\t\t\t\t\tENTRY WordDescriptor::no_dynamics" << endl;
```



```
199 #endif
200
201         //----------- Automatics - objects -------------//
202
203         //----------- Automatics - other ---------------//
204
205         //----------- Code begins ----------------------//
206
207         //-----------------------------------------------//
208         // If no internal attributes are set as          //
209         // DYNAMIC (waiting to be assigned), return      //
210         // TRUE.  Otherwise, return FALSE.               //
211         //-----------------------------------------------//
212
213     if (count == C_DYNAMIC || gender == G_DYNAMIC || language == L_DYNAMIC)
214     {
215 #ifdef DEBUG
216         cerr << "\t\t\tDEBUG no_dynamics returning FALSE" << endl;
217 #endif
218         return( FALSE );
219     }
220     else
221     {
222 #ifdef DEBUG
223         cerr << "\t\t\tDEBUG no_dynamics returning TRUE" << endl;
224 #endif
225         return( TRUE );
226     }
227 );
228
229         //-----------------------------------------------//
230         // Member function resolve_dynamics().           //
231         //-----------------------------------------------//
232
233 void WordDescriptor::resolve_dynamics(DynamicWordDescriptor& in_dyna)
234 {
235 #ifdef DEBUG
236     cerr << "\t\t\t\t\tENTRY WordDescriptor::resolve_dynamics" << endl;
237 #endif
238
239         //----------- Automatics - objects -------------//
240
241         //----------- Automatics - other ---------------//
242
243         //----------- Code begins ----------------------//
244
245         //-----------------------------------------------//
246         // For any internal attributes which are set     //
247         // as DYNAMIC (waiting to be assigned), attempt  //
248         // to obtain a value from the input              //
249         // DynamicWordDescriptor.                        //
250         //-----------------------------------------------//
251
252     if (count == C_DYNAMIC)
253     {
254 #ifdef DEBUG
255         cerr << "\t\t\tDEBUG resolving dynamic attribute count for WD" << endl;
256 #endif
257         count = in_dyna.get_count();    // Might still be dynamic (but might not
    ).
258     }
259     if (gender == G_DYNAMIC)
260     {
261 #ifdef DEBUG
262         cerr << "\t\t\tDEBUG resolving dynamic attribute gender for WD" << endl;
263 #endif
```

```
264             gender = in_dyna.get_gender();
265     #ifdef DEBUG
266             cerr << "\t\t\tDEBUG resolved dynamic attribute gender for WD" << endl;
267     #endif
268         }
269         if (language == L_DYNAMIC)
270         {
271     #ifdef DEBUG
272             cerr << "\t\t\tDEBUG resolving dynamic attribute lang for WD" << endl;
273     #endif
274             language = in_dyna.get_language();
275         }
276     );
```

```
1
2       5-May-93, Kenneth E. Staffan
3
4       When saved to floppy disks, the German Language Tutor system was left
5       in a "stable state".  All source files, scripts, and build procedures were
6       baselined in SCCS.  The system was then built, leaving executables and
7       reference copies of all files (source code in /src directories, include
8       files in /hdr directories, build procedures and scripts in the top-level and
9       component-level directories).  One file, compoundstring.C compiles with
10      warnings.  These are due to the use of the 'regexp' utility, and the
11      debug_print() member function.  I didn't bother too much with them, because
12      they do not affect the system at run-time, debug_print() is only debug
13      code, and I wanted to switch to another regular expression package anyway.
14
15      The system was executed, leaving behind debug and system administration
16      logs.  The entire directory structure was then put on floppies.
17
18      Anyone interested in using, enhancing, or otherwise digging into this
19      code, should reference the corresponding thesis report, which also
20      contains user and system administrator manuals.
21
22      The following summarizes the directories and files saved:
23
24              tutor - top-level directory
25
26                  READ_ME      - this file.
27
28                  /SCCS        - sccs controlled top-level files.
29
30                  baseline_all
31                  checkin_all
32                  checkout_all
33                  print_all
34                  split        - supporting scripts (all in SCCS also).
35
36                  /hdr         - top-level include files (all in SCCS also).
37
38                  makefile     - system build procedure (in SCCS also).
39                  users.data   - system input - valid usernames.
40                  glt          - system executable (run using split script).
41                  debug.log    - system output - verbose debugging output.
42                  glt.log      - system output - run-time gathered usage data.
43
44                  /construction
45                  /dictionary
46                  /log
47                  /models
48                  /session
49                  /users       - component-level directories.  All components are
50                                 similar, so the 'users' component is broken down
51                                 further to illustrate:
52
53                      /SCCS    - sccs controlled component-level files.
54
55                      /hdr     - component-level include files (all in SCCS also).
56                      /lis     - compiler-generated listing files (not currently used).
57                      /obj     - compiler-generated individual object modules.
58                      /src     - component-level source files (all in SCCS also).
59
60                      makefile - component build procedure (in SCCS also).
61                      users.o  - component object module, to be linked at top-level.
62
63      NOTE:  Because the *_all scripts do some pretty global things, they should
64             be used with care.  I always made a backup copy of the directory
65             structure before I ran them - just in case!
```

```
#
#
#  The procedure for building the German language tutor thesis program (glt).
#  This makefile may invoke makefiles at the sub-system level, located below
#  this directory.
#
#  Modification History:
#
#  When     Who          Why
#  ----     ---          ---
#  07Nov92  Ken Staffan  Initial stub version.
#  22Nov92       "       First config - single sub-system, 'models'
#  20Feb93       "       Changed models.lnk to models.o & tutor target to glt.
#  26Mar93       "       Added other sub-systems and global header files.
#  08Apr93       "       Phrase.h became compoundstring.h.
#  12Apr93       "       Added hint.h
#  17Apr93       "       Ensure that dictionary builds before phrase models.
#  20Apr93       "       Added userinterface_defs.h, word_and_phrase_defs.h and
#                        worddescriptor_defs.h.
#
#


#
#  Maintain memory of previous runs.
#

.KEEP_STATE:

#
#  The "make" command to be passed to sub-system builds.  (Passes down flags
#  which were used when invoking this top-level makefile.)
#

MAKE = make $(MFLAGS)

#
#  Sub-directories of necessary sub-systems.  (List alphabetically)
#

SUB_DIRS=  \
        construction  \
        dictionary  \
        log  \
        models  \
        session  \
        users


#
#  The expected outputs of the above listed sub-systems.  (IMPORTANT NOTE:
#       the order in which these are listed affects the order in which
#       certain static objects are created.  These are constructed roughly
#       from the bottom up in this list.  dictionary.o is listed below
#       models.o so that the dictionary is avaliable before constructing
#       basephrases, which use the dictionary to determine knowledge level.)
#

SUB_PARTS=  \
        construction/construction.o  \
        log/log.o  \
        models/models.o  \
        dictionary/dictionary.o  \
        session/session.o  \
        users/users.o

#
```



```
#  Global include files.  The GBL_HDRS macro is passed on down to the
#  sub-sytems, so this is where any include files which may be shared
#  across sub-systems are defined.  The HDRS macro is used at this level
#  for "building" these header files (basically just pulling them out of
#  configuration management.
#

GBL_HDRS=  \
        ../hdr/basephrase.h  \
        ../hdr/compoundstring.h  \
        ../hdr/dictionary.h  \
        ../hdr/hint.h  \
        ../hdr/log_defs.h  \
        ../hdr/log.h  \
        ../hdr/phrasedescriptor.h  \
        ../hdr/phrasemodellist.h  \
        ../hdr/statistics.h  \
        ../hdr/statistics_defs.h  \
        ../hdr/stdtype.h  \
        ../hdr/user_defs.h  \
        ../hdr/userinterface.h  \
        ../hdr/userinterface_defs.h  \
        ../hdr/userlist.h  \
        ../hdr/word.h  \
        ../hdr/word_and_phrase_defs.h  \
        ../hdr/worddescriptor.h  \
        ../hdr/worddescriptor_defs.h

HDRS=  \
        hdr/basephrase.h  \
        hdr/compoundstring.h  \
        hdr/dictionary.h  \
        hdr/hint.h  \
        hdr/log_defs.h  \
        hdr/log.h  \
        hdr/phrasedescriptor.h  \
        hdr/phrasemodellist.h  \
        hdr/statistics.h  \
        hdr/statistics_defs.h  \
        hdr/stdtype.h  \
        hdr/user_defs.h  \
        hdr/userinterface.h  \
        hdr/userinterface_defs.h  \
        hdr/userlist.h  \
        hdr/word.h  \
        hdr/word_and_phrase_defs.h  \
        hdr/worddescriptor.h  \
        hdr/worddescriptor_defs.h

#
#  The rules for "building" global include files.
#

hdr/%.h: SCCS/s.%.h
        #************************** Get global include file
        cd hdr; sccs -d../ get $(@F); cd ..;

##############################################################################
#
#  Actual build procedure begins here.
#
##############################################################################

all:    gbl_incs  sub_systems  glt

#
```

```
133  #  Target for updating global header files.  (No explicit action - forces the
134  #  SCCS extraction, if needed).
135  #
136
137  gbl_incs: $(HDRS)
138          #********************** Update global header files.
139
140  #
141  #  Target for updating sub-systems.
142  #
143
144  sub_systems:
145          #********************** Check each sub-system of tutor.
146          for i in $(SUB_DIRS) ; do \
147                  cd $$i ; $(MAKE) GBL_HDRS="$(GBL_HDRS)" ; cd .. ; \
148          done
149
150  #
151  #  Target for the actual tutoring system executable - "glt"
152  #
153
154  glt:  $(SUB_PARTS)
155          #********************** Build the executable - "glt"
156          CC $(SUB_PARTS) -o glt
```

```
#
#
#  The procedure for building the construction subsystem - one subsystem in the
#  German language tutor thesis program.
#
#  Modification History:
#
#  When     Who           Why
#  ----     ---           ---
#  26Mar93 Ken Staffan  Initial version.
#  06Apr93      "       Renamed phrase_modifier.h to PC_defs.h, PM_*.C to
#                       PC_*.C, and data_phrase_modifiers.C to data_PC_defs.C
#  08Apr93      "       phrase.C became compoundstring.C
#  20Apr93      "       Added utility_gender1_var.C, PC_gender2_first_word.C,
#                       utility_gender2_var.C, PC_wrong_third_word.C,
#                       PC_wrong_fourth_word.C, PC_wrong_first_and_second_
#                       word.C, PC_nouncap_second_word.C, utility_not_
#                       capitalized.C.
#  28Apr93      "       Added PC_extra_article_fourth.
#


#
#  Maintain memory of previous runs.
#

.KEEP_STATE:


#
#  Include files which may be used by source in this subsystem.  This
#  includes both the local include files, and the global include files
#  which are available to all subsystems.
#

HDRS=  \
        $(GBL_HDRS)  \
        hdr/PC_defs.h  \
        hdr/dynamicworddescriptor.h

#
#  List of object modules which are built for this subsystem.
#

OBJS=  \
        obj/PC_extra_article_fourth.o  \
        obj/PC_gender1_first_word.o  \
        obj/PC_gender2_first_word.o  \
        obj/PC_nouncap_second_word.o  \
        obj/PC_wrong_first_and_second_word.o  \
        obj/PC_wrong_first_word.o  \
        obj/PC_wrong_fourth_word.o  \
        obj/PC_wrong_second_word.o  \
        obj/PC_wrong_third_word.o  \
        obj/compoundstring.o  \
        obj/data_PC_defs.o  \
        obj/dynamicworddescriptor.o  \
        obj/phrasedescriptor.o  \
        obj/utility_gender1_var.o  \
        obj/utility_gender2_var.o  \
        obj/utility_not_capitalized.o  \
        obj/word.o  \
        obj/worddescriptor.o

#
#  Implicit build rules for source code and include files.
```



```
#

src/%.C: SCCS/s.%.C
        #***************************** Get construction source code file
        cd src; sccs -d../ get $(@F); cd ..;

hdr/%.h: SCCS/s.%.h
        #***************************** Get construction include file
        cd hdr; sccs -d../ get $(@F); cd ..;


################################################################################
##
#
#  Actual build procedure begins here.
#
################################################################################
##

construction.o:  $(OBJS)
        #***************************** Build a single construction object module
        ld -r $(OBJS) -o construction.o


$(OBJS):  src/$$(@F:.o=.C) $(HDRS)
        #***************************** Compile construction source file
        cd src; /usr/lang/SC1.0/CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C
) ;  \
        rm *..c;  \
        mv $(@F) ../obj/$(@F); cd ..


#       cd src; CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C) ;  \
#       mv $(@F:.o=..c) ../lis;  \
#       mv $(@F) ../obj/$(@F); cd ..
```

```
1     #
2     #
3     #  The procedure for building the dictionary subsystem - one subsystem in the
4     #  German language tutor thesis program.
5     #
6     #  Modification History:
7     #
8     #  When     Who          Why
9     #  ----     ---          ---
10    #  26Mar93 Ken Staffan  Initial version.
11    #  05Apr93      "       Renamed dictentry to dictionaryentry.
12    #  17Apr93      "       Added dictionary_defs.h.
13    #
14
15
16    #
17    #  Maintain memory of previous runs.
18    #
19
20    .KEEP_STATE:
21
22
23    #
24    #  Include files which may be used by source in this subsystem.  This
25    #  includes both the local include files, and the global include files
26    #  which are available to all subsystems.
27    #
28
29    HDRS=  \
30           $(GBL_HDRS)  \
31           hdr/dictionary_defs.h  \
32           hdr/dictionaryentry.h
33
34    #
35    #  List of object modules which are built for this subsystem.
36    #
37
38    OBJS=  \
39           obj/data_dictionary.o  \
40           obj/dictionary.o  \
41           obj/dictionaryentry.o
42
43    #
44    #  Implicit build rules for source code and include files.
45    #
46
47    src/%.C: SCCS/s.%.C
48           #***************************** Get dictionary source code file
49           cd src; sccs -d../ get $(@F); cd ..;
50
51    hdr/%.h: SCCS/s.%.h
52           #***************************** Get dictionary include file
53           cd hdr; sccs -d../ get $(@F); cd ..;
54
55
56    ##############################################################################
      ##
57    #
58    #  Actual build procedure begins here.
59    #
60    ##############################################################################
      ##
61
62    dictionary.o:  $(OBJS)
63           #***************************** Build a single dictionary object module
64           ld -r $(OBJS) -o dictionary.o
```



```
65
66
67    $(OBJS):  src/$$(@F:.o=.C) $(HDRS)
68           #***************************** Compile dictionary source file
69           cd src; /usr/lang/SC1.0/CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C
      ) ; \
70           rm *..c;  \
71           mv $(@F) ../obj/$(@F); cd ..
72
73
74    #      cd src; CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C) ;  \
75    #      mv $(@F:.o=..c) ../lis;  \
76    #      mv $(@F) ../obj/$(@F); cd ..
```

```
#
#
#  The procedure for building the log subsystem - one subsystem in the
#  German language tutor thesis program.
#
#  Modification History:
#
#  When     Who           Why
#  ----     ---           ---
#  26Mar93 Ken Staffan  Initial version.
#



#
#  Maintain memory of previous runs.
#

.KEEP_STATE:


#
#  Include files which may be used by source in this subsystem.  This
#  includes both the local include files, and the global include files
#  which are available to all subsystems.
#

HDRS=  \
        $(GBL_HDRS)  \
        hdr/logentry.h

#
#  List of object modules which are built for this subsystem.
#

OBJS=  \
        obj/log.o  \
        obj/logentry.o

#
#  Implicit build rules for source code and include files.
#

src/%.C: SCCS/s.%.C
        #**************************** Get log source code file
        cd src; sccs -d../ get $(@F); cd ..;

hdr/%.h: SCCS/s.%.h
        #**************************** Get log include file
        cd hdr; sccs -d../ get $(@F); cd ..;


################################################################################
##
#
#  Actual build procedure begins here.
#
################################################################################
##

log.o:  $(OBJS)
        #**************************** Build a single log object module
        ld -r $(OBJS) -o log.o


$(OBJS):  src/$$(@F:.o=.C) $(HDRS)
        #**************************** Compile log source file
```



```
        cd src; /usr/lang/SC1.0/CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C
) ; \
        rm *..c;  \
        mv $(@F) ../obj/$(@F); cd ..


#       cd src; CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C) ;  \
#       mv $(@F:.o=..c) ../lis;  \
#       mv $(@F) ../obj/$(@F); cd ..
```

```
#
#
#  The procedure for building the models subsystem - one subsystem in the
#  German language tutor thesis program.
#
#  Modification History:
#
#  When     Who           Why
#  ----     ---           ---
#  07Oct92 Ken Staffan   Initial version.
#  23Nov92       "       Added session.o, userlist.h and userlist.o.
#  27Nov92       "       Added session.h, stdtype.h, modellist.h, modellist.o,
#                        dictentry.o, dictentry.h, dictionary.o, dictionary.h,
#                        word.o, word.h, worddescriptor.o, and worddescriptor.h.
#  05Dec92       "       Added basephrase.h & .o.
#  19Dec92       "       Added basephraselist and basephrasepool.h and .o.
#  20Dec92       "       Renamed modellist* to phrasemodellist*, added
#                        data_basephrasepools.o.
#  28Dec92       "       Renamed data_basephrasepools to data_phrasemodellist,
#                        added phrasemodel.h & .C
#  30Dec92       "       Added data_dictionary.C.
#  31Dec92       "       Added phrasedescriptor.h & .C and phrase.h & .C.
#  16Jan93       "       Added generation of ..c (C code generated from C++).
#                        Takes a lot of space.  Should do on demand only.
#  17Jan93       "       Added pattern, patterntree, phrasemodifier and
#                        basephrasetrans.h & .C.  Alphabetized lists.
#  23Jan93       "       Renamed patterntree to translationtree.
#  06Feb93       "       Renamed link file from .lnk to .o for RIT CC.
#  09Feb93       "       Don't generate ..c until RIT quota problem resolved.
#  20Feb93       "       Added dynamicworddescriptor.*.
#  25Feb93       "       Added PM_*, data_phrase_modifiers.C & phrase_modifier.h
#                        Deleted phrasemodifier.h & .C
#  26Feb93       "       Added -DDEBUG to CC command.
#  28Feb93       "       Added PM_genderl_first_word.C, user.h & .C, and
#                        data_users.C
#  06-Mar-93     "       Added BPT_no_translation.C.
#  07-Mar-93     "       Added userinterface.h & .C.
#  13-Mar-93     "       Added log.h & .C, logentry .h & .C, exit_tutor.C.
#  15-Mar-93     "       Added statistics.h & .C
#  26-Mar-93     "       First cut at breaking out all other subsystems.
#  05-Apr-93     "       Renamed pattern.* to patternnode.*.
#  10-Apr-93     "       Removed obsolete basephrasetrans.h & .C
#  12-Apr-93     "       Added hint.C.
#  28-Apr-93     "       Added BPT_drop_fourth_word.C
#



#
#  Maintain memory of previous runs.
#

.KEEP_STATE:


#
#  Include files which may be used by source in this subsystem.  This
#  includes both the local include files, and the global include files
#  which are available to all subsystems.
#

HDRS=  \
        $(GBL_HDRS)  \
        hdr/basephraselist.h  \
        hdr/basephrasepool.h  \
        hdr/patternnode.h  \
        hdr/phrasemodel.h  \
        hdr/translationtree.h


#
#  List of object modules which are built for this subsystem.
#

OBJS=  \
        obj/BPT_drop_fourth_word.o  \
        obj/BPT_no_translation.o  \
        obj/basephrase.o  \
        obj/basephraselist.o  \
        obj/basephrasepool.o  \
        obj/data_phrasemodellist.o  \
        obj/hint.o  \
        obj/patternnode.o  \
        obj/phrasemodel.o  \
        obj/phrasemodellist.o  \
        obj/translationtree.o


#
#  Implicit build rules for source code and include files.
#

src/%.C: SCCS/s.%.C
        #***************************** Get models source code file
        cd src; sccs -d../ get $(@F); cd ..;

hdr/%.h: SCCS/s.%.h
        #***************************** Get models include file
        cd hdr; sccs -d../ get $(@F); cd ..;


################################################################################
##
#
#  Actual build procedure begins here.
#
################################################################################
##

models.o:  $(OBJS)
        #***************************** Build a single models object module
        ld -r $(OBJS) -o models.o


$(OBJS):  src/$$(@F:.o=.C) $(HDRS)
        #***************************** Compile models source file
        cd src; /usr/lang/SC1.0/CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C
) ;  \
        rm *..c;  \
        mv $(@F) ../obj/$(@F); cd ..


#       cd src; CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C) ;  \
#       mv $(@F:.o=..c) ../lis;  \
#       mv $(@F) ../obj/$(@F); cd ..
```

```
1     #
2     #
3     #  The procedure for building the session subsystem - one subsystem in the
4     #  German language tutor thesis program.
5     #
6     #  Modification History:
7     #
8     #  When     Who          Why
9     #  ----     ---          ---
10    #  26Mar93 Ken Staffan  Initial version.
11    #  20Apr93      "       Added statistics_defs.h
12    #  28Apr93      "       Added sys_exit_tutor().
13    #
14
15
16    #
17    #  Maintain memory of previous runs.
18    #
19
20    .KEEP_STATE:
21
22
23    #
24    #  Include files which may be used by source in this subsystem.  This
25    #  includes both the local include files, and the global include files
26    #  which are available to all subsystems.
27    #
28
29    HDRS=  \
30           $(GBL_HDRS)  \
31           hdr/session.h
32
33    #
34    #  List of object modules which are built for this subsystem.
35    #
36
37    OBJS=  \
38           obj/exit_tutor.o  \
39           obj/session.o  \
40           obj/statistics.o  \
41           obj/sys_exit_tutor.o  \
42           obj/userinterface.o
43
44    #
45    #  Implicit build rules for source code and include files.
46    #
47
48    src/%.C: SCCS/s.%.C
49           #**************************** Get session source code file
50           cd src; sccs -d../ get $(@F); cd ..;
51
52    hdr/%.h: SCCS/s.%.h
53           #**************************** Get session include file
54           cd hdr; sccs -d../ get $(@F); cd ..;
55
56
57    ##############################################################################
      ##
58    #
59    #  Actual build procedure begins here.
60    #
61    ##############################################################################
      ##
62
63    session.o:  $(OBJS)
64           #**************************** Build a single session object module
```



```
65           ld -r $(OBJS) -o session.o
66
67
68    $(OBJS):  src/$$(@F:.o=.C) $(HDRS)
69           #**************************** Compile session source file
70           cd src; /usr/lang/SC1.0/CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C
      ) ;  \
71           rm *..c;  \
72           mv $(@F) ../obj/$(@F); cd ..
73
74
75    #      cd src; CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C) ;  \
76    #      mv $(@F:.o=..c) ../lis;  \
77    #      mv $(@F) ../obj/$(@F); cd ..
```

```
1     #
2     #
3     #  The procedure for building the users subsystem - one subsystem in the
4     #  German language tutor thesis program.
5     #
6     #  Modification History:
7     #
8     #  When     Who          Why
9     #  ----     ---          ---
10    #  26Mar93 Ken Staffan  Initial version.
11    #  18Apr93      "       Removed data_users.C in favor of external user
12    #                       file.
13    #
14
15
16    #
17    #  Maintain memory of previous runs.
18    #
19
20    .KEEP_STATE:
21
22
23    #
24    #  Include files which may be used by source in this subsystem.  This
25    #  includes both the local include files, and the global include files
26    #  which are available to all subsystems.
27    #
28
29    HDRS=  \
30           $(GBL_HDRS)  \
31           hdr/user.h
32
33    #
34    #  List of object modules which are built for this subsystem.
35    #
36
37    OBJS=  \
38           obj/user.o  \
39           obj/userlist.o
40
41    #
42    #  Implicit build rules for source code and include files.
43    #
44
45    src/%.C: SCCS/s.%.C
46           #***************************** Get users source code file
47           cd src; sccs -d../ get $(@F); cd ..;
48
49    hdr/%.h: SCCS/s.%.h
50           #***************************** Get users include file
51           cd hdr; sccs -d../ get $(@F); cd ..;
52
53
54    ##############################################################################
      ##
55    #
56    #  Actual build procedure begins here.
57    #
58    ##############################################################################
      ##
59
60    users.o:  $(OBJS)
61           #***************************** Build a single users object module
62           ld -r $(OBJS) -o users.o
63
64
```



```
65    $(OBJS):  src/$$(@F:.o=.C) $(HDRS)
66           #***************************** Compile users source file
67           cd src; /usr/lang/SC1.0/CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C
      ) ;  \
68           rm *..c;  \
69           mv $(@F) ../obj/$(@F); cd ..
70
71
72    #      cd src; CC -DDEBUG +i -c -I../hdr -I../../hdr $(@F:.o=.C) ;  \
73    #      mv $(@F:.o=..c) ../lis;  \
74    #      mv $(@F) ../obj/$(@F); cd ..
```

```
1   ##############################################################################
2   #    German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993
3   ##############################################################################
4   #
5   #       baseline_all
6   #
7   #  This script could be destructive, so the setup must be done manually
8   #  before running this script.  The procedure would be:
9   #
10  #       - Execute the checkout_all script to extract all files.
11  #       - Remove all files from the /SCCS subdirectories.
12  #       - Run this script to re-create baseline versions of all source.
13  #       - If it appears that the script executed successfully, execute
14  #               the cleanup_all script to remove all ,* (note leading
15  #               comma) files left by SCCS, and all .h and .C in the
16  #               directories which have comma-files.
17  #
18  #  Modification History:
19  #
20  #  Revisions    By              Reason
21  #  ---------    --              ------
22  #  26Apr93      Ken Staffan     v1.0 release with final thesis report.
23  #
24
25  #
26  # Create top-level .h files, scripts, and build procedures.
27  #
28
29  sccs create READ_ME
30  sccs get READ_ME
31
32  sccs create users.data
33  sccs get users.data
34
35  sccs create *.h
36
37  sccs create makefile
38  sccs get makefile
39  chmod 700 makefile
40
41  sccs create split
42  sccs get split
43  chmod 700 split
44
45  sccs create baseline_all
46  sccs get baseline_all
47  chmod 700 baseline_all
48
49  sccs create checkin_all
50  sccs get checkin_all
51  chmod 700 checkin_all
52
53  sccs create checkout_all
54  sccs get checkout_all
55  chmod 700 checkout_all
56
57  sccs create print_all
58  sccs get print_all
59  chmod 700 print_all
60
61  sccs create cleanup_all
62  sccs get cleanup_all
63  chmod 700 cleanup_all
64
65  #
66  # Change to each component sub-directory, and create .h and .C files,
```



```
67  # and build procedures.
68  #
69
70  cd construction
71  sccs create *.h
72  sccs create *.C
73  sccs create makefile
74  sccs get makefile
75  chmod 700 makefile
76
77  cd ../dictionary
78  sccs create *.h
79  sccs create *.C
80  sccs create makefile
81  sccs get makefile
82  chmod 700 makefile
83
84  cd ../log
85  sccs create *.h
86  sccs create *.C
87  sccs create makefile
88  sccs get makefile
89  chmod 700 makefile
90
91  cd ../models
92  sccs create *.h
93  sccs create *.C
94  sccs create makefile
95  sccs get makefile
96  chmod 700 makefile
97
98  cd ../session
99  sccs create *.h
100 sccs create *.C
101 sccs create makefile
102 sccs get makefile
103 chmod 700 makefile
104
105 cd ../users
106 sccs create *.h
107 sccs create *.C
108 sccs create makefile
109 sccs get makefile
110 chmod 700 makefile
111
112 cd ..
```

```
1   ##############################################################################
2   #   German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993
3   ##############################################################################
4   #
5   #       checkin_all
6   #
7   #  This script will replace all source files which were extracted for
8   #  change by "checkout_all".  The user will be prompted for the SCCS
9   #  comment.  (A future enhancement could be to have the script ask once,
10  #  and feed the comment into SCCS).  This script does not need to be
11  #  updated when source files are added or deleted (though it will fail
12  #  if new source files exist which have not yet been "sccs create"d.
13  #  It assumes that all of the files are ih the directory above their
14  #  respective /SCCS directory.
15  #
16  #  Modification History:
17  #
18  #  Revisions    By              Reason
19  #  ---------    --              ------
20  #  26Apr93      Ken Staffan     v1.0 release with final thesis report.
21  #
22
23  #
24  # Replace the top-level .h files, scripts, and build procedures.
25  #
26
27  sccs delta *.h
28
29  sccs delta READ_ME
30  sccs get READ_ME
31
32  sccs delta users.data
33  sccs get users.data
34
35  sccs delta makefile
36  sccs get makefile
37  chmod 700 makefile
38
39  sccs delta split
40  sccs get split
41  chmod 700 split
42
43  sccs delta baseline_all
44  sccs get baseline_all
45  chmod 700 baseline_all
46
47  sccs delta checkin_all
48  sccs get checkin_all
49  chmod 700 checkin_all
50
51  sccs delta checkout_all
52  sccs get checkout_all
53  chmod 700 checkout_all
54
55  sccs delta print_all
56  sccs get print_all
57  chmod 700 print_all
58
59  sccs delta cleanup_all
60  sccs get cleanup_all
61  chmod 700 cleanup_all
62
63  #
64  # Change to each component sub-directory, and replace .h and .C files and
65  # build procedures.
66  #
```



```
67
68  cd construction
69  sccs delta *.h
70  sccs delta *.C
71  sccs delta makefile
72  sccs get makefile
73  chmod 700 makefile
74
75  cd ../dictionary
76  sccs delta *.h
77  sccs delta *.C
78  sccs delta makefile
79  sccs get makefile
80  chmod 700 makefile
81
82  cd ../log
83  sccs delta *.h
84  sccs delta *.C
85  sccs delta makefile
86  sccs get makefile
87  chmod 700 makefile
88
89  cd ../models
90  sccs delta *.h
91  sccs delta *.C
92  sccs delta makefile
93  sccs get makefile
94  chmod 700 makefile
95
96  cd ../session
97  sccs delta *.h
98  sccs delta *.C
99  sccs delta makefile
100 sccs get makefile
101 chmod 700 makefile
102
103 cd ../users
104 sccs delta *.h
105 sccs delta *.C
106 sccs delta makefile
107 sccs get makefile
108 chmod 700 makefile
109
110 cd ..
```

```
1   ##############################################################################
2   #   German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993
3   ##############################################################################
4   #
5   #       checkout_all
6   #
7   #  This script will extract all source files (top-level .h, and component-
8   #  level .C) and all build procedures and scripts, from SCCS for edit
9   #  (change).  There is no equivalent script
10  #  to get read-only versions, since these will normally be left in the
11  #  various /hdr and /src subdirectories after a build.  This will leave
12  #  the files in the directory above the /SCCS directory from which they
13  #  were extracted.  The script is intended to be run from the top-level
14  #  directory, and will need to updated if any source files are added or
15  #  deleted.
16  #
17  #  NOTE that the script first deletes any versions in the directories,
18  #  so care should be taken not to invoke this script while individual
19  #  files are checked out (it will overwrite them).
20  #
21  #  Modification History:
22  #
23  #  Revisions    By              Reason
24  #  ---------    --              ------
25  #  26Apr93      Ken Staffan     v1.0 release with final thesis report.
26  #
27
28  #
29  # Extract the top-level .h files, scripts, and build procedures.
30  #
31
32  rm -f READ_ME
33  rm -f users.data
34  rm -f makefile
35  rm -f split
36  rm -f baseline_all
37  rm -f checkin_all
38  rm -f checkout_all
39  rm -f print_all
40  rm -f cleanup_all
41  rm -f *.h
42  rm -f *.C
43
44  sccs edit READ_ME
45  sccs edit users.data
46  sccs edit makefile
47  chmod 700 makefile
48  sccs edit split
49  chmod 700 split
50  sccs edit baseline_all
51  chmod 700 baseline_all
52  sccs edit checkin_all
53  chmod 700 checkin_all
54  sccs edit checkout_all
55  chmod 700 checkout_all
56  sccs edit print_all
57  chmod 700 print_all
58  sccs edit cleanup_all
59  chmod 700 cleanup_all
60
61  sccs edit basephrase.h
62  sccs edit compoundstring.h
63  sccs edit dictionary.h
64  sccs edit hint.h
65  sccs edit log.h
66  sccs edit log_defs.h
```



```
67  sccs edit phrasedescriptor.h
68  sccs edit phrasemodellist.h
69  sccs edit statistics.h
70  sccs edit statistics_defs.h
71  sccs edit stdtype.h
72  sccs edit user_defs.h
73  sccs edit userinterface.h
74  sccs edit userinterface_defs.h
75  sccs edit userlist.h
76  sccs edit word.h
77  sccs edit word_and_phrase_defs.h
78  sccs edit worddescriptor.h
79  sccs edit worddescriptor_defs.h
80
81  #
82  # Change to each component sub-directory, and extract .h and .C files.
83  #
84
85  cd construction
86  rm -f makefile
87  rm -f *.h
88  rm -f *.C
89  sccs edit makefile
90  sccs edit PC_defs.h
91  sccs edit dynamicworddescriptor.h
92  sccs edit PC_extra_article_fourth.C
93  sccs edit PC_gender1_first_word.C
94  sccs edit PC_gender2_first_word.C
95  sccs edit PC_nouncap_second_word.C
96  sccs edit PC_wrong_first_and_second_word.C
97  sccs edit PC_wrong_first_word.C
98  sccs edit PC_wrong_fourth_word.C
99  sccs edit PC_wrong_second_word.C
100 sccs edit PC_wrong_third_word.C
101 sccs edit compoundstring.C
102 sccs edit data_PC_defs.C
103 sccs edit dynamicworddescriptor.C
104 sccs edit phrasedescriptor.C
105 sccs edit utility_gender1_var.C
106 sccs edit utility_gender2_var.C
107 sccs edit utility_not_capitalized.C
108 sccs edit word.C
109 sccs edit worddescriptor.C
110 cd ../dictionary
111 rm -f makefile
112 rm -f *.h
113 rm -f *.C
114 sccs edit makefile
115 sccs edit dictionary_defs.h
116 sccs edit dictionaryentry.h
117 sccs edit data_dictionary.C
118 sccs edit dictionary.C
119 sccs edit dictionaryentry.C
120 cd ../log
121 rm -f makefile
122 rm -f *.h
123 rm -f *.C
124 sccs edit makefile
125 sccs edit logentry.h
126 sccs edit log.C
127 sccs edit logentry.C
128 cd ../models
129 rm -f makefile
130 rm -f *.h
131 rm -f *.C
132 sccs edit makefile
```

```
133   sccs edit basephraselist.h
134   sccs edit basephrasepool.h
135   sccs edit patternnode.h
136   sccs edit phrasemodel.h
137   sccs edit translationtree.h
138   sccs edit BPT_drop_fourth_word.C
139   sccs edit BPT_no_translation.C
140   sccs edit basephrase.C
141   sccs edit basephraselist.C
142   sccs edit basephrasepool.C
143   sccs edit data_phrasemodellist.C
144   sccs edit hint.C
145   sccs edit patternnode.C
146   sccs edit phrasemodel.C
147   sccs edit phrasemodellist.C
148   sccs edit translationtree.C
149   cd ../session
150   rm -f makefile
151   rm -f *.h
152   rm -f *.C
153   sccs edit makefile
154   sccs edit session.h
155   sccs edit exit_tutor.C
156   sccs edit session.C
157   sccs edit statistics.C
158   sccs edit sys_exit_tutor.C
159   sccs edit userinterface.C
160   cd ../users
161   rm -f makefile
162   rm -f *.h
163   rm -f *.C
164   sccs edit makefile
165   sccs edit user.h
166   sccs edit user.C
167   sccs edit userlist.C
168   cd ..
```

```
1   ##############################################################################
2   #   German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993
3   ##############################################################################
4   #
5   #       cleanup_all
6   #
7   #  This script is only intended to be executed after a successful execution
8   #  of baseline_all.  It cleans up the SCCS create residual files.
9   #
10  #  Modification History:
11  #
12  #  Revisions    By              Reason
13  #  ---------    --              ------
14  #  26Apr93      Ken Staffan     v1.0 release with final thesis report.
15  #
16
17  #
18  #  Remove top-level residual files.
19  #
20
21  rm -f ,*
22  rm -f *.h
23
24  #
25  #  Remove component-level residual files.
26  #
27
28  cd construction
29  rm -f ,*
30  rm -f *.h
31  rm -f *.C
32  cd ../dictionary
33  rm -f ,*
34  rm -f *.h
35  rm -f *.C
36  cd ../log
37  rm -f ,*
38  rm -f *.h
39  rm -f *.C
40  cd ../models
41  rm -f ,*
42  rm -f *.h
43  rm -f *.C
44  cd ../session
45  rm -f ,*
46  rm -f *.h
47  rm -f *.C
48  cd ../users
49  rm -f ,*
50  rm -f *.h
51  rm -f *.C
52  cd ..
```

```
#############################################################################
#   German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993
#############################################################################
#
#       print_all
#
#  This script prints all source code, scripts and build procedures in
#  the desired listing order.  Note that the printing command may need
#  change on a different machine.
#
#  Modification History:
#
#  Revisions    By              Reason
#  ---------    --              ------
#  26Apr93      Ken Staffan     v1.0 release with final thesis report.
#


#
#  Print all include files, in alphabetical order.
#

a2ps  \
        construction/hdr/PC_defs.h  \
        hdr/basephrase.h  \
        models/hdr/basephraselist.h  \
        models/hdr/basephrasepool.h  \
        hdr/compoundstring.h  \
        hdr/dictionary.h  \
        dictionary/hdr/dictionary_defs.h  \
        dictionary/hdr/dictionaryentry.h  \
        construction/hdr/dynamicworddescriptor.n  \
        hdr/hint.h  \
        hdr/log.h  \
        hdr/log_defs.h  \
        log/hdr/logentry.h  \
        models/hdr/patternnode.h  \
        hdr/phrasedescriptor.h  \
        models/hdr/phrasemodel.h  \
        hdr/phrasemodellist.h  \
        session/hdr/session.h  \
        hdr/statistics.h  \
        hdr/statistics_defs.h  \
        hdr/stdtype.h  \
        models/hdr/translationtree.h  \
        users/hdr/user.h  \
        hdr/user_defs.h  \
        hdr/userinterface.h  \
        hdr/userinterface_defs.h  \
        hdr/userlist.h  \
        hdr/word.h  \
        hdr/word_and_phrase_defs.h  \
        hdr/worddescriptor.h  \
        hdr/worddescriptor_defs.h  \
| lpr -Pgates

#
#  Print all source files, in alphabetical order.
#

a2ps \
        models/src/BPT_drop_fourth_word.C  \
        models/src/BPT_no_translation.C  \
        construction/src/PC_extra_article_fourth.C  \
        construction/src/PC_gender1_first_word.C  \
        construction/src/PC_gender2_first_word.C  \
        construction/src/PC_nouncap_second_word.C  \
```



```
        construction/src/PC_wrong_first_and_second_word.C  \
        construction/src/PC_wrong_first_word.C  \
        construction/src/PC_wrong_fourth_word.C  \
        construction/src/PC_wrong_second_word.C  \
        construction/src/PC_wrong_third_word.C  \
        models/src/basephrase.C  \
        models/src/basephraselist.C  \
        models/src/basephrasepool.C  \
        construction/src/compoundstring.C  \
        construction/src/data_PC_defs.C  \
        dictionary/src/data_dictionary.C  \
        models/src/data_phrasemodellist.C  \
        dictionary/src/dictionary.C  \
        dictionary/src/dictionaryentry.C  \
        construction/src/dynamicworddescriptor.C  \
        session/src/exit_tutor.C  \
        models/src/hint.C  \
        log/src/log.C  \
        log/src/logentry.C  \
        models/src/patternnode.C  \
        construction/src/phrasedescriptor.C  \
        models/src/phrasemodel.C  \
        models/src/phrasemodellist.C  \
        session/src/session.C  \
        session/src/statistics.C  \
        session/src/sys_exit_tutor.C  \
        models/src/translationtree.C  \
        users/src/user.C  \
        session/src/userinterface.C  \
        users/src/userlist.C  \
        construction/src/utility_gender1_var.C  \
        construction/src/utility_gender2_var.C  \
        construction/src/utility_not_capitalized.C  \
        construction/src/word.C  \
        construction/src/worddescriptor.C  \
| lpr -Pgates

#
#  Print all scripts and build procedures.
#

a2ps  \
        READ_ME  \
        makefile  \
        construction/makefile  \
        dictionary/makefile  \
        log/makefile  \
        models/makefile  \
        session/makefile  \
        users/makefile  \
        baseline_all  \
        checkin_all  \
        checkout_all  \
        cleanup_all  \
        print_all  \
        split  \
        users.data  \
| lpr -Pgates
```

```
1   ##############################################################################
2   #   German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993
3   ##############################################################################
4   #
5   #       split
6   #
7   #  This script allows the system executable to be run, redirecting any
8   #  debug or error output to the file debug.log.  (Leaving the standard
9   #  user interface displaying to the screen.)
10  #
11  #  Modification History:
12  #
13  #  Revisions     By              Reason
14  #  ---------     --              ------
15  #  26Apr93       Ken Staffan     v1.0 release with final thesis report.
16  #
17
18  sh -c "glt 2> debug.log"
```

```
1       1
2       6
3       stu_user
4       RealName
5       2
6       0
7       sys_user
8       RealName
```