1989

Detecting errors in software using a parameter checker: An Analysis

John Sexton A.

Rochester Institute of Technology
Computer Science Department

Detecting Errors in Software Using a Parameter Checker: An Analysis

by

John A. Sexton

A thesis, submitted to
The Faculty of the Computer Science Department,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Prof. Jeffrey Lasky

Prof. Peter G. Anderson

Prof. John A. Biles

April 28, 1989

Trademark Acknowledgments:

80286 and iRMX-86 are trademarks of the Intel Corporation.

Detecting Errors in Software Using a
Parameter Checker: An Analysis

Table of Contents (continued)

Table of Contents (continued)

Figures:

Tables:

Abstract

This paper will discuss a study of parameter errors that occurred in a software product developed using the "C" language. Through the use of a parameter checker that was developed from an existing compiler, the frequency and type of parameter errors over several releases of an application are studied. In addition to parameter checking, other static analysis data was collected concerning function usage (or non-usage) and function return value comparisons. This paper may serve as a useful complement to previous static analyses done for the FORTRAN (Knuth - 1971), COBOL (Jarrah - 1979) and Pascal (Brookes - 1982) languages.

The study presents statistics that show the large number of parameter errors caused by the weak type-checking of the "C" language. This paper also discusses the cost associated with detecting and repairing parameter errors that might otherwise have been detected by static analysis methods.

In summing up the presence of these parameter errors the author discusses the impact of the ANSI "C" standard on this issue.

The code used to build the data analysis portion of the Parameter Checker is presented in the appendices.

ACM Computing Review Categories and Subject Descriptors:

D.2.5 [Software Engineering]: Testing and Debugging
- debugging aids

D.2.9 [Software Engineering]: Management
- Software Quality Assurance

D.3.4 [Software Engineering]: Processors
- compilers

General Terms:

Compilers, Static Analysis

Additional Key Words and Phrases:

Parameter checking, type checking, software quality, C language, cross referencing.

1.0 Introduction

The processing of software for the detection of defects can take place in a "static" or "dynamic" manner. Static error detection tools are programs that act on a set of source code and test for violations of syntax or for violations of a user defined set of rules. Common examples of static detection tools are compilers, a UNIX "lint" utility, cross-reference mappers, and programs that present the structure of software routines (function hierarchies). Dynamic analyses are performed when the program is running. To collect the data for a dynamic analysis requires additional software or hardware. Usually the dynamic analysis is concerned with performance metrics.

Most languages supply a method for partioning source code into more than one module via the use of subroutines. The use of subroutines implies that the calling function and the called function must have a matched set of parameter lists. However, the degree to which these parameter lists are checked varies depending on the language which is used. Languages such as Ada and Pascal have made strong type-checking of parameters an integral part of the language syntax.

Languages such as "C" and FORTRAN are weakly type-checked languages. They do not provide for a rigorous checking of parameter lists. In the case of the "C" language, some compilers will often coerce a parameter into another class whether the user is aware of it or not.

When developing and maintaining a large application in the "C" language over a period of years, the number of parameter lists that are created, modified or deleted increases with time. Each one of these parameter lists can be the source of potential errors if it is

not matched with its corresponding partner correctly. Since the "C" language does not provide a method for verifying the proper types of parameters, a parameter checker was developed to address this problem.

1.1 Problem Statement.

This thesis will determine the effectiveness of using a parameter checking program to detect possible errors in software. The problem of parameter checking over separate modules is one that should be addressed whenever weakly type-checked languages are used. As Perelgut stated during his development of the Turing Plus language, "This lack of restrictions and language-supported checking, can make "C" code unreadable and unmaintainable." (1986). All of Johnson's modifications to Smalltalk for building an optimizing compiler, revolved around increasing the type-correctness of variables (1986).

In fact most enhancements to language compilers required cross-module parameter type-checking to be implemented. Gantenbein extended the "C" language to allow dynamic binding of subroutines by storing parameter information similar to the data required for the parameter checker of this paper (1987). In order to give Pascal the feature of linking separate modules, Kieburtz defined the parameters in a way similar to that defined for the current proposed ANSI "C" standard (1978).

The effectiveness of this study's approach in this study will be measured in terms of person-days that could have been saved with the use of a parameter checking program. Of special interest will be the study of error detection as the software proceeds

through several releases in its life-cycle.

1.2 Reasons for the Project.

As the advances made in computer hardware continue at a tremendous rate, the attempt of software developers to keep pace has resulted in a rapid rise in the complexity of applications. This increased complexity has meant that only the simplest of programs can be handled by a single person and that the typical application will consist of a multi-person development team writing a large number of source modules.

Such was the case of the applications development that is studied in this paper. The software for this project was contributed by eight people over approximately three years and as such lacked a cohesive method for insuring the correct use of functions.

When using a weakly typed language such as "C", the occurrence of function parameter mismatches becomes a common source of programming errors. This is especially true when functions and modules are being assembled from a number of team members.

Although a shortcoming of the "C" language (weak type-checking) directly leads to more programming errors, this is more than out-weighed by its advantages. "What is good about C? C is popular because it is flexible, portable, efficient and available" (Stroustrup - 1986).

Aside from building a parameter checker, other steps have been taken to bolster the weaknesses of "C". The development of the C++ language addresses the type-checking area as well as other

short comings of the language (Stroustrup - 1986). The addition of prototyping to the ANSI standard is an enhancement to the compilation process that will address parameter mismatches (see section 5.2).

Most of the mismatches were only discovered during a debugging session. Indeed, the debugging session was often terminated at the discovery of one of these bugs because the error caused the machine environment to crash. Only after completing a cycle of editing, linking and reloading the debugging environment, was testing able to continue. The occurrence of many of these errors significantly reduced the available product development resources and resulted in a considerable amount of lost time in software development. "Each defect that is directly exposed is one that will not have to be made visible through the process of dynamic testing, or worse, remain latent through all of testing only to manifest itself during the operational life of the software" (Dunn - 1984).

Given that the compiler had to build code for the function calls, it seemed that it should also be able to detect such obvious type-clash errors among parameters if all the source code was available. As noted by Brookes (1982), "a tool to do this cross module type-checking can be developed in one of two ways. A specific syntax analyzer can be written or, modifications can be made to an existing compiler". Performing modifications to an existing compiler also yields the added benefit of providing the user with the necessary syntactic and lexical checking for the language. Since the source code for the "C" compiler for RMX was available, the effort to modify it

into a parameter checker was undertaken.

In addition to finding these parameter mismatches, a tool that could check the correctness of parameters across module boundaries could possibly detect errors that would otherwise slip through the testing phase. This tool could be of use in both the development cycle and as a final quality check prior to release of a product.

1.3 Project Goals.

Using documents and source code from several releases of an application, an error history will be generated and answers to the following questions will be formulated.

This thesis addresses itself towards two areas, first as a quantitative measure of parameter errors in a "C" language environment and second, as a measure of poor parameter checking on a "real" application through several cycles in its life-cycle.

Concerning the topic of developing with the "C" language:

A) In a qualitative sense, what do the findings show about developing applications in the "C" environment.

B) In a quantitative sense, what do the findings show about developing applications in the "C" environment.

C) How does the "C" language affect the way software is (or should) be developed?

Concerning the results of this thesis as it applies to the quality of the specific application that was studied:

A) Have any of the problems reported on previous releases been directly attributable to a parameter error that

could have been caught by a parameter checker utility?

B) If there were undetected parameter errors that caused a problem, what was the cost to fix those errors? What would have been the cost to fix them prior to releasing the product?

C) Are there still parameter errors in the application that could cause future problems?

2. Type Checking Overview

Type checking involves the detection of any instance where an operand and an operator of a language are used together incorrectly.

Strongly typed languages have very strict rules as to what combinations of operands and operators are allowed and they do not automatically adjust the operands to suit the equation. Conversely, weakly typed languages allow a more liberal mixing of operands and operators. These languages usually accomplish this by modifying the operands along certain guidelines to fit the existing rules.

"A language is strongly typed if its compiler can guarantee that the programs it accepts will execute without type errors." (Aho, et al, 1986).

Type checking exists to some extent in all compilers. Compilers perform a number of syntactical and semantic checks on their input languages. Among these static checks lies the realm of type checking.

2.1 Issues in Typing

2.1.1 Time of Type Checking

Most type checking and parameter checking occurs between the parsing phase and the intermediate code generation phase of a compiler. If these phases happen in a single pass then checking still occurs between these two phases for a given line. A one-pass compiler for these two phases will have restrictions as to how much checking can be done (i.e. all symbols must be defined prior to

use).

2.1.2 Overloading

Some operators can be used with a variety of types. These operators are said to be "overloaded". For example, the arithmetic operators (+, -, *, /) can all be used with different operands. The presence and type of the operands indicates how the operator will be applied.

2.1.3 Coercion

One way of achieving overloading is to force a change in an operands type so that they can be mixed with other operands. This forced conversion of operands is called coercion.

Coercion is implicit if it is done automatically by the compiler and explicit if the programmer must specify it. Implicit coercions are limited to changing a variables representation but not its value (i.e. a variable should not be truncated in order to use it with another variable; the second variable should be lengthened).

Strongly typed languages are by definition limited to explicit conversions of type.

2.1.4 Polymorphism

Polymorphism is the ability to define a piece of code that will perform the its statements on a variety of operand types. This is something like overloading an

entire function. In Ada, the "generic" function serves as a polymorphic function.

2.2 Parameter Typing in Languages

2.2.1 Introduction

A subset of the realm of type checking is parameter type checking. This is the set of rules that apply to variables passed as parameters to subroutines or functions. The type checking that goes on at this level involves the comparison of the subroutine invocation and the subroutine definition. In addition, unique checks need to be made to guarantee that both sides of the comparison have the proper number of parameters.

"Languages, such as Pascal and Ada, have been designed with features such as strong typing to increase the amount of error detection that can be done by the compiler" (Ottenstein, 1983).

The tradeoff in all languages is the degree of freedom of expression allowed to the programmer. How much should you have to explicitly specify and how much should you allow the compiler to manipulate?

2.2.2 Languages with strongly typed parameters

Languages that are strongly typed have two major advantages over weakly typed languages. First, the programmer must work at a closer conceptual distance between the problem definition and its implementation.

This closeness reduces the ambiguity and variability that can result from a compiler driven translation. Second, the compiler has more information available to conduct further checks on the validity of the program.

Ada controls the use of parameters by making programmers specify an interface for all modules. These interface functions are then kept in program libraries for re-use. The invocation of a program library function will cause an interface check of both the parameters and result. Ada even takes this feature one further level and provides a method of grouping like functions together into "packages". Use of a routine in a package, will automatically make all the needed function interfaces available.

Pascal enforces parameter checking by requiring the compilation of all source code at one time and the definition of all symbols before they are used. This forces the user to make all symbol information available to the compiler for verification upon usage.

2.2.3 Languages with weakly typed parameters

The C programming language has historically taken the opposite view of the strongly type languages. C does not rely on the collection or availability of symbol data to perform type checking. Instead, C handles the process of type checking by following certain rules to combine symbols of mixed types into expressions that are valid. One problem with this is that it does not

protect the programmer from mismatched parameters. Since the definition and invocation of a subroutine may not be available at the same time, only rudimentary changes in the parameters can be made and these do not guarantee a match. Subroutine result mismatches could also occur for the same reasons. In fact, a mechanism of function declarations was provided to solve just this problem.

2.3 Type-checking in Reliability and Maintenance of Software

2.3.1 Affect on the Edit-Compile-Link-Debug Cycle

The affects of strong type checking on the time it takes to do a software turnaround cycle can be misleading. The programmer usually notices (and complains about) the increase in compile time that is required. The increased checking that takes place discovers more errors in the front-end of the cycle and causes more iterations at the front-end of the cycle. However, the time spent finding errors at this point is more than repaid by the reduced time spent doing time-expensive debugging.

2.3.2 Long term code quality

The benefits of strong typing extend beyond the realm of mixed types and parameter mismatches. This feature allows the programmer to define operations in a meaningful way.

2.3.3 Software Re-use

Since all strongly typed languages require some method for preserving symbol information between functions, the concept of software reuse is greatly enhanced. With the exception of Pascal implementations that require all code in one module, these preserved interface definitions insure a consistent and productive use of existing code.

2.4 Research Extensions to Typing

Some interesting research has been done recently in extending computer languages by broadening or extending the use of type checking to non-traditional areas.

2.4.1 Authorization Dimension (Gerard, 1988)

Languages are strongly typed in order to insure that memory locations are used in a consistent manner. This is called their structural dimension. An authorization dimension would impose rules for the use of this consistent memory.

An memory location would be subject to four authorizations: Read (R), Write (W), Execute (E) and Structure (S). Read permission would allow the contents of an identifier to be examined. Write permission would allow the contents of an identifier to be modified. Execute permission would allow execution of an identifier (this is almost always set). Structure permission would allow the internal structure of an identifier to be examined.

All of the work needed to implement this added authorization checking could occur at compile time. Each variable would be assigned an authorization set at compile time and this set would be enforced for the scope of the variable.

Variables passed to subroutines could have there authorization set passed with them or decreased (never increased). This method could be used to limit the range of variable usage across many subroutines. The declaration of read-only and write-only variables would enable the compiler to catch many additional errors with static checking.

The benefits of declaring the use of a variable adds another valid set of type checking that can be implemented.

2.4.2 Dynamic Binding in Strongly Typed Languages (Gantenbein, 1987)

Dynamic binding is usually found only in weakly typed languages. This paper describes the effort to bring run time binding with strong type checking to languages that otherwise do not support the mix of these features (C and Pascal).

Languages that do support dynamic binding often impose a significant performance penalty due to their run-time type checking. The alternative presented here involves the use of two language components: a compiler that

produces intermediate code and an interpreter the assembles, loads, and runs the code. As in most dynamically binding systems, the subroutine declaration/ invocation is the level at which the checking is performed.

Not all subroutines are dynamically linked. Static linking of subroutines is assumed, unless specified in a given source module (i.e. extern abc()). In the first phase, source modules are passed through the compiler and an interface files are generated for invocations of subroutines that have been declared as being dynamic. This interface file contains all necessary information to allow type checking of all symbols when the subroutine is finally defined.

The interpreter achieves dynamic binding by loading the intermediate code into an internal code segment for assembly and execution. Within in this internal segment if a call to a dynamically bound routine is detected, the interface file is used to resolve the remaining run-time type checking (i.e parameters and in the case of the C language, return values).

The method of data collection described in this article is similar to the method used in the parameter checker of this paper.

2.4.3 Physical Units (Manner, 1986)

A proposal has been made to extend the type checking of computer languages into the realm of physical units

(i.e. meters, seconds, grams, etc.). This was proposed in order to capitalize on the ability of language compilers to allow the specification of usage constraints on program operators. "The more constraints a language allows you to declare, the more coding errors can be avoided during program development...".

Programming languages in use today are not as explicit as those used in physical calculations. In physical expressions most quantities and explicit values are expressed with certain unit types.

The language extensions are based upon some simple rules. First, physical quantities have unique dimensions (a.k.a. types; i.e. length, mass, volts, etc.). Second, only quantities of the same dimension can be added and subtracted. However, there are no restrictions for multiplication and division (in order to create new units). Third, most functions can be called with very limited types of arguments. This allows the dimensions of parameters to be checked upon called to functions. Functions that are not limited in their arguments (i.e. sqrt) could be declared dimensionless. These rules form the basis for a set of checks (addition, subtraction, assignments and subroutine parameters of unlike dimensions) that could be performed on physical quantities if the typing information for dimensions could be defined and preserved.

To measure a quantity of a certain dimension, units (objects) are used. The explicit declaration of units and their ranges allows several additional rules to enforced at compile time.

First, while conversions between dimensions is not allowed, conversion between units of the same dimension is just a matter of scaling (i.e. Fahrenheit to Celsius). Second, new quantities of a specific unit will automatically be assigned the appropriate dimension. Third, the ranges specified for each unit can be used to guarantee the appropriate internal machine representation for a quantity. The requirement of a range can help eliminate the poor habit of using machine representation limits as boundaries.

2.5 Summary

The topic of type checking is still active in the research world and is being pursued in a number of directions. Enhancements to languages are being made in order to broaden the scope of type checking by adding new definition parameters to variables. Extending the role of type checking across module boundaries, both staticly and dynamicly, also provides new tools for reducing errors.

With the emergence of strongly type-checked Ada, and the increasing desire for reduced software life-cycle costs, it seems that we are moving down a path of increased specification for symbols within programming languages.

3.0 Project Description

3.1 Description of Input Data

The source code for the application will be put through a parameter checker. The output will then be combined with a historical data base of the product's performance to pinpoint the causes of known problems. The products historical data base is comprised of several paper-based forms.

3.1.1 Application Software

The application that will be studied is a multi-job, multi-tasking program written in the 'C' language. The application runs on an Intel 80286 based micro-computer under the Intel iRMX-86 real-time operating system.

The package consists of two major types of programs. The first type is a multi-tasking job that runs in a background mode processing transactions received from a network. There can be from one to three jobs running in the background at any one time. The second type is a multi-tasking job that runs in the foreground handling the user interface and/or various system administrative duties. There is always one foreground job running that acts as a parent process. It can load and invoke any one of six different jobs as a child process.

3.1.2 Trouble Reports

These are reports that are used by in-house personnel to record and track problems that are found in field

locations. These reports are initiated by field software support personnel. They provide a description of the problem and as much other data as can be obtained while observing the problem system. The descriptions are usually non-technical in nature and are used as a general starting point for in-house debugging work.

3.1.3 Observation Reports

These are reports that are used by in-house Software Quality Assurance (SQA) personnel to report software errors or product deviations from specification. These reports are obtained by putting the software through a test plan that has been developed to exercise all the features and functions of the product. All product versions were subjected to this test plan prior to final release.

3.1.4 Software Change Forms (SCF)

These forms are used to document the changes to source code needed to fix items reported in Observation or Trouble Reports. Included in the paperwork is the name of all changed modules and the affected executable programs. A SCF may or may not include annotated listings of software modifications.

3.2 The Parameter Checker Tool

3.2.1 Introduction

The 'C' language was developed with the idea of giving

the programmer a wide range of flexibility when writing code. The flexibility came at the price of not implementing many of the safeguards found in other languages, principally, type checking and parameter checking. Protection against these types of errors lies mainly with the programmer.

The range of 'C' compiler syntax checking runs from very poor to good. At best, the compilers usually limit themselves to detecting single line, common coding mistakes. The idea of having checks between multiple functions is usually limited to return values used in assignments. Even this simple checking is restricted to functions that were declared in a common module (file).

In the UNIX environment, the LINT program was written to supplement the checking of the "C" compiler. The UNIX "C" compiler's error checking is confined to source lines that are uninterpretable according to the "C" language syntax. LINT's main function was to perform additional checks outside the normal syntax checks. LINT flags errors that would not necessarily prevent code generation but are considered unusual use of the language. It also detects errors that may cause problems if ported to another environment, or are just poor programming practices.

A major restriction of LINT is that it is limited to working on a single file at a time. This restriction prevents it from addressing the problem of parameter

checking across multiple module boundaries. With its current wide range of language checking, parameter checking would seem like a logical extension of LINT's features.

In the Intel RMX environment, a tool with a LINT-like capability was not available. Thus, the parameter checking work required modifying the available "C" compiler source code to achieve the desired purpose.

3.2.2 Scope of the Parameter Checker

The parameter checker program was developed to overcome limitations of current compilers in the area of argument checking. The program was written to be used in checking multiple files for defects that have one of the following scopes:

a) within a single function
b) within a single module (file)
c) across modules

3.2.3 Types of Errors Detected

3.2.3.1 'C' Syntax Errors

These errors are caused by a violation of the 'C' programming language syntax rules and found by all valid 'C' compilers as they process individual files.

3.2.3.2 Function Definition vs. Return Errors

These errors arise when the variable type used by the 'return' statements within a function do not match the return definition of the function.

In the example in figure 3.1, most compilers would convert the integer variable to a pointer and return the value without complaining. However, the value returned would in most cases usually result in a pointer that was meaningless.

(Figure 3.1)

Example:

```
char *transform ()
{
    int status;
    .
    .   /* ERROR: function defined to   */
    .   /*        return a char pointer */
    return (status);
}
```

3.2.3.3 Parameter Errors

Finding parameter errors requires that the function definition and invocation information be maintained across multiple source modules. Possible errors consist of too few parameters, too many parameters, or mismatched parameter types.

Figure (3.2)

Example:

```
foo (a, b, c)
int     a;
char    *b;
long    c;
{}

bar ()
```

```
{
    long          x;
    char         *y;
    unsigned int z;

    foo (x, y)

/* 'x' would be flagged as being of the    */
/* wrong type. 'y' would be OK and a param */
/* missing message would be output.        */
}
```

3.2.3.4 Function Invocation vs. Function Definition

These errors occur when a function invocation does not match up properly with a function definition.

Figure (3.3)

Example:

```
main()
{
    int a;

    a= transform ();

   /* ERROR: without "char *transform();"*/
   /* a return value of "int" is assumed */
}

/*
**  If the following function definition was
**  in a separate module, the parameter checker
**  would notice the difference between the
**  definition and the invocation above.
*/
    char *transform ()
    {
        char    *xyz;

        return (xyz);
    }
```

3.2.3.5 Function Declaration vs. Function Definition

These errors occur when a function declaration does not match up properly with a function

definition.

Figure (3.4)

Example:

```
main()
{
    char    *foo();

    foo();
}
/*
**  If the following function definition was
**  in a separate module, the parameter
**  checker would notice the difference
**  between the definition and the
**  invocation above.
*/
int foo ()
{
        .
        .
    return (1);
}
```

3.2.4 Additional Features of the Parameter Checker

3.2.4.1 Detection of Unused Functions

Functions that have been defined in the source code but not invoked will be flagged. This code could be investigated for possible removal.

3.2.4.2 Detection of a Missing Return Value Within a Function

These errors are diagnosed during the processing of an individual function within a module. Functions that are not defined as being of type void() should always have a return value present in the return statement.

Figure (3.5)

Example:

```
int foo()
{
    .
    return;      /* ERROR: no value */
}
```

3.2.4.3 Cross-Reference Map

The collection of all function definition, declaration and invocation information allows the parameter checker to produce a comprehensive cross-reference map.

The map will include the module name, function name and line number for all invocations and declarations.

3.3 The Cost of Errors

3.3.1 Person-days as a Unit of Cost

The benefit to be gained from any reduction in errors can be measured in several ways (increased product quality, better performance, reduced development costs, etc.). However, many of the measurements are of a subjective nature. In order to obtain an objective measure of the benefits offered by parameter checking, the potential savings will be quantified in terms of person-days.

These person-days are those that would not have been needed by development, support, or test personnel, if the errors found by the parameter checker had been detected

and corrected prior to release or testing.

3.3.2 Different Levels of Error Detection

The errors found in Observation Reports differ from the errors found in Trouble Reports in many ways. Errors documented in Observation Reports are not present in any of the released software because they were found and fixed during the development/test phase of the product release. Errors documented in Trouble Reports were found in the field after the release of the product. Field detected problems require the time of field support, in-house support, development and Software Quality Assurance personnel.

Clearly, the potential savings in avoiding Trouble Reports is higher than avoiding Observation Reports. The work needed to fix a Trouble Report is documented in the Software Change Forms.

3.3.3 Cost Components of Error Correction

The following provides an explanation of the functions and role of each group involved in the release of a product (Field, Support, Development, Quality Assurance).

3.3.3.1 Field Personnel

The Field Support personnel that deal with software troubles typically have been trained in the product from a feature/function point of

view. Generally, they do not have an in-depth knowledge of the internal design of a software product. Field personnel have limited access to the source code.

Problem detection and correction start with a call and a visit to the site by the field engineer. A trip to the customer site may often require a trip of several hours by automobile or plane. The detection efforts of the field engineer will usually be limited to isolating the problem to a specific set of actions or hardware and attempting to establish some repeatability.

Data collected from these actions is then sent to the home office with any other pertinent information. The field engineer is also responsible for informing the customer of the nature of the problem and any short-term actions necessary to work around it. The average time estimated for these activities is one and a half (1.5) days.

Later, when the problem has been corrected by the home office, the field engineer will be responsible for returning to the customer site and installing and/or implementing the correction to the problem. The average time estimated for these activities is one (1) day.

3.3.3.2 Customer Software Personnel

The Customer Software personnel act as the initial in-house contact for all software problems reported from the field. They accept the data from the field engineer and perform an initial problem analysis. Based upon the severity of the problem, they may have to involve the Software Development personnel that were responsible for the initial product design and implementation.

They are responsible for tracking the problem resolution, coordination of personnel from the field and various departments that get involved, modification of the software to fix the problem, and verification that the correction was implemented and that the problem no longer exists. The average time estimated for these activities is three (3) days.

3.3.3.3 Development Personnel

The Development personnel get involved with the resolution of field problems in either a consultative manner, or they may take an active part in fixing the problem if changes to the design of the application are required. The average time estimated for these activities is two (2) days.

3.3.3.4 Software Quality Assurance Personnel

Software Quality Assurance personnel will apply the new software to the product test plan and note any discrepancies. Any observations noted in this stage may require further activities by the software developers and a product re-submittal for testing. The average time estimated for these activities is five (5) days.

3.3.4 Cost Measurement Formula

The formulas in Table 3.1 will be used to quantify the cost for fixing a parameter error detected in an Observation Report and a Trouble Report.

(Table 3.1)

	Observation Report	Trouble Report
a) Field:	0.0 pd	2.5 pd
b) Support:	0.0 pd	3.0 pd
c) Development:	2.0 pd	2.0 pd
d) Quality Assurance:	5.0 pd	5.0 pd
Total:	7.0 pd	12.5 pd

(pd = person day)

While not all errors take the same amount of effort to correct, an estimate was made for the time needed by members of all involved departments. The numbers and the ratios in Table 3.1 were chosen as representative for the type of errors encountered in the first one and a half years of support for the application.

3.3.5 Cost as a Function of Releasing a Product

Usually the fixes for parameter based Observation Reports were re-submitted for testing along with fixes for non-parameter based Observation Reports. Therefore, the Quality Assurance time may not always be directly attributed to the Observation Report error. The correction of a parameter based Trouble Report may be the sole reason for starting a test cycle and incurring the cost of Quality Assurance.

There is also a cost associated in running the parameter checker. For a release level, a setup time of two (2) person-days is needed for the parameter checker. This is a one time cost that covers the creation of input files for the parameter checker and the maintenance of these files over the development/test cycles of any given release.

4.0 Project Implementation

4.1 Data Collection

The Historical data base reports and source code will be collected for four releases (which are defined as A, B, C and D). The parameter checker routine will be run on all source code for the various release levels.

4.2 Data Classification

These reports will be broken down by release level and classified into two error groups; those of parameter origin and others.

The output of the parameter checker will be tabulated and classified by the nature of the errors (see section 3.2.3 Types of Errors Detected).

4.3 Presentation of Data

4.3.1 Application Software

Examination of the application software releases is based on the following:

# of Executables - The number of linked jobs that make up the application release level.

Lines of Code - Lines of 'C' source code for each executable. This total does NOT include comments, empty lines or lines from #include files.

Table 4.1: Line Counts by Release Level

Release Level	# of Executables	# Lines of Code
A	7	30245
B	7	31073
C	10	53440
D	10	54136

4.3.2 Trouble Reports

Examination of the Trouble Reports is based on the following:

# of Trouble Reports - Trouble reports filed per release level.

Non-S/W TRs - Trouble Reports that were caused by installation, hardware or documentation.

S/W non-param TRs - Software caused trouble reports that were not parameter related.

of S/W Param TRs - Software caused trouble reports that ARE parameter related.

Table 4.2: Trouble Report (TR) Totals by Release Level

Release Level	# of Trouble Reports	# Non -S/W TRs	# of S/W non-param TRs	# of S/W Parameter TRs
A	8	0	8	0
B	12	8	3	1
C	6	4	2	0
D	2	2	0	0
Total:	28	14	13	1

4.3.3 Observation Reports

Examination of the Observation Reports is based on the following:

- # of ORs - Observation reports filed per release level.
- # Non-S/W ORs - Observation Reports that were caused by installation, hardware or documentation.
- # S/W non-param ORs - Software caused observation reports that were not parameter related.
- # of S/W Param ORs - Software caused observation reports that ARE parameter related.

Table 4.3: Observation Report (OR) Totals by Release Level

Release Level	# of ORs	# Non -S/W ORs	# of S/W non-param ORs	# of S/W Parameter ORs
A	30	9	21	0
B	10	3	7	0
C	107	23	84	0
D	9	1	8	0
Total:	156	36	120	0

4.3.4 Parameter Checker Output

Examination of the Parameter Checker Output is based on the following:

Executable # - This number is an identifier for a specific executable and will retain its value across release levels.

Total PC errors - Func. Definition Return Errors
+ Return Type Errors
+ Declaration Errors
+ Invocation Errors
+ Parameter Errors

Func. Definition/
Return Errors - These errors arise when the variable type used by the 'return' statements within a function do not match the return definition of the function (see 3.2.4.2).

Return Type Errors - These errors occur when the results of a function invocation are do not match properly with a function definition (see 3.2.4.4).

Declaration Errors - These errors occur when a function has not been declared the same as it was defined (see 3.2.4.5).

Invocation Errors - The number of function

invocations that contained erroneous parameters.

Parameter Errors - Errors consist of too many parameters, too few parameters or mismatched parameter types (see section 3.2.4.3 and Tables 5, 6, 7 and 8).

Table 4.4 defines the Parameter Checker Error totals for each of the four release levels

Table 4.4: Parameter Checker Error Totals by Release Level

Release Level 'A'

Executable Number	PC detected Errors	Func. Def. /Return Errors	Return Type Errors	Declaration Errors	Invocation Errors	Param. Errors
1	3748	29	557	46	976	2140
2	1528	19	379	35	315	780
3	332	13	60	3	80	176
4	1077	15	235	7	248	572
5	893	15	197	6	210	465
6	899	15	199	7	211	467
7	25	0	6	0	6	13
Total:	8502	106	1633	104	2046	4613

Release Level 'B'

Executable Number	PC detected Errors	Func. Def. /Return Errors	Return Type Errors	Declaration Errors	Invocation Errors	Param. Errors
1	3793	40	589	46	970	2148
2	1536	19	383	35	315	784
3	332	13	60	3	80	176
4	1129	15	249	7	258	600
5	901	15	201	6	210	469
6	903	15	203	7	211	467
7	25	0	6	0	6	13
Total:	8619	117	1691	104	2050	4657

Table 4.4 (continued)

Release Level 'C'

Executable Number	PC	Total detected Errors	Func. Def. /Return Errors	Return Type Errors	Declaration Errors	Invocation Errors	Param. Errors
1		4062	66	743	61	938	2254
2		1165	23	249	11	265	617
3		1133	18	301	7	210	597
4		2631	88	488	31	612	1412
5		861	18	191	7	199	446
6		863	18	193	8	200	444
7		25	0	6	0	6	13
8		406	10	118	8	64	206
9		1008	12	179	39	258	520
10		1034	16	135	15	255	613
Total:		13188	269	2603	187	3007	7122

Release Level 'D'

Executable Number	PC	Total detected Errors	Func. Def. /Return Errors	Return Type Errors	Declaration Errors	Invocation Errors	Param. Errors
1		4232	77	765	64	1004	2322
2		1103	24	242	11	249	577
3		1282	18	334	7	242	681
4		2613	85	293	76	705	1454
5		869	18	194	7	201	449
6		878	18	197	8	204	451
7		25	0	6	0	6	13
8		406	10	118	8	64	206
9		1296	12	246	39	318	681
10		1034	16	135	15	255	613
Total:		13738	278	2530	235	3248	7447

Table 4.5 defines the number and type of Parameter Errors found in each of the four release levels

Table 4.5: Parameter Error Types by Release Level

Release Level 'A'

Executable Number	Total Parameter Errors	Extra Params.	Missing Params.	Type Mismatch	Pointer Mismatch
1	2140	9	217	1134	780
2	780	5	2	615	158
3	176	13	0	93	70
4	572	5	0	426	141
5	465	5	0	362	98
6	463	0	0	367	96
7	13	0	0	7	6
Total:	4609	37	219	3004	1349

Release Level 'B'

Executable Number	Total Parameter Errors	Extra Params.	Missing Params.	Type Mismatch	Pointer Mismatch
1	2158	9	217	1169	763
2	784	5	2	619	158
3	176	13	0	93	70
4	600	5	0	444	151
5	469	5	0	366	98
6	467	0	0	371	96
7	13	0	0	7	6
Total:	4667	37	219	3069	1342

Table 4.5 (continued)

Release Level 'C'

Executable Number	Total Parameter Errors	Extra Params.	Missing Params.	Type Mismatch	Pointer Mismatch
1	2254	2	161	1284	807
2	617	5	0	418	194
3	597	13	0	398	186
4	1412	0	1	707	704
5	446	5	0	321	120
6	444	0	0	326	118
7	13	0	0	7	6
8	206	0	0	170	36
9	520	0	0	365	155
10	613	1	0	373	239
Total:	7122	26	162	4369	2565

Release Level 'D'

Executable Number	Total Parameter Errors	Extra Params.	Missing Params.	Type Mismatch	Pointer Mismatch
1	2322	1	171	1319	831
2	577	5	0	407	165
3	681	13	0	449	219
4	1454	0	1	738	715
5	449	5	0	324	120
6	451	0	0	331	120
7	13	0	0	7	6
8	206	0	0	170	36
9	681	0	0	465	216
10	613	1	0	373	239
Total:	7447	25	172	4583	2667

5.0 Analysis of Output

5.1 Analysis of Trouble Reports

The number of Trouble Reports for the product increased slightly as the application went through an initial phase of installations. Over the course of the next releases, the number of Trouble Reports declined as the application entered a more mature, stable phase. As shown in section 4.2.2.2, only one Trouble Report included an error that could be tracked backed to a parameter mistake.

The parameters to the function "perr" (print error, see figure 5.1) are passed to a "sprintf" call. A mismatch of the parameters to the "sprintf" format caused garbage to be output to a file instead of meaningful diagnostic information; thereby making the reasons for the original problem more difficult to find.

(Figure 5.1)

```
Definition of erroneously invoked function :
*********************************************************
Function Name        : perr
Return Type          : Void
Number of Parameters : 3
Parameter #1 : fmt                    (*Char)
Parameter #2 : arg1                   (*Char)
Parameter #3 : arg2                   (*Char)

*ERROR* : In file DCISCR.CL0 at LINE #58 from indx_create()
 Erroneous invocation :
 *Param #1  "%x, name=%s"        *Char
 *Param #2  indx_fno             U_Integer  TYPE MISMATCH
 *Param #3  indx_fil[indx_fno]   *Char
```

The first parameter is used as the format string for a "sprintf" function call. The function "perr" is expecting two additional character pointers (4 bytes each) on the stack as objects for

the format string. By putting an unsigned integer on the stack as the second parameter, the function will now pull two incorrect pointers off of the stack. The second pointer of the function will consist of the unsigned integer and the first two bytes of the third parameter. The third pointer will use the second two bytes of the third parameter and the next two bytes on the stack (which happen to be part of the return address).

When the third pointer is used to satisfy the "%s" control in the "sprintf" function, the data to which it points will be transferred until a null character is found. The absence of a null character could cause large amounts of extraneous data to be copied into the destination array. The destination array is a local variable (i.e on the stack). Over-writing the local variable will compromise the stack and potentially crash the program.

Since the third parameter (which now contains part of the return address) is used as a read-only value, no harm is done to it. The stack is adjusted by the calling function so the missing two bytes do not harm any stack pointers.

5.1.1 Effect of Trouble Report Errors on the Application

The misaligned parameter error (see figure 5.1) occurred during the handling of an I/O error. While not the main problem of the trouble report, this parameter error masked the original problem and potentially compounded the error.

5.1.2 Problem Resolution

Since the parameters that are passed to the "sprintf" function will go onto the stack as four bytes, the "sprintf" format string should be expecting parameters that will be four-bytes values. Therefore, the second and third parameters to the "perr" function are required to be four byte values (character pointers: %s, or long ints: %D or %U).

All calls to the "perr" function were inspected. Those that did not conform to the above requirements were recoded. Many additional calls to "perr" were found to have errors of this type. All these errors were fixed and packaged as part of the release level 'C' effort.

5.1.3 Cost of Problem Resolution

In applying the cost formula to this problem, a value of 12.5 person-days was obtained to respond to this trouble report (see Table 5.1).

Table 5.1:

	Trouble Report
a) Field:	2.5 pd
b) Support:	3.0 pd
c) Development:	2.0 pd
d) Quality Assurance:	5.0 pd
Total:	12.5 pd

(pd = person day)

5.2 Analysis of Observation Reports

In the four release levels studied, a total of 120 Observation

Reports could be attributed to software deficiencies. However, none of these could be directly attributable to parameter errors. Several of the Observation Reports were suspected of being caused by parameter errors, but the nature of Observation Report data did not lend itself to a thorough examination of the problem's underlying cause.

An Observation Report is more concerned with documenting an error rather than its ultimate solution. The Software Change Forms are meant to serve as the documentation for error fixes, but if the problem was introduced into the testing cycle because of code for a new feature, then the fix was not necessarily documented. Thus many Observations that are recorded are not answered with specific documentation.

5.3 Analysis of Parameter Checker Output

While all error totals reported by the Parameter Checker have gone up with each release level, the parameter errors themselves show an opposite trend. The totals for Missing Parameters and Extra Parameters have been decreasing across release levels (see Tables 4.4 and 4.5). These parameters are the ones most often associated with causing problems in software, due to a higher potential for causing damage to the stack. In this section all occurrences of Extra and Missing Parameters that exist in the latest release level ('D') will be presented along with any other parameter error types that are considered potentially dangerous.

For all Missing Parameters, Extra Parameters and other potentially dangerous errors, a description of the effects of the

error will be presented as well as a solution to the conflict (see Appendix B for a complete listing of the raw Parameter Checker data).

5.3.1 Presentation of Parameter Errors for Release Level 'D'

The evaluation of the errors will be presented in order of the executable numbers for Release Level 'D'.

Executable #1:

An extra parameter was found in an invocation of the function "add_log". The function definition of "add_log" does contain a second parameter (the variable "stat"). Since the first parameter (the variable "db_p") is in the proper location in the parameter list it is used correctly and the second parameter is ignored. When the function is complete, the stack is correctly cleaned up because the calling function performs the stack frame adjustments.

(Figure 5.2)

```
Definition of erroneously invoked function :
*******************************************************
Function Name         : add_log
Declaration Line      : 161
Return Type           : Void
Number of Parameters : 1
Parameter #1 : dbp          (*Struct DB_STR, Bytes = 16)

*ERROR* : In DCADLG.CL4 at LINE #83 from trlog_task
 Erroneous invocation :
 *Param #1  db_p         *Struct    DB_STR, Bytes=16
 *Param #2  &stat        *U_Integer  EXTRA
```

The 171 missing parameter errors for this executable are all caused by calls to the "perr" function (see Table

4.5). This function, as described earlier (see section 5.1), has a high potential for causing a stack compromise. Not all the cases caused the aforementioned stack misalignment.

Executable #2:

The extra parameters present in this executable are caused by using a stub function to satisfy the requirements of the system linker. The module DCSFMT.CL3 contains multiple functions and is present in several executables. The function (FMT_PIN) that contains the call to the stubbed function (PIN) will never be invoked for executable #2. In other words, the function FMT_PIN is unused code in this executable.

(Figure 5.3)

```
Definition of erroneously invoked function :
************************************************************
Function Name         : PIN
Declaration Line      : 1
Return Type           : Integer
Number of Parameters : 0

*ERROR* : In DCSFMT.CL3 at LINE #664 from FMT_PIN
 Erroneous invocation :
 *Param #1  PCALC1          U_Integer  EXTRA
 *Param #2  PCALC2          U_Integer  EXTRA
 *Param #3  PCALC3          U_Integer  EXTRA
 *Param #4  CARD_NUM        U_Long     EXTRA
 *Param #5  PIN_NUM         *Char      EXTRA
```

An analysis of all other detected errors for executable #2 found that none posed any danger to the application.

Executable #3:

The Extra Parameter errors were all caused by stub

functions as described in executable #2.

An analysis of all other detected errors for executable #3 found that none posed any danger to the application.

Executable #4:

The executable contained a call to the "setvideo" function that had a missing parameter. The missing parameter will cause the function to use the return address on the stack as a value to represent the video parameter.

(Figure 5.4)

```
Definition of erroneously invoked function :
************************************************************
Function Name         : setlabel
Declaration Line      : 60
Return Type           : Integer
Number of Parameters : 5
Parameter #1 : scrnptr     (*Struct SCREEN, Bytes = 1944)
Parameter #2 : row         (Integer)
Parameter #3 : col         (Integer)
Parameter #4 : bufptr      (*Char)
Parameter #5 : video       (U_Char)

*ERROR* : In DCSIOT.CL4 at LINE #753 from iotermrec_scrn
 Erroneous invocation :
 *Param #1  screenptr        *Struct SCREEN, Bytes=1944
 *Param #2  5                Integer
 *Param #3  17               Integer
 *Param #4  status_label     *Char
 *Param #5                            MISSING
```

An analysis of all other detected errors for executable #4 found that none posed any danger to the application.

Executable #5:

The Extra Parameter errors were all caused by stub functions as described in executable #2.

An analysis of all other detected errors for executable #5 found that none posed any danger to the application.

Executable #6:

There are no missing or extra parameter errors in this executable.

An analysis of all other detected errors for executable #6 found that none posed any danger to the application.

Executable #7:

There are no missing or extra parameter errors in this executable.

An analysis of all other detected errors for executable #7 found that none posed any danger to the application.

Executable #8:

There are no missing or extra parameter errors in this executable.

An analysis of all other detected errors for executable #8 found that none posed any danger to the application.

Executable #9:

There are no missing or extra parameter errors in this executable. However, an error concerning mismatched parameters was detected (see Figure 5.5).

The parameter to the function rmlb() is the output of

the function rmtb(). The function rmlb() requires a character pointer as a parameter and normally the function rmtb() returns a character pointer. In this function the rmtb() was not specifically declared so the compiler defaulted into returning an integer.

The function rmlb() takes a character pointer and removes all leading blanks from the string by sliding the characters down over the blanks. Since this function will be dealing with an invalid pointer, it is possible that it could delete code or data.

(Figure 5.5)

```
Definition of erroneously invoked function :
************************************************************
Function Name         : rmlb
Declaration Line      : 694
Return Type           : *Char
Number of Parameters : 1
Parameter #1 : str                    (*Char)

*ERROR* : In DCRDPT.CL4 at LINE #772 from dpt_scrn
 Erroneous invocation :
 ****Param #1  rmtb       Integer    TYPE MISMATCH

 Example Code:
        rmlb (rmtb (dept_keyp));
```

The function "rmtb" was NOT declared as returning a "char *" thus generating an illegal return value that is used for the function "rmtb" as a parameter.

An analysis of all other detected errors for executable #9 found that none posed any danger to the application.

Executable #10:

The executable contains an extra parameter in a call to

the "LOG_MSTAT" function. The extra parameter is the "POINT_NO" parameter which misaligned the stack offsets to the "LOG_MBOX" parameter. "LOG_MBOX" is a mailbox that is used for inter-task communication. When a message is to be sent, the mailbox is detected as being invalid and the send operation is aborted.

(Figure 5.6)

```
Definition of erroneously invoked function :
************************************************************
Function Name         : LOG_MSTAT
Declaration Line      : 84
Return Type           : Integer
Number of Parameters : 4
Parameter #1 : LE_NUM                (U_Integer)
Parameter #2 : MLOG_COD              (U_Integer)
Parameter #3 : MLOG_ID               (U_Integer)
Parameter #4 : LOG_MBOX              (U_Integer)

*ERROR* : In DCMPRC.CL3 at LINE #617 from MPROC_TASK
 Erroneous invocation :
 *Param #1  MLE_NUM         U_Integer
 *Param #2  0x0211          Integer    TYPE MISMATCH
 *Param #3  0               Integer    TYPE MISMATCH
 *Param #4  POINT_NO        Integer    TYPE MISMATCH
 *Param #5  LOG_MBOX        U_Integer  EXTRA
```

5.3.2 Effects of Parameter Errors on the Application

Executable #1:

The extra parameter to the "add_log" function caused no problems for the application. However, if the called function ("add_log") did the stack cleanup when exiting, the stack would be compromised (see section 5.2 on ANSI 'C' changes).

The missing parameters to the "perr" function caused no problems for the application during normal operations. However, if the program were to take a error processing

path and encounter one of the calls to "perr" that caused a stack misalignment, the system could suffer a fatal error.

Executable #2:

The use of stub functions is merely a short-cut to get past the system linker. By stubbing the PIN function, instead of breaking the FMT_PIN function into a separate file, unused code was added to our executable.

Executable #3:

The extra parameters were caused by the use of stub functions. The use of stub functions is merely a short-cut to get past the system linker. By stubbing the functions, instead of breaking the functions into separate files, unused code was added to our executable.

Executable #4:

The Missing Parameter to the "setlabel" will be resolved by using part of the return address for the parameter value. The "Setlabel" function tests for valid parameter values (including "video" parameter) before it does any work. If an error is detected, "setlabel" will return an ERROR condition. It is possible that the stack value used for the missing object would fall into the range of valid values. In that case, an invocation of this function will result in the CRT having a field with an unwanted video attribute (inverse, blinking, etc.).

Executable #5:

The extra parameters were caused by the use of stub functions. The use of stub functions is merely a short-cut to get past the system linker. By stubbing the functions, instead of breaking the functions into separate files, unused code was added to our executable.

Executable #6:

There are no missing parameters, extra parameter errors or other detected errors in this executable that pose any threat to the application.

Executable #7:

There are no missing parameters, extra parameter errors or other detected errors in this executable that pose any threat to the application.

Executable #8:

There are no missing parameters, extra parameter errors or other detected errors in this executable that pose any threat to the application.

Executable #9:

The function "rmlb" removes leading blanks from a string. If the garbage pointer returned from "rmtb" points at an area of memory filled with spaces (i.e. 0x20), then this memory will be overwritten as it moves the following memory locations over it. This

overwriting of memory could cause a crash.

Executable #10:

The call to the "log_mstat" code is executed when a user has made a data file change. The "send message to a mailbox" operation that is aborted prevents the file change from being noted to the audit file on the disk.

While the omission of this event from the disk does not cause any problems to the operation of the program, it represents a serious flaw for the users of the application. This package is running an application that requires an audit trail. Failure to note a change in the system configuration is a serious security breach.

5.3.3 Problem Resolution

Executable #1:

The extra parameter in the call to the "add_log" function will be removed.

The missing parameters for the calls to the "perr" function will be replaced with NULL pointers where appropriate and the format string objects will be corrected (made to four byte objects) to prevent a pointer misalignment on the call to "sprintf" (see section 5.1).

Executable #2:

The FMT_PIN function will be put in a separate file and

only linked in with the executable that requires it. This will allow us to remove the use of the PIN stub function and avoid unused code in the executable.

Executable #3:

The requirement for stub functions will be removed by breaking out required functions into separate files.

Executable #4:

The missing parameter will be filled in with an appropriate value.

Executable #5:

The requirement for stub functions will be removed by breaking out required functions into separate files.

Executable #6:

No additional work is required for this executable.

Executable #7:

No additional work is required for this executable.

Executable #8:

No additional work is required for this executable.

Executable #9:

The declaration "char *rmtb();" will be added to the function "dpt_scrn()" in the file DCRDPT.CL5. This will eliminate the TYPE MISMATCH error.

Executable #10:

The POINT_NO parameter will be removed from the "LOG_STAT()" function invocation at line #617 in function MPROC_TASK in file DCMPRC.CL3.

5.3.4 Cost of Problem Resolution

Since the errors discussed in section 5.3 have not been found in either the field or test laboratory, no indirect costs (i.e. non-development time) has been lost to them.

The cost to fix the discovered errors is two (2) person-days (see Table 5.2).

Table 5.2:

Correct Errors:	0.5 pd
Generate Application:	0.5 pd
Test Application (Development Personnel):	1.0 pd
Total:	2.0 pd

No Quality Assurance time is factored in since the above fixes will not cause a release effort, but will be one of many changes in a subsequent release.

5.4 Analysis of Other Parameter Checker Output

5.4.1 Unused Functions

One of the additional features of the parameter checker program is the detection of unused functions. Often many functions are grouped together in a file for

convenience. However, when this module is added to a program because one of the functions is needed, the program also links in all the unused functions in the file. Breaking up the functions into separate modules would reduce the executable program size by removing these unused functions.

Table 5.3 shows the number of unused functions present in the executables from release 'D' of the application.

Table 5.3 - Unused functions in Release Level 'D'

Executable Number	Total Lines of 'C' Code	Total Unused Functions	Lines of Unused 'C' Code	% of Total Unused
1	13909	15	327	2.4
2	3631	12	388	10.1
3	3575	10	272	7.6
4	15442	2	50	0.3
5	2956	11	436	14.7
6	2955	10	419	14.2
7	254	0	0	0.0
8	1363	6	133	9.8
9	7147	4	95	1.3
10	2904	6	125	4.3
Total:	54136	76	2245	4.15

The total number of unused lines of code represents 4.15% of the 54,136 lines of code for this release. The bulk of the unused code could be eliminated if the functions were broken up into separate modules.

5.4.2 Potential Danger of "TYPE MISMATCH and POINTER MISMATCH" Errors

The bulk of errors detected by the parameter checker (97.4% in Release 'D'; see Table 4.5) consisted of mismatched parameter types and mismatched pointer types.

The parameter checker flags differences such as "int" vs. "unsigned" as TYPE MISMATCH errors. This particular difference manifests itself during assignment operations where sign extension takes place or in logical expressions. Although these are not parameter errors, they can serve as useful flags for potential assignment errors. Each event of this type was reviewed to determine if it posed any threat to the application.

Another form of TYPE MISMATCH common with this application code was the "char" vs. "int" type. This may seem like a dangerous conflict since these types are of different length, but the compiler for this system always loads parameter values as two-byte words. This type of error has a large danger potential if the application should be ported to a system that does not perform this sort of parameter padding.

The POINTER MISMATCH errors occurred when a pointer parameter was not declared of the same type by the calling function and the function definition.

Since all pointers are four bytes on this system, there is never a stack problem with these errors. The only danger potential with these events is if a function attempts to use the pointers for placing values into variables indirectly. Attempts to place a value indirectly through a character pointer results in one byte being placed.

Putting the the same value through a integer pointer would result in two bytes being placed.

(Figure 5.7)

```
main
{
    int junk = 256;      /* lower byte = 0000 0000   */
                         /* upper byte = 0000 0001   */
    foo (&junk);
}

foo (p1)
char    *p1;
{
    int number;

    number = *p1;
                /* the indirect operation on p1 (a  */
                /* character pointer) would retrieve*/
                /* the lower byte (a zero) and would*/
                /* extend it out to fill the local  */
                /* integer variable.  Thus, "number"*/
                /* would be zero (0).               */
}
```

Therefore, while a POINTER MISMATCH is not a parameter error, it points out a potential area of assignment error. Each POINTER MISMATCH was checked for potential danger to the application.

6.0 Summary

6.1 Review of "C" Language Project Goals

The results of this study (see tables 4.1 and 4.5) show a very high number of detected parameter errors. While the presence of such a large number of errors is not enough to label this as a "poor quality" application, it can and should be a cause for concern. The opportunity for potential errors being in the program is directly proportional to the number of detected mismatches.

In Japanese quality studies, an error rate of greater than 10 defects per 1000 lines of code is taken as unacceptable (a defect is taken as a error that requires correction; Miller - 1987). If you were to require that the application be free of parameter errors, then the error rate of 1 per 3.9 lines of code (256 per 1000 lines of code) is well above the acceptable level. The high degree of errors is an indictment of the weakness of the "C" language type-checking.

As the study shows a continuing increase in the number of errors as the application grows, extra precautions must be used when implementing and maintaining systems using the "C" language. Many techniques can be used to detect errors during all phases of product development. The Parameter Checker is just one tool available during the implementation phase of a project to supplement the error diagnostics of a compiler and linker. Other devices such as code reviews or structured walk-throughs can assist in finding problems in an application.

6.2 Review of Application Quality Project Goals

Although the total number of errors detected in this application was much higher than expected, the impact on the application in perceived quality was low.

An analysis of the data has shown that a fault found in the application has been attributed to a parameter error that could have been caught using the Parameter Checker.

The error (discussed in section 5.1) caused an estimated 7.5 person-days to be spent for the correction. This cost could have been avoided by using the Parameter Checker.

The Parameter Checker found three previously undetected errors in release level 'D' of the application and highlighted how the use of stub functions is causing the inclusion of unused functions. The newly detected errors will be fixed in a future release and the use of stub functions will be addressed.

While the number of errors detected by the Parameter Checker was very high (1 error / 3.9 lines of code; See Tables 4.1 and 4.5) the impact of these detected errors on the current application environment was low. Only one of these errors has manifested itself in a released product.

6.3 What is the role of Parameter Checking in future development?

6.3.1 Changes in Software Development Cycle

One of the advantages of the Parameter Checker is that the data collection process is automated. It is hard to argue against a tool that helps deliver software with

fewer bugs, especially when the tool is available and requires little time or effort to maintain and execute.

It would seem that given the low cost of using the Parameter Checker, that it would be beneficial to include its use as a required step in the release of a software product. The files necessary to support parameter checking should be maintained and archived as a standard part of developing or upgrading any application software. In addition, the use of parameter checking should be encouraged during the development or upgrade of a product.

In order to allow for further quality control studies of errors and their origins in this application, additional documentation will have to available concerning the corrections made for Observation Reports.

6.3.2 The ANSI 'C' Standard

Currently the 'C' language and its libraries are undergoing standardization by the ANSI X3J11 committee. One of the new features being proposed in the standards is called "prototyping". With this feature the programmer will be able to declare the type (and optionally the name) of all parameters in function definitions and function declarations (see Figure 6.1). By making this information available to the compiler, parameter checking can occur directly in the compilation phase.

If the user creates an #include file of these prototypes, parameter checking between functions in different modules will be possible by merely adding the #include file to each source code file.

(Figure 6.1)

Prototype example:

```
char *strncpy (char *s1, const char *s2, int n)
```

- Function returns a character pointer.
- Parameter #1 is a character pointer.
- Parameter #2 is a character pointer and the data it points at is not modifiable.
- Parameter #3 is an integer.

Another feature in the proposed standard is the use of the keyword "void". Under ANSI 'C', functions that are defined to return pointers of type "void *" will be exempt from producing pointer mismatch errors during assignments. This means that the assignment of the pointer returned by malloc() in Figure 6.2 will not cause a pointer mismatch error.

(Figure 6.2)

```
main ()
{
    void    *malloc (int);
    char    *bar;
    int     *foo;

    foo = malloc (10);
    bar = malloc (10);

        /* the output from malloc() can */
        /* be assigned to any pointer   */
        /* type without error           */
}
```

Finally, the last major feature of the 'C' standard that applies to parameter checking, is the ability to declare functions with a variable number of arguments (see Figure 6.3). This will allow prototypes to be built for the printf and scanf family of standard functions as well as user defined functions that have a variable number of arguments.

(Figure 6.3)

```
char *printf (char *format, ...);
```

- Function returns a character pointer.
- Parameter #1 is a character pointer.
- After Parameter #1 there is a variable number of parameters possible (0 to n, indicated by the ellipsis "..."). Any arguments that appear are not checked against the use of the function and the default argument promotions are applied to them.

6.3.3 Hardware Advancements

One of the most useful mechanisms being provided by the newest generation of microprocessors is memory management at the hardware level. As memory is allocated by programs, descriptors are created for each segment. The descriptors are used to trap access attempts outside of the memory allocated to the specific variable. Bad pointer parameters that are put on the stack run a very high chance of generating one of these memory protection errors. Since the called function is very likely to use the pointer to access data, a memory error trap will occur, thus flagging the bad parameter. Granted, this is not preventing a parameter error from

happening, but lacking a parameter checker or a prototyping facility, this mechanism has proven to be one of the best way to trap pointer parameter errors.

6.4 Conclusions

The total number of errors detected in this application was much higher than expected and seems to be increasing in direct proportion to the number of lines of source code. Regardless of the effects to this point, the high number of parameter errors present in this source code could mean potential problems for the application over its life cycle. As maintenance changes are made to the source code, the possibility exists that one of the parameter errors previously safely ignored may become dangerous to the application. For example, if a a programmer modifies a function that has a variable declared as a character pointer, the desired changes may not carry over to a called routine that incorrectly declares the pointer as being of type integer.

Clearly, porting the application code to a different system where basic characteristics of the environment are different is impossible without addressing the errors detected in this study. A change in type sizes (i.e. int, short, etc.), variable promotion rules, parameter padding, address boundary restrictions or other machine or compiler dependent rules could cause currently harmless errors to have far more serious consequences.

Since parameter errors were made, it can be surmised that most of the deadly parameter-type errors for this application were

caught through testing by development personnel or Software Quality Assurance personnel. Potentially deadly errors that are part of error handling routines seem more likely to slip through the testing process. Primarily because not all error paths are as likely to be tested as is the program's mainstream functionality.

The current trend of increased "Quality Consciousness" is spreading to the Software Engineering field. The long held belief that you could "inspect quality into a product" is being abandoned. Traditional quality control (i.e. "after the fact" testing and inspection) will not work over the long term as products become more complex and sophisticated.

Quality control methods adopted by the Japanese have proven very successful in turning out reliable products. The basis of some of these methods is the reduction of variability, the statistical evaluation of components, and attacking poor quality as a system wide problem. All of these current methods involve the quantifying of the types and frequency of errors. The detection and discovery of the cause of errors becomes the first step in fixing the processes that allow them.

The Parameter Checker, aside from finding errors, is a powerful automated tool for collecting one set of this quantitative data and as such is a welcome addition to the software engineer's tool set. Using the Parameter Checker can directly impact and significantly increase the overall quality and performance of the applications programs.

Bibliography

Aho, A.V., Sethi, R., Ullman, J.D.:
Compilers, Principles, Techniques and Tools,
Addison-Wesley, 1986

Al Jarrah, M. M., Torsun, I. S.:
"An Empirical Analysis of COBOL Programs",
Software - Practice and Experience (GB),
Vol. 9, pp. 341-359 1979

Brookes, G. R., Wilson, I. R., Addyman, A. M.:
"A Static Analysis of Pascal Program Structures",
Software - Practice and Experience (GB),
Vol. 12, No. 10 pp. 959-63 October, 1982

Conti, R. A.:
"Software Productivity Features Provided by the Ada
Language and the VAX Ada Compiler",
Digitial Technical Journal
No. 6 pp. 51-61, February, 1988

Dunn, R.:
"Static Analysis for Software Defect Removal",
Software Defect Removal,
Chapter 5, pp. 126-149,
McGraw-Hill, 1984

Eventoff, W., Anderson, G., Price, R., Rabinowitz, I.:
"Ada: A significant software-engineering tool",
Mini-Micro Systems
pp. 209-231, April, 1981

Fosdick, L. D., Osterweil, L. J.:
"Data Flow Analysis in Software Reliability",
ACM Computing Surveys
Vol. 8 No. 3 pp. 305-330 September, 1976

Gantenbein, R. E.:
"Support for Dynamic Binding in Strongly Typed Languages",
SIGPLAN Notices
Vol. 22, no. 6 pp. 69-74 June, 1987

Gerard, S. N.:
"Adding an Authorization Dimension to Strong Type Checking",
SIGPLAN Notices
Vol. 23, no. 6 pp. 145-151 June, 1988

Hendrix, J.E.:
Small C Handbook,
Reston Publishing Company, 1984

Intel Corp.:
"lint: C Program Checker",
Xenix Programming,
Chapter 3, pp.3-1 to 3-12, 1984

Intel Corp.:
C Compiler Manual, 1984
Human Interface Manual, 1984

Jachner, J., Agarwal, V. K.:
"Data Flow Anomaly Detection",
IEEE Transactions on Software Engineering
Vol. 10 No. 4, pp. 432-437 July, 1984

Johnson, R. E.:
"Type-checking Smalltalk",
OOPSLA '86. Object-Oriented Programming Systems,
Languages and Applications.
SIGPLAN Notices
Vol.21, no.11 pp. 315-321 November, 1986

Kernighan, B. W., Ritchie, D. M.:
The C Programming Language,
Prentice-Hall, 1978

Kieburtz, R. B., Barabash, W., Hill, C. R.:
"A Type-Checking Program Linkage System for Pascal",
IEEE 3rd International Conference on Software Engineering,
1978, pp. 23-28

Knuth, D. E.:
"An empirical study of FORTRAN programes",
Software - Practice and Experience (GB),
Vol. 1, pp. 105-133 1971

Manner, R.:
"Strong Typing and Physical Units",
SIGPLAN Notices
Vol. 21, pp. 11-20 March, 1986

Miller, E. F.:
"Automated Software Engineering Tools"
ACM Professional Development Seminar
November, 1987 Kent State University

Ottenstein, K. J., Ottenstein, L. M.:
Debugging Information Obtained via Program Transformations",
MELECON '83 (IEEE).
Vol. 1, pp. A.7-15 May, 1983

Perelgut, S., Cordy, J. R.:
"Turing Plus: A Comparison with C and Pascal",
OOPSLA '86. Object-Orientated Programming Systems,
Languages and Applications.
SIGPLAN Notices
Vol.21, no.11 315-321 November, 1986

Plauger, P. J.:
"Standard C"
The C Users Journal,
Vol. 6 no.3, pp. 17-23 March/April, 1988

Shahdad, B. M.:
"A Survey of Static Analysis Features of Compilers",
IEEE Proceedings of the Symposium on Application and
Assessment of Automated Tools for Software Development,
1983, pp. 156-65

Stroustrup, B.:
"An Overview of C++",
SIGPLAN Notices,
Vol. 21 No. 10, pp. 7-18 October, 1986

Swan, R. B.:
"Common 'C' Bugs",
The C Journal,
Vol. 1 no.3, pp. 15-17, 1985

A. The Parameter Checker Tool

This appendix will discuss the development of the Parameter Checker program, how it collects data and how performs its analysis. Also included will be the non-compiler source code that went into the program.

A.1 Development of the Parameter Checker

As with any program, the development of the Parameter Checker required the establishment of project goals, the development of a strategy and the implementation of the strategy.

The primary reason for the parameter checker was that "C" compilers do not perform type-checking between parameters used in function definitions and invocations. Thus a programmer could write code using an improper invocation without realizing they had done so. The outcome can cause major problems when the program is actually executed. The errors caused by improper function calls are not always readily seen and can be time consuming to track done.

In determining the Parameter Checkers functionality, it was decided to check the validity of a function by the number of parameters used, the type (i.e. integer, character, etc.) of each parameter and the type of the value returned by the function itself. Other options added included the ability to obtain a function cross-reference map, the checking of "C" library functions and RMX system calls via predefined function definitions, and the suppression of type mismatch error diagnostics when the parameter compared are both pointer

objects.

The development of strategy started with the breaking of the project into segments. The first segment was to develop a command line parser to capture the input files, output destination and option selections. The second segment was to capture the necessary data from all the source code modules and save them off to disk. The segment was the development of the actual parameter analyzer.

Although the command line parser and the parameter analyzer had to be written from scratch, we were able to obtain the source code for a "C" compiler and decided to use it for the pre-processing and parsing portions of the data collection phase. We modified the "C" parser to write the needed data to the disk whenever it detected function definitions, declarations and invocations. In addition to the function and parameter information saved, the location of each occurrence was saved (source module filename and line number).

After all input files have been processed for data collection, the analysis is performed and optionally a cross-reference map is built.

The final stage involved writing a manager module to control the execution of each segment and report any system or resource errors encountered.

A.2 Data Collection

The parameter checker is concerned with the following occurrences:

a) the invocation of a function

b) the definition of a function

c) the declaration of a function

The initial key to detecting these items revolves around a variable that maintains the "lexical level". The "lexical level" will indicate one of three program parsing states. A level of zero (0) means that the current search point is outside of any subroutine. A level of one (1) means the search point has detected a function definition but has not yet entered the body of the function (i.e. the code enclosed by the main set of curly braces ("{}"). A level of two (2) means the search point is inside the body of a function.

At a lexical level of zero the program may encounter either function declarations (zero to many) or a function definition. When a non-reserved symbol, followed by a "(" is detected, the program will preserve all the following symbol information until a closing ")" is found. If the closing ")" is followed by a "," or ";", then the function symbol is tagged as a declaration, otherwise it is a definition.

If a declaration was found, it is logged in the INVOCATION storage area (a file named /work/inv_file) since both declarations and invocations must be matched up against a definition at a later time.

If a definition was detected, then the lexical level is now set to one (1). At this level the program will collect information on the parameters for the function. Initially the symbols found between the "(" and the ")" were stored with default information. Now those symbols will be modified with the

parameter definitions and this information will be stored in the DEFINITION storage area (a file named /work/def_file).

After the parameter information, the program will find the opening "{" to signify the beginning of the function body (i.e. lexical level two (2)). At this level, the program may encounter either function declarations (zero to many) or function invocations (zero to many).

When a pattern of reserved symbol, non-reserved symbol, and a "(" is detected, the program has detected a function declaration. The declaration is also logged in the INVOCATION storage area.

When a pattern of a non-reserved symbol followed by a "(" is detected, the program has found a function invocation. The program will preserve all the following symbol information until a closing ")" is found. This information is then written to the INVOCATION storage area.

In addition to the symbol information that is captured for each event, the parameter checker module saves the name of the source code file being processed, the line number within the file and the name of the current function body being processed.

A.3 Data Analysis

After all the input modules have been passed through the data collection phase, the actual resolution of definitions versus invocations and declarations begins.

The first phase of checking is called "definition driven"

parameter checking. In this step, the presence of an entry in the definition storage area will cause the invocation area to be scanned. If an invocation entry is found for the specified function, then parameter checking will take place.

The parameter checking involves comparing the number of parameters and the type of each parameter. If a parameter is a pointer, then the definition parameter and the invocation parameter are checked to make sure that they are of the same depth (i.e. single pointer: [char *] vs. double pointer: [char **]). If a parameter has a type of "structure", then the size of the structure types are checked for a mismatch.

If a declaration entry is found for the specified function, then the function return value of the definition will be compared to the declaration. Any mismatch in type will be noted.

If no invocation area entries are found to match a definition then the function is flagged as being "unused". Defined, but not invoked functions can be targeted for removal.

The second phase switches to "invocation driven" parameter checking. This optional step, searches the invocation table and seeks to match up the remaining entries against the function definitions in specified libraries. The files of library definitions are empty functions with their parameters declared. These files are processed like any other source code module. The library file is read and the function definition information is put into a new definition storage area. The definition area is then scanned for matches with the remaining invocation entries. Any invocation entries that are left after the library

scans are completed will be flagged as undefined functions.

Library files were provided for the standard "C" library functions and all iRMX operating system calls. Additional entries to these files could be made with any text editor.

A.4 Parameter Checker Development Items

The source code included in section A.4.1 handles the parameter data storage, retrieval and comparison functions. The source code from the modified compiler is not included as it is proprietary information. The "#include" files listed in section A.4.2 contain structure definitions that will save all of the needed data to do parameter checking.

A.4.1 "C" Code for Parameter Checker

The following source code modules are listed (see listing #1):

1) param_chk.c - this is the main line routine which calls the preprocessor and parser for each routine, and finally the parameter checker routines to analyze the data.

2) com_parse.c - this module is concerned with parsing the RMX command line format and setting up the appropriate files and option flags.

3) pcheck.c - this module takes the stored data and does all the parameter and

function checks described in this project.

4) dsk_hndlr.c - this module is concerned with saving the interim symbol data on the disk. Only a small function table is kept in memory, the rest of the data is stored in temporary files until the parameter checking is done.

A.4.2 Include Files for the Parameter Checker

The following include files are listed (see listing #2):

1) tbl_str.h - this file contains a) all the temporary data structures used to hold symbol information during parsing, and b) the structures for the records in the temporary disk files.

2) def.h - this file contains various defines for use in the parameter checker source code.

A.4.3 MAKE File for the Parameter Checker

This MAKE file is used for compiling and linking the application in an iRMX-86 environment (see listing #3).

A.4.4 Notes on the Implementation of a Parameter Checker

The parameter checker builds two temporary files on the

disk to save the bulk of the needed information. In order to prevent the application from spending an inordinate amount of time accessing the disk during the data collection phase, a table of function definition names and function invocation names are kept in memory tables as an index.

The definition disk file is just a series of records. The invocation disk file resembles a "linked-list" format. Each disk record contains room for an offset that will point to the next occurrence of a function invocation. These offsets are filled in as new invocations/declarations are added to the disk file.

The memory based index table for the definition data contains the function names and the starting location for the function's definition record. This table is scanned during the actual parameter checking. The memory based index table for the invocation file contains the function names, the offset for the first record of that function type and the offset for the last record of that function type. During the data collection phase, new invocation records are appended to the file and the offset in the previous record is updated.

The index tables are globals and are a constant size. They were made a constant size due to memory allocation characteristics of the iRMX-86 operating system. Under iRMX-86, repeated memory allocations can cause the available memory blocks to become very fragmented over a

period of time. This condition causes the compiler parser to fail on a large memory request. Therefore the tables were made a static size.

The size of the tables dictates the number of unique function definition names and invocation names that can be tracked. To increase the size of these tables, simply change the DEF_MAP_SIZE and INV_MAP_SIZE defines in the "tbl_str.h" file. The default sizes are 325 definitions and 375 invocations. On systems that are not-virtual memory, you may need to try several attempts to find the maximum possible table size for a given hardware memory size. If you're fortunate enough to port this program to a virtual memory system (or one with huge amounts of available memory), you may wish to redo the table allocation method to make it more flexible (i.e. dynamic allocation).

A.4.5 Use of the Parameter Checker

Information on how to invoke the parameter checker is available on-line (see figure A.1). To display the invocation instructions, enter just the name of the executable (param_chk) on the command line.

(Figure A.1)

FORMAT FOR CALL TO PARAM_CHK

<Version 2.20 DefTbl=325 InvTbl=375>

PARAM_CHK input_file [preposition output_file] [flags] [controls]

where:

INPUT_FILE :

direct addressing : any valid filename, wildcards allowed

indirect addressing : @filename (contains filenames; one per line)

PREPOSITION: to, over, or after

OUTPUT_FILE: a single file pathname, no wildcard characters allowed

FLAGS : one or more of the following separated by blanks:

c - include "C" library functions
p - pointer type mismatch suppression
r - include RMX functions
x - include function cross reference map

CONTROLS : include(name) - directs the preprocessor in it's search for #include files

define(name[,value]) - name is defined to have given value; default value equals 1

undefine(name) - removes one of the pre-processor's built-in definitions

```
----------------------
>>> File: param_chk.c
----------------------


/*****************************************************************************
**                                                                          **
**  LISTING #1  -   'C' Source code for Parameter Checker                   **
**                                                                          **
*****************************************************************************/


#include "ctype.h"
#include "udi.h"
#include "cc0.h"
#include "tbl_str.h"
#include "def.h"

#define CPP_FILE  ":work:expand"    /* preprocessor work file        */
#define DEF_FILE ":work:def_file"   /* definition file name          */
#define INV_FILE ":work:inv_file"   /* invocation file name          */
#define NFNAME   80                 /* maximum length of a file name */

/*****************************************************************************
 * Version buffer :
 *            any modification to VersNumMsg must also be made in this buffer.
*****************************************************************************/

#ifdef PC_LARGE
char VERSION[] = "program_version_number=2.20program_name=PARAM_CHK08/12/87";
char VersNumMsg[] = "Version 2.20  DefTbl=350  InvTbl=400";
#else
char VERSION[] = "program_version_number=2.20program_name=PARAM_CHK08/12/87";
char VersNumMsg[] = "Version 2.20  DefTbl=325  InvTbl=375";
#endif

char file[NFNAME];              /* current file being processed          */
char main_fname[NFNAME];        /* main filename; used for table data    */
DMAP DefMap[DEF_MAP_SIZE];      /* pointer to Definition Map             */
FILE *ifp;                      /* FILE pointer to the input file        */
FILE *ofp;                      /* preprocessor output FILE pointer      */
FILE *IndFilePtr;               /* pointer to file when using indirect   */
                                /* addressing.                           */
FILE *DefFilePtr;               /* pointer to function definition file   */
FILE *InvFilePtr;               /* pointer to function invocation file   */
IMAP InvMap[INV_MAP_SIZE];      /* pointer to Invocation Map             */
int  DmapCount;                 /* number of entries in Definition Map   */
int  ImapCount;                 /* number of entries in Invocation Map   */
int IndirectFlg;                /* signals indirect addressing used      */
int line;                       /* current line number                   */
jmp_buf death;

int LibFlg;                     /* indicates processing library file    */
/*
```

```
 *  The following are the possible switches enabled by the user.
*/
int C_flag;                     /* include 'C' library functions        */
int P_flag;                     /* pointer type mismatch suppression    */
int R_flag;                     /* include RMX functions                */
int X_flag;                     /* include function cross reference map */

FILE *open_file();              /* opens a file                         */
int close_file();               /* closes a file                        */
int get_mode      ();           /* determines file access mode          */
int get_next_file ();           /* obtains next file to process         */
int parse_command ();           /* parses the command line              */
int process_file  ();           /* processes the input file             */
void create_strings ();         /* transforms characters into strings   */
void free_dmap  ();             /* frees current definition map         */
void inv_pcheck ();             /* invocation driven parameter checker  */
void parm_chk   ();             /* definition driven parameter checker  */
void print_undef();             /* prints out undefined functions       */
void prt_format ();             /* prints call format to the terminal   */
void prt_header ();             /* prints header to the terminal        */


/******************************** Diebold 1000 ********************************
*
*1 Function:    main                        Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Driver for Parameter Checker
*
*3 Invocation:  Not invoked
*
*4 Inputs:      None
*
*5 Outputs:     None
*
*6 Caveats:     Memory for Definition Map is freed and DefMap set to NULL
*               if processing of libraries is necessary.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
******************************************************************************/
/*
*   a) parse the commmand line.
*   b) open output file.
*   c) process input files
*   d) compare invocation and definition tables.
*   e) process library files if flags set.
*   f) close output file.
*/

main ()
{
    char ctrl_buf[MAX_STR_LEN];   /* buffer for preprocessor controls   */
    char in_fname [MAX_STR_LEN];  /* input filename buffer              */
    char mode [2];                /* file access mode                   */
```

```
char out_fname [MAX_STR_LEN]; /* output filename buffer             */
char prep_val;                /* preposition value                  */
FILE *out_fptr;               /* pointer to user output file        */
int  cppargc;                 /* number of cpp controls used        */
int  undef_inv_flg;           /* flag indicating undefined functions*/

/*************************************************************************/
/*      Initialize variables.                                            */
/*************************************************************************/
C_flag = FALSE;
P_flag = FALSE;
R_flag = FALSE;
X_flag = FALSE;
IndirectFlg = FALSE;
LibFlg = FALSE;
ctrl_buf[0] = '\0';
cppargc = 0;
DmapCount = 0;
ImapCount = 0;

/*************************************************************************/
/*      Parse the command line.                                          */
/*************************************************************************/

if (parse_command (in_fname, out_fname, &prep_val, ctrl_buf, &cppargc)
    != OK)
{
    prt_format ();
    exit(ERROR);
}
/*************************************************************************/
/*      Print header.                                                    */
/*************************************************************************/

prt_header ();

/*************************************************************************/
/*      Determine file access mode for output file.                      */
/*************************************************************************/

if (get_mode (prep_val, mode) != OK)
{
    prt_format ();
    printf("ERROR : Invalid preposition value\n\n");
    exit(ERROR);
}

/*************************************************************************/
/*      Open output file for required access.                            */
/*************************************************************************/
if ((out_fptr= open_file (out_fname, mode)) == NULL)
{
    exit(ERROR);
}
printf ("\n\tOutput will be written to %s\n\n", out_fname);
fflush(stdout);
```

```
/*************************************************************************/
/*      Open function definition and invocation files for writing.       */
/*************************************************************************/
if ((InvFilePtr = open_file (INV_FILE, "wb")) == NULL)
    exit (ERROR);
if ((DefFilePtr = open_file (DEF_FILE, "wb")) == NULL)
    exit (ERROR);

/*************************************************************************/
/*      Transform control buffer from a buffer containing characters     */
/*      to a buffer containing character strings.                        */
/*      (This is necessary for the preprocessor.)                        */
/*************************************************************************/
create_strings (ctrl_buf);

/*************************************************************************/
/*      Process given input files.                                       */
/*************************************************************************/
while (*in_fname)
{

    /*********************************************************************/
    /*      Open input file tp process.                                  */
    /*********************************************************************/

    if (process_file (in_fname, ctrl_buf, cppargc) != OK)
    {
        printf ("ERROR while processing %s\n", in_fname);
        if(str_comp(out_fname, ":co:"))
            fprintf(out_fptr, "ERROR while processing %s\n", in_fname);
        break;
    }
    /*
     *  Obtain next file to be processed.
    */
    if (get_next_file (in_fname) != OK)
    {
        printf ("ERROR : Unable to obtain next file\n");
        exit (ERROR);
    }
}
/*
 *  Close definition and invocation files.
*/
if (close_file(InvFilePtr, INV_FILE) == ERROR)
    exit(ERROR);
if (close_file(DefFilePtr, DEF_FILE) == ERROR)
    exit(ERROR);
/*
 *  Reopen files for reading.
*/
if ((InvFilePtr = open_file (INV_FILE,"rb")) == NULL)
    exit(ERROR);
if ((DefFilePtr = open_file (DEF_FILE,"rb")) == NULL)
    exit(ERROR);
/*
 *  Compare definitions and invocations.
```

```
*/
parm_chk(out_fptr);

/*
 *  Close definition file.
*/
if (close_file (DefFilePtr, DEF_FILE) == ERROR)
    exit(ERROR);
/*
 *  Determine if any undefined functions.
*/
if (undef_inv_flg = check_undef())
{
    LibFlg = 1;         /* Processing a library; suppress strict messages */
    if (C_flag)         /* Process 'C' library file */
    {
        /*
         *  Open definition file and reinitialize Definition Map.
        */
        if ((DefFilePtr = open_file (DEF_FILE, "wb")) == NULL)
            exit(ERROR);
        DmapCount = 0;
        if (process_file (":sd:param_chk/clib.def", ctrl_buf, cppargc)
        != OK)
        {
            printf
             ("ERROR while processing \":sd:param_chk/clib.def\"\n");
            if(str_comp(out_fname, ":co:"))
            {
                fprintf(out_fptr,
                  "ERROR while processing \":sd:param_chk/clib.def\"\n");
            }
            exit(ERROR);
        }
        if (close_file(DefFilePtr, DEF_FILE) == ERROR)
            exit(ERROR);
        /*
         * Perform parameter checking.
        */
        if ((DefFilePtr = open_file (DEF_FILE, "rb")) == NULL)
            exit(ERROR);
        inv_pcheck(out_fptr,":sd:param_chk/clib.def");
        if (close_file(DefFilePtr, DEF_FILE) == ERROR)
            exit(ERROR);
        undef_inv_flg = check_undef();
    }
    if (undef_inv_flg)
    {
        /*
         *  Process RMX library and perform parameter checking.
        */
        if (R_flag)
        {
            if ((DefFilePtr = open_file (DEF_FILE, "wb")) == NULL)
                exit(ERROR);
            DmapCount = 0;
            if (process_file (":sd:param_chk/rmx.def", ctrl_buf, cppargc)
```

```
                != OK)
            {
                printf
               ("ERROR while processing \":sd:param_chk/rmx.def\"\n");
                if(str_comp(out_fname, ":co:"))
                {
                    fprintf(out_fptr,
                    "ERROR while processing \":sd:param_chk/rmx.def\"\n");
                }
                exit(ERROR);
            }
            if (close_file(DefFilePtr, DEF_FILE) == ERROR)
                exit(ERROR);
        /*
         *  Open definition file for read and perform parameter checking.
        */
            if ((DefFilePtr = open_file (DEF_FILE, "rb")) == NULL)
                exit(ERROR);
            inv_pcheck(out_fptr, ":sd:param_chk/rmx.def");
            if (close_file(DefFilePtr, DEF_FILE) == ERROR)
                exit(ERROR);
            undef_inv_flg = check_undef();
        }
        if (undef_inv_flg)
        {
            /*
             * List undefined functions.
            */
            print_undef(out_fptr);
            fflush(stdout);
        }
    }
}
/*
 *  Close invocation file.
*/
 if (close_file(InvFilePtr, INV_FILE) == ERROR)
    exit(ERROR);

/*************************************************************************/
/*      Close output file.                                               */
/*************************************************************************/

if (close_file (out_fptr, out_fname) == ERROR)
    exit (ERROR);

/*************************************************************************/
/*      Close Indirect file if indirect addressing used.                 */
/*************************************************************************/
if (IndirectFlg)
{
    if (fclose (IndFilePtr) == ERROR)
    {
    printf("ERROR: Unable to close indirect address file\n");
    exit(ERROR);
    }
}
```

```
    /*************************************************************************/
    /*      Free disk space used by work files.                              */
    /*************************************************************************/
    fflush(stdout);
/*  open_def_file("w");
    open_inv_file("w");
    close_def_file ();
    close_inv_file ();*/

    printf ("\n\tPARAMETER CHECKING COMPLETE\n");
    exit (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    process_file                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Expands source file and makes a call to the parser.
*
*3 Invocation:  process_file (in_fname_ptr)
*
*4 Inputs:      in_fname_ptr (char *) = pointer to input filename
*
*5 Outputs:     Function returns:   0  - no errors
*                                  -1 - error
*
*6 Caveats:     Function definitions and invocations are added to their
*               respective tables.
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*************************************************************************/
/*
*   a) Expand input file.   (Preprocessor opens and closes the input file.)
*   b) Open expanded file and send to the parser.
*   c) Close expanded file.
*/

int process_file (in_fname_ptr, ctrl_buf_ptr, cppargc)
char *in_fname_ptr;
char *ctrl_buf_ptr;
int  cppargc;
{
    char *strncpy();        /* standard string copy */

    printf ("\n\tProcessing %s\n", in_fname_ptr);
    /*
     *  Expand input source file.
    */
    if (expand_file (in_fname_ptr, ctrl_buf_ptr, cppargc) != OK)
    {
        return (ERROR);
    }
    strncpy(main_fname, in_fname_ptr, NFNAME - 1);
```

```
    main_fname[NFNAME - 1]= '\0'; /* necessary for proper source filename*/
    if ((ifp = open_file(CPP_FILE, "r")) == NULL)
    {
        return(ERROR);
    }
    /*
     *  Parse expanded file, building function definition and
     *  invocation tables.
    */
    if (cc0 () != OK)
    {
        close_file (ifp, CPP_FILE);
        return(ERROR);
    }
    if (close_file (ifp, CPP_FILE) == ERROR)
    {
        return(ERROR);
    }
    return (OK);
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    get_mode                        Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Determines proper mode to open file.
*
*3 Invocation:  get_mode (prep_val, mode_ptr)
*
*4 Inputs:      prep_val (int)     = value of preposition (TO, OVER, AFTER)
*               mode_ptr (char *) = pointer to buffer for mode string
*
*5 Outputs:     Function returns:   0  - proper mode
*                                  -1 - improper mode
*
*6 Caveats:
*
*7 Author:  Christopher J. Knouff           Date: 09 June 1985
*           John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
******************************************************************************/

int get_mode (prep_val, mode_ptr)
char prep_val;
char *mode_ptr;
{
    char *strcpy();         /* standard string copy */

    switch ( prep_val)
    {
        case 1 :
        case 2 :
                strcpy (mode_ptr,"w");  /* TO or OVER preposition          */
                break;
        case 3 :
```

```
                strcpy (mode_ptr,"a");  /* AFTER preposition encountered    */
                break;
        default :
                *mode_ptr = '\0';       /* incorrect format encountered     */
                return (ERROR);
                break;
    }
    return (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    prt_format                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Prints proper format for invocation to Parameter Checker
*
*3 Invocation:  prt_format ()
*
*4 Inputs:      None
*
*5 Outputs:     None
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void prt_format ()
{
    printf("%c", 26);                                   /* clear the screen         */
    printf("\t\t   %c%c%c", 27, 71, 52);                /* reverse mode             */
    printf(" FORMAT FOR CALL TO PARAM_CHK ");
    printf("%c%c%c", 27, 71, 48);                       /* normal mode              */
    printf ("      <%s>\n\n", VersNumMsg);              /* Version number           */
    printf("PARAM_CHK input_file [preposition output_file] [flags] ");
    printf("[controls]\n");
    printf("where :\n");
    printf("\tINPUT FILE :\n\t\tdirect addressing   : ");
    printf("any valid filename, wildcards allowed\n");
    printf("\t\tindirect addressing : @filename\n");
    printf("\tPREPOSITION: to, over, or after\n");
    printf("\tOUTPUT_FILE: a single file pathname, no wildcard characters");
    printf(" allowed\n");
    printf("\tFLAGS      : one or more of the following separated");
    printf("  by blanks:\n");
    printf("\t\t\tc - include 'C' library functions\n");
    printf("\t\t\tp - pointer type mismatch suppression\n");
    printf("\t\t\tr - include RMX functions\n");
    printf("\t\t\tx - include function cross reference map\n");
    printf("\tCONTROLS   :\n");
    printf("\t\tinclude(name)          - directs preprocessor in search for\n");
    printf("\t\t                         #include files\n");
    printf("\t\tdefine(name[,value]) - name is defined to have given\n");
```

```
    printf("\t\t                          value; default value equals 1\n");
    printf("\t\tundefine(name)          - removes one of preprocessor's\n");
    printf("\t\t                          built-in definitions\n");
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    prt_header                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Prints header after Parameter Checker is invoked.
*
*3 Invocation:  prt_header ()
*
*4 Inputs:      None
*
*5 Outputs:     None
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void prt_header ()
{
    extern char VersNumMsg[];   /* Message displaying version number        */

    /*************************************************************************/
    /*      Print main heading.                                              */
    /*************************************************************************/

/*   Comment out graphics characters  */

/*  printf ("%c", 26);                          */ /* clear the screen      */
/*  printf ("          %c%c%c", 27, 71, 52);*/ /* reverse mode             */
    printf (" PARAMETER CHECKER ");
/*  printf ("%c%c%c", 27, 71, 48);              */ /* normal mode           */
    printf ("  <%s>\n\n", VersNumMsg);             /* Version number        */

    /*************************************************************************/
    /*      Display options set by the user.                                 */
    /*************************************************************************/

    if ( C_flag || P_flag || R_flag || X_flag)
    {
        printf("\tThe following options will be included :\n");
        if ( C_flag)
        {
            printf ("\t\t-include 'C' library functions\n");
        }
        if (P_flag)
        {
            printf ("\t\t-pointer type mismatch suppression\n");
        }
```

```
        if (R_flag)
        {
            printf ("\t\t-include RMX functions\n");
        }
        if (X_flag)
        {
            printf ("\t\t-include function cross reference map\n");
        }
    }
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    get_next_file                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Obtains filename of next file to process.
*
*3 Invocation:  get_next_file (fname_ptr)
*
*4 Inputs:      fname_ptr (char *)  = pointer to buffer to store file name
*
*5 Outputs:     Function returns:   0  - no errors encountered
*                                   -1 - error
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int get_next_file (fname_ptr)
char *fname_ptr;
{
    int get_in_fname ();            /* obtains name of file to process      */

    if (get_in_fname (fname_ptr) != OK)
    {
        return (ERROR);
    }
    return (OK);
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    expand_file                  Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Expands the given source file by invoking preprocessor.
*
*3 Invocation:  expand_file (in_fname_ptr)
*
*4 Inputs:      in_fname_ptr (char *)   = pointer to name of file to expand
*
*5 Outputs:     Function returns:   0  - no errors encountered
*                                   -1 - error
```

```
*
*6 Caveats:     Preprocessor is invoked as an external job.
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*************************************************************************

int expand_file (in_fname_ptr, ctrl_buf_ptr, cppargc)
char *in_fname_ptr;
char *ctrl_buf_ptr;
int  cppargc;
{
    int status;

    if ((ifp = open_file(in_fname_ptr, "r")) == NULL)
    {
        return(ERROR);
    }
    if ((ofp = open_file(CPP_FILE, "w")) == NULL)
    {
        return(ERROR);
    }
    status = cpp(in_fname_ptr, ctrl_buf_ptr, cppargc);
    if (close_file(ifp, in_fname_ptr) == ERROR)
    {
        status = ERROR;
    }
    if (close_file(ofp, CPP_FILE) == ERROR)
    {
        status = ERROR;
    }
    return(status);
}

FILE *open_file (filename, mode)
char *filename;
char *mode;
{
    FILE *file_ptr;

    if ((file_ptr = fopen(filename, mode)) == NULL)
    {
        printf("ERROR : Unable to open \"%s\"\n", filename);
    }
    return (file_ptr);
}

int close_file (file_ptr, filename)
FILE *file_ptr;
char *filename;
{
    if (fclose(file_ptr))
    {
        printf("ERROR : Unable to close \"%s\"\n", filename);
```

```
        return(ERROR);
    }
    return(OK);
}

void create_strings (buffer)
char *buffer;
{
    char *buf_ptr;

    for (buf_ptr = buffer; *buf_ptr != '\0'; buf_ptr++)
    {
        if (*buf_ptr == ' ')
            *buf_ptr = '\0';
    }
}
```

>>> File: com_parse.c

```
#include "ctype.h"
#include "stdio.h"
#include "def.h"

extern char defsym[];           /* default value for defines            */
extern FILE *IndFilePtr;        /* pointer to file containing the list  */
                                /* of files to process.                 */
extern int IndirectFlg;         /* signals indirect addressing used     */

/* the following are the possible switches enabled by the user          */

extern int C_flag;              /* include 'C' library functions        */
extern int P_flag;              /* pointer type mismatch suppression    */
extern int R_flag;              /* include RMX functions                */
extern int X_flag;              /* include function cross reference map */

/******************************** Diebold 1000 *********************************
*
*1 Function:    parse_command           Copyright (c) Diebold Inc. 1985
*
*2 Summary:     parse_command parses the command line generated by the call
*               to Parameter Checker.
*
*3 Invocation:  parse_command (in_fname_ptr, out_fname_ptr, prep_val_ptr,
*                             ctrl_buf_ptr, argc_ptr)
*
*4 Inputs:      in_fname_ptr  - pointer to input filename buffer
*               out_fname_ptr - pointer to output filename buffer
*
*5 Outputs:     prep_val_ptr  - pointer to preposition value
*               ctrl_buf_ptr  - pointer to the command line buffer for the
*                               'C' preprocessor
*               argc_ptr      - pointer number of arguments in command line
*                               'C' preprocessor
```

```
*
*               Returned Value:
*                      0 - no errors encountered
*                     -1 - error
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff                Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

int parse_command (in_fname_ptr,out_fname_ptr,prep_val_ptr,ctrl_buf_ptr,
                   argc_ptr)
char *in_fname_ptr;
char *out_fname_ptr;
char *prep_val_ptr;
char *ctrl_buf_ptr;
int  *argc_ptr;
{
    char param_buf [MAX_STR_LEN];   /* parameter buffer                    */

    int     get_flags ();           /* obtains input flags                 */
    int     get_in_fname ();        /* obtains input file                  */
    int     get_out_fname ();       /* obtains output file                 */
    int     get_prep ();            /* obtains the preposition             */

    /*************************************************************************/
    /*      Obtain input filename.                                           */
    /*************************************************************************/

    if ((get_in_fname (in_fname_ptr) != OK) ||
        (*in_fname_ptr == NULL_CHAR))
    {
        return (ERROR);
    }

    /*************************************************************************/
    /*      Obtain preposition.                                              */
    /*************************************************************************/

    if ( get_prep (param_buf, prep_val_ptr, ctrl_buf_ptr, argc_ptr) != OK)
    {
        return (ERROR);
    }
    if (*prep_val_ptr > 0)
    {
        /*********************************************************************/
        /*      Obtain output filename.                                      */
        /*********************************************************************/

        if (get_out_fname(out_fname_ptr, *prep_val_ptr) != OK)
        {
            return (ERROR);
        }
```

```
    }

    /*************************************************************************/
    /*      Obtain and set option flags if more parameters to process.       */
    /*************************************************************************/

    if (param_buf[0])
    {
        if (*prep_val_ptr != 0)      /* a preposition and output filename or */
        {                            /* a keyword was obtained, not a flag;  */
                                     /* thus contents of param_buf no longer */
                                     /* needed since it was already processed*/
            param_buf[0] = NULL_CHAR;
        }
        if(*prep_val_ptr == KEYWORD)
        {
            *prep_val_ptr = DFLT_PREP;  /* default preposition */
        }
        if (get_flags (param_buf, ctrl_buf_ptr, argc_ptr) != OK)
        {
            return (ERROR);
        }
    }
    if (*prep_val_ptr == DFLT_PREP)
    {
        strcpy(out_fname_ptr,":co:");   /* output defaults to std. output   */
        *prep_val_ptr = TO_PREP;        /* preposition defaults to "TO"     */
    }
    return (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    get_in_path              Copyright (c) Diebold Inc. 1985
*
*2 Summary:     get_in_path obtains path_name of the input file.
*
*3 Invocation:  get_in_path (path_ptr);
*
*4 Inputs:      None
*
*5 Outputs:     Output Parameters :
*                     path_ptr     - pointer to file pathname
*
*               Returned value    :
*                      0 - no errors encountered
*                     -1 - error
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*************************************************************************/
```

```
int get_in_path (path_ptr)
char *path_ptr;
{
    unsigned sys_status;                    /* system status */

    void RQ$C$GET$INPUT$PATHNAME ();

    RQ$C$GET$INPUT$PATHNAME ( path_ptr, (unsigned)MAX_STR_LEN, &sys_status);
    if (sys_status != OK)
    {
        return (ERROR);
    }
    return (OK);
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    get_in_fname                  Copyright (c) Diebold Inc. 1985
*
*2 Summary:     get_in_fname obtains name of file to process.
*
*3 Invocation:  get_in_fname (fname_ptr);
*
*4 Inputs:      None
*
*5 Outputs:     Output Parameters :
*                   fname_ptr    - pointer to file to be processed
*
*               Returned value    :
*                      0 - no errors encountered
*                     -1 - error
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int get_in_fname (fname_ptr)
char *fname_ptr;
{
    int chk_in_file ();                     /* checks validity of file      */
    int get_in_path ();                     /* obtains file pathname        */

    if (IndirectFlg)                        /* check for indirect addressing    */
    {
        if (get_fname_ind (fname_ptr) != OK)
        {
            *fname_ptr = NULL_CHAR;
            return (OK);
        }
    }
    else
    {
```

```
    /*****************************************************************/
    /*      Obtain input pathname.                                   */
    /*****************************************************************/

    if ( get_in_path (fname_ptr) != OK)
    {
        return (ERROR);
    }

    if (!(*fname_ptr))                  /* return if no file to process */
    {
        return (OK);
    }

    /*****************************************************************/
    /*      Check for indirect addressing.                           */
    /*****************************************************************/

    if (*fname_ptr == '@')
    {
        IndirectFlg = 1;

        /*************************************************************/
        /*      Check validity of file.                              */
        /*************************************************************/

        if ( chk_in_file ( fname_ptr + 1) != OK)
        {
            return (ERROR);
        }

        /*************************************************************/
        /*      Open file containing filename list and obtain the name  */
        /*      of the first file to be processed.                   */
        /*************************************************************/

        if (!(IndFilePtr = fopen (fname_ptr + 1, "r")))
        {
            return (ERROR);
        }

        if (get_fname_ind (fname_ptr) != OK)
        {
            return (ERROR);
        }

    }   /* end if '@' */
}   /* end if-then-else */

/*****************************************************************/
/*      Determine validity of file to be processed.              */
/*****************************************************************/

if (chk_in_file (fname_ptr) != OK)
{
    return (ERROR);
}
```

```
    return (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    chk_in_file              Copyright (c) Diebold Inc. 1985
*
*2 Summary:     check if input filename exists.
*
*3 Invocation:  chk_in_file (pathname_ptr);
*
*4 Inputs:      pathname_ptr - pointer to pathname of file
*
*5 Outputs:     Returned value:
*                     0 - file valid
*                    -1 - file not valid (error)
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int chk_in_file (pathname_ptr)
char *pathname_ptr;
{
    unsigned connect;                           /* connection to the file   */
    unsigned sys_status;                        /* system status            */

    unsigned RQ$C$GET$INPUT$CONNECTION ();
    void     RQ$S$DELETE$CONNECTION    ();

    connect = RQ$C$GET$INPUT$CONNECTION (pathname_ptr, &sys_status);
    if (sys_status != OK)
    {
        return (ERROR);
    }

    RQ$S$DELETE$CONNECTION (connect, &sys_status);
    if (sys_status != OK)
    {
        return (ERROR);
    }
    return (OK);    /* file valid */
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    get_out_fname              Copyright (c) Diebold Inc. 1985
*
*2 Summary:     get_out_fname obtains output file
*
*3 Invocation:  get_out_fname (fname_ptr, prep_val);
```

```
*
*4 Inputs:       None.
*
*5 Outputs:      fname_ptr    - name of output file
*                prep_val     - preposition value (TO, OVER or AFTER)
*
*                Returned value    :
*                       0 - no errors encountered
*                      -1 - error
*
*6 Caveats:      None.
*
*7 Author:       Christopher J. Knouff                Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int get_out_fname (fname_ptr, prep_val)
char *fname_ptr;
char prep_val;
{
    /*******************************************************************************/
    /*      Obtain output pathname.                                                */
    /*******************************************************************************/

    if (get_out_path (fname_ptr) != OK)
    {
        return (ERROR);
    }

    /*******************************************************************************/
    /*      Check validity of file.                                                */
    /*******************************************************************************/

    if (chk_out_file (fname_ptr, prep_val) != OK)
    {
        return (ERROR);
    }
    return (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:     get_out_path                Copyright (c) Diebold Inc. 1985
*
*2 Summary:      get_out_path obtains output pathname
*
*3 Invocation:   get_out_path (fname_ptr, prep_val);
*
*4 Inputs:       None.
*
*5 Outputs:      path_ptr     - pointer to filename path
*
*                Returned value:
*                       0 - no errors encountered
```

```
*                    -1 - error
*
*6 Caveats:      None.
*
*7 Author:       Christopher J. Knouff                Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

int get_out_path (path_ptr)
char *path_ptr;
{
    char key_buf [MAX_STR_LEN];          /* keyword buffer                    */
    struct str_tab_tag string_tbl;       /* string table                      */

    int get_param ();

    if (get_param (key_buf, &string_tbl) != OK)
    {
        return (ERROR);
    }
    if ((key_buf[0]) || (string_tbl.num != 1))
    {
        *path_ptr = '\0';
        return(ERROR);
    }
    strcpy(path_ptr, string_tbl.string);
    return (OK);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:     chk_out_file             Copyright (c) Diebold Inc. 1985
*
*2 Summary:      chk_out_file determines the validity of the output file.
*
*3 Invocation:   chk_out_file (pathname_ptr, prep_val);
*
*4 Inputs:       pathname_ptr - pointer to pathname of file
*                prep_val     - preposition value
*
*5 Outputs:      Returned value:
*                      0 - file valid
*                     -1 - file not valid (error)
*
*6 Caveats:      None.
*
*7 Author:       Christopher J. Knouff                Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

int chk_out_file (pathname_ptr, prep_val)
```

```
char *pathname_ptr;
char prep_val;
{
    char *temp_ptr;                                 /* temporary file ptr       */
    unsigned connect;                               /* connection to the file   */
    unsigned sys_status;                            /* system status            */

    unsigned RQ$C$GET$OUTPUT$CONNECTION ();
    void RQ$S$DELETE$CONNECTION ();


    /*************************************************************************/
    /*      Check for the presence of wildcard characters.                   */
    /*************************************************************************/

    for (temp_ptr = pathname_ptr; *temp_ptr; temp_ptr++)
    {
        if ((*temp_ptr == '?') || (*temp_ptr == '*'))
        {
            return (ERROR);
        }
    }

    connect = RQ$C$GET$OUTPUT$CONNECTION (pathname_ptr,
                                          (unsigned char)prep_val,
                                          &sys_status);
    if (sys_status != OK)
    {
        return (ERROR);
    }

    RQ$S$DELETE$CONNECTION (connect, &sys_status);
    if (sys_status != OK)
    {
        return (ERROR);
    }
    return (OK);    /* file valid */
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    get_param                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     get_param makes the actual system call to obtain the parameter
*               from the command line.
*
*3 Invocation:  get_param (keyword_ptr, value_ptr)
*
*4 Inputs:      None.
*
*5 Outputs:     keyword_ptr  -  pointer to keyword portion of
*                               received parameter.
*                value_ptr    - pointer to value portion of
*                               received parameter.
*               Returned value:
*                         0 - no errors encountered
*                        -1 - invalid flag
```

```
*6 Caveats:      None.
*
*7 Author:       Christopher J. Knouff                Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

int get_param (keyword_ptr, value_ptr)
char *keyword_ptr;
struct str_tab_tag *value_ptr;
{
    char dummy_index;                       /* receives junk from HI call since */
                                            /* a preposition list is not used.  */
    char more;                              /* determines if a parameter was    */
                                            /* received from command line.      */
    unsigned int err_code;                  /* system error code                */
    unsigned char RQ$C$GET$PARAMETER (); /* HI call to obtain parameter     */

    more = RQ$C$GET$PARAMETER (keyword_ptr, (unsigned)MAX_STR_LEN,
              (char *)value_ptr,(unsigned)sizeof(struct str_tab_tag),
              &dummy_index, NULL, &err_code);

    if (err_code != OK)
    {
        return (ERROR);
    }
    return (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:     get_flags                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:      get_flags obtains any flags set by the user from the
*                command line.
*
*3 Invocation:   get_flags (flag_buf, ctrl_buf_ptr, argc_ptr)
*
*4 Inputs:       flag_buf      - pointer to buffer that will store
*                                the received flag.
*                ctrl_buf_ptr  - pointer to the command line buffer for the
*                                'C' preprocessor
*                argc_ptr      - pointer number of arguments in command line
*
*5 Outputs:      Returned value:
*                          0 - no errors encountered
*                         -1 - invalid flag
*6 Caveats:      None.
*
*7 Author:       Christopher J. Knouff                Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/
```

```
int get_flags (flag_buf, ctrl_buf_ptr, argc_ptr)
char *flag_buf;
char *ctrl_buf_ptr;
int *argc_ptr;
{
    char key_buf [MAX_STR_LEN];      /*  keyword buffer                      */
    struct str_tab_tag string_tbl;   /*  string table                        */

    int flag_chk ();                 /* checks validity of a given flag      */
    int keyword_chk ();              /* handles parameters with keywords     */

    /*************************************************************************/
    /*      Determine if a flag is already in the buffer.                    */
    /*************************************************************************/

    if (*flag_buf)
    {
        if (flag_chk (flag_buf) != OK)
        {
            return (ERROR);
        }
    }
    do {

    /*************************************************************************/
    /*      Obtain a parameter from the command line.                        */
    /*************************************************************************/

        if (get_param (key_buf, &string_tbl) != OK)
        {
            return (ERROR);
        }
        if (string_tbl.num == 0)
            *flag_buf = 0;
        else
            strcpy(flag_buf, string_tbl.string);
        if (key_buf[0])
        {
            /*
             *  Check keyword and place in command buffer.
            */
            if (keyword_chk(key_buf, &string_tbl,ctrl_buf_ptr, argc_ptr)
             != OK)
            {
                return(ERROR);
            }
        }
        else
        {
            /*********************************************************************/
            /*      Check validity of given flag if a flag was obtained.         */
            /*********************************************************************/

            if (*flag_buf)
            {
                if (flag_chk (flag_buf) != OK)
```

```
                {
                    return (ERROR);
                }
            }
        }
    } while (*flag_buf);    /* end do-while */

    return (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    flag_chk                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     flag_chk checks the validity of a flag. If valid, the
*               flag is set.
*
*3 Invocation:  flag_chk (flag_ptr)
*
*4 Inputs:      flag_ptr - pointer to flag to check
*
*5 Outputs:     Returned value:
*                        0 - valid flag
*                       -1 - invalid flag
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int flag_chk (flag_ptr)
char *flag_ptr;
{
    int valid_flg = TRUE;               /* signals valid flag               */

    if ( *flag_ptr == '-')              /* acceptable flag format: '-flag'  */
    {
        flag_ptr++;
    }

    if (*(flag_ptr + 1) == NULL_CHAR)   /* only one character flags allowed */
    {
        switch (toupper (*flag_ptr))
        {
            case 'C':
                    C_flag = TRUE;
                    break;
            case 'P':
                    P_flag = TRUE;
                    break;
            case 'R':
                    R_flag = TRUE;
                    break;
            case 'X':
```

```
                    X_flag = TRUE;
                    break;
            default :
                    valid_flg = FALSE;
                    break;
        }
        if (valid_flg)
        {
            return (OK);
        }
    }
    return (ERROR);
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    get_prep                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     get_prep obtains the preposition from the command line.
*
*3 Invocation:  get_prep (prep_buf, prep_val_ptr, ctrl_buf_ptr, argc_ptr)
*
*4 Inputs:      ctrl_buf_ptr  - pointer to the command line buffer for the
*                               'C' preprocessor
*               argc_ptr      - pointer number of arguments in command line
*
*5 Outputs:     prep_buf      - buffer to store the preposition string.
*               prep_val_ptr  - pointer to the preposition value:
*                                    0 - default preposition, 1 - TO
*                                    2 - OVER, 3 - AFTER
*
*               Returned value:
*                         0 - no errors
*                        -1 - error encountered
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int get_prep (prep_buf, prep_val_ptr, ctrl_buf_ptr, argc_ptr)
char *prep_buf;
char *prep_val_ptr;
char *ctrl_buf_ptr;
int  *argc_ptr;
{
    char key_buf [MAX_STR_LEN];         /* keyword buffer                    */
    struct str_tab_tag string_tbl;      /* string table                      */

    /*****************************************************************/
    /*      Obtain a parameter from the command line.                */
    /*****************************************************************/

    if (get_param (key_buf, &string_tbl) != OK)
```

```
    {
        return (ERROR);
    }
    if (key_buf[0])
    {
        /*
         *  Check keyword and place in command buffer if legitimate.
        */
        if (keyword_chk(key_buf, &string_tbl, ctrl_buf_ptr, argc_ptr) != OK)
        {
            return(ERROR);
        }
        *prep_val_ptr = KEYWORD;
    }
    else
    {
    /*************************************************************************/
    /*      If a parameter was obtained, check if a preposition.             */
    /*************************************************************************/

        strcpy(prep_buf, string_tbl.string);
        if (*prep_buf)
        {
            *prep_val_ptr = get_prep_val (prep_buf);
        }
        else
        {
            *prep_val_ptr = DFLT_PREP;  /* no preposition obtained, default */
        }
    }
    return (OK);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    get_prep_val                  Copyright (c) Diebold Inc. 1985
*
*2 Summary:     get_prep_val obtains the value of a given preposition
*
*3 Invocation:  get_prep_val (prep_ptr)
*
*4 Inputs:      prep_ptr      - pointer to the preposition string
*
*5 Outputs:     Returned value:     0 - default preposition, 1 - TO
*                                   2 - OVER, 3 - AFTER
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*************************************************************************/

int get_prep_val ( prep_ptr)
char *prep_ptr;
```

```
{
    if (!str_comp (prep_ptr, "TO"))
    {
        return (TO_PREP);
    }
    if (!str_comp (prep_ptr, "OVER"))
    {
        return (OVER_PREP);
    }
    if (!str_comp (prep_ptr, "AFTER"))
    {
        return (AFTER_PREP);
    }
    return (DFLT_PREP);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    str_comp                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     str_comp determines if two character strings are equivalent
*
*3 Invocation:  str_comp (str1_ptr, str2_ptr)
*
*4 Inputs:      str1_ptr - pointer to the first string
*               str2_ptr - pointer to the second string
*
*5 Outputs:     Returned value:
*                        0 - strings are equivalent
*                        1 - strings are not equivalent
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int str_comp (str1_ptr, str2_ptr)
char *str1_ptr;
char *str2_ptr;
{
    for ( ; toupper(*str1_ptr) == toupper(*str2_ptr) ;str1_ptr++,str2_ptr++)
    {
        if (*str2_ptr == NULL_CHAR)
        {
            return (0);
        }
    }
    return (1);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    get_fname_ind              Copyright (c) Diebold Inc. 1985
```

```
*
*2 Summary:     get_fname_ind obtains a filename when using indirect
8               addressing.
*
*3 Invocation:  get_fname_ind (fname_ptr)
*
*4 Inputs:      str1_ptr - pointer to the first string
*               str2_ptr - pointer to the second string
*
*5 Outputs:     fname_ptr - pointer to received file
*
*               Returned value:
*                     0 - received a filename
*                    -1 - unable to obtain string
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int get_fname_ind (fname_ptr)
char *fname_ptr;
{

    if (!fgets (fname_ptr, MAX_STR_LEN, IndFilePtr))
    {
        return (ERROR);
    }

    fname_ptr [strlen(fname_ptr) - 1] = NULL_CHAR; /* replace end-of-fname  */
                                                   /* marker with a null    */
    return (OK);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    keyword_chk             Copyright (c) Diebold Inc. 1985
*
*2 Summary:     keyword_chk places keywords in the control buffer used by
*               the 'C' preprocessor.
*
*3 Invocation:  keyword_chk (key_buf, string_tbl_ptr, ctrl_buf_ptr, argc_ptr)
*
*4 Inputs:      keyword             - keyword to be added
*               string_table_ptr    - table of pre-defined strings
*
*5 Outputs:     ctrl_buf_ptr  - pointer to the command line buffer for the
*                               'C' preprocessor
*               argc_ptr      - pointer number of arguments in preprocessor
*                               command line buffer
*
*               Returned value:
*                     0 - no error
```

```
*                   -1 - error adding to command buffer
*
*6 Caveats:      None.
*
*7 Author:       Christopher J. Knouff                Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

int keyword_chk (key_buf, string_tbl_ptr, ctrl_buf_ptr, argc_ptr)
char *key_buf;
struct str_tab_tag *string_tbl_ptr;
char *ctrl_buf_ptr;
int *argc_ptr;
{
    /*
     *  Place keyword in control buffer (used by the 'C' preprocessor).
    */
    if (concat_str(ctrl_buf_ptr, key_buf, argc_ptr) != OK)
        return(ERROR);
    if (!(str_comp(key_buf, "include")) || !(str_comp(key_buf, "undefine")))
    {
        if(string_tbl_ptr->num != 1)
        {
            return(ERROR);
        }
        if (concat_str(ctrl_buf_ptr, string_tbl_ptr->string, argc_ptr) != OK)
            return(ERROR);
    }
    else if (!(str_comp(key_buf, "define")))
    {
        if (string_tbl_ptr->num == 1)
        {
            if (concat_str(ctrl_buf_ptr,string_tbl_ptr->string,argc_ptr)
              != OK)
                return(ERROR);

            if (concat_str (ctrl_buf_ptr, defsym, argc_ptr) != OK)
                return(ERROR);
        }
        else if (string_tbl_ptr->num == 2)
        {
            if (concat_str(ctrl_buf_ptr,string_tbl_ptr->string,argc_ptr)
              != OK)
                return(ERROR);
        /*
         *  The address computation which serves as the second parameter of
         *  the following call obtains the position in the string table,
         *  which is used by Human Interface calls, of the second value
         *  received in the HI get$parameter call. The value, a string,
         *  represents the value of the identifier being defined.
        */
            if (concat_str (ctrl_buf_ptr, string_tbl_ptr->string +
                        strlen(string_tbl_ptr->string) + 1, argc_ptr) != OK)
                return(ERROR);
```

```
        }
        else
        {
            return(ERROR);
        }
    }
    else
    {
        return(ERROR);                  /* invalid keyword */
    }
    return(OK);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    concat_str                      Copyright (c) Diebold Inc. 1985
*
*2 Summary:     concat_str concatenates a string on to the end of the buffer
*               to be used by the 'C' preprocessor.
*
*3 Invocation:  concat_str (buffer_ptr, str_ptr, argc_ptr)
*
*4 Inputs:      buffer_ptr  - command buffer
*               str_ptr     - keyword to be added
*
*5 Outputs:     argc_ptr      - pointer number of arguments in preprocessor
*                               command line buffer
*
*               Returned value:
*                     0 - no error
*                    -1 - error adding to command buffer
*
*6 Caveats:     None.
*
*7 Author:      Christopher J. Knouff               Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int concat_str (buffer_ptr, str_ptr, argc_ptr)
char *buffer_ptr;
char *str_ptr;
int  *argc_ptr;
{
    if ((strlen(buffer_ptr) + strlen(str_ptr)) > (MAX_STR_LEN - 2))
    {
        return(ERROR);
    }
    strcat(buffer_ptr, str_ptr);
    strcat(buffer_ptr, " ");        /* insert a blank at end */
    (*argc_ptr)++;
    return(OK);
}
```

```
--------------------
>>> File: PCHECK.C
--------------------


#include "cc0.h"
#include "tbl_str.h"

#define TRUE    (int)1
#define FALSE   (int)0
/*
 *  Possible parameter errors.
*/
#define NO_ERR   (char)0        /* no error                    */
#define TYPE_ERR (char)1        /* type mismatch               */
#define BYTE_ERR (char)2        /* structure byte count error*/
#define MISS     (char)3        /* missing parameter           */
#define EXTRA    (char)4        /* extra parameter             */
#define PNT_ERR  (char)5        /* pointer mismatch            */
/*
 * Simplify code.
*/
#define DEPARAM (def_entry->dfparams)
#define DPARAM  (def_ptr->dfparams)
#define FPARAM  (fcall_ptr->fcparams)

char *err_tbl[] = {             /* error diagnostics for parameter checking */
    "",
    "TYPE MISMATCH",
    "MISMATCHED BYTE COUNT",
    "MISSING",
    "EXTRA",
    "POINTER MISMATCH"
};

char *type_tbl [] = {           /* variable types */
    "None",
    "Char",
    "U_Char",
    "Short",
    "U_Short",
    "Integer",
    "U_Integer",
    "Ptr",
    "Long",
    "U_Long",
    "Float",
    "Double",
    "Void",
    "Struct",
    "F_Struct",
    "Union",
    "F_Union",
    "Enum",
    "F_Enum"
};
```

```
extern DMAP DefMap[];           /* Definition Map array                      */
extern IMAP InvMap[];           /* Invocation Map array                      */
extern int  P_flag;             /* flag to suppress pointer type mismatch    */
extern int  X_flag;             /* flag for including cross-reference map    */

/****************************** Diebold 1000 ********************************
*
*1 Function:    ret_type_chk                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Determines if the return type of a function assumed at the
*               time of its invocation matches its defined return type.
*
*3 Invocation:  ret_type_chk(def_ptr, fcall_ptr)
*
*4 Inputs:
*               def_ptr   (DEF *)   = pointer to function definition block
*               fcall_ptr (FCALL *) = pointer to function invocation block
*
*5 Outputs:     Function returns:   TRUE  (1) - return type mismatch
*                                   FALSE (0) - no mismatch
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 23 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

int ret_type_chk(def_ptr, fcall_ptr)
DEF   *def_ptr;
FCALL *fcall_ptr;
{
    /*
     *  Compare pointer depth and return type.
    */
    if ((def_ptr->dfret_type != fcall_ptr->fcdecl_type) ||
        (def_ptr->dfp_depth != fcall_ptr->fcp_depth))
    {
        return(TRUE);
    }
    if ((fcall_ptr->fcdecl_type >= T_STRUCT) &&
            (fcall_ptr->fcdecl_type <= T_FUNION))
    {
        /*
         *  Check structure information.
        */
        if (fcall_ptr->fcs_info->s_bytes != def_ptr->dfs_info->s_bytes)
        {
            return(TRUE);
        }
    }
    return(FALSE);      /* return types match */
}

/****************************** Diebold 1000 ********************************
```

```
*
*1 Function:     check_undef                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:      Determines the existence of any undefined functions.
*
*3 Invocation:   check_undef ()
*
*4 Inputs:       None
*
*5 Outputs:      Function returns:   1 - undefined functions exist
*                                    0 - no undefined functions found
*
*6 Caveats:
*
*7 Author:       Christopher J. Knouff          Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

int check_undef ()
{
    int index;              /* used as an index     */

    extern IMAP InvMap[];   /* Invocation Map array                    */
    extern ImapCount;       /* number of entries for Invocation Map */

    /*
     *  Search the Invocation Map until an undefined function is found
     *  or no more entries.
    */

    for (index = 0; index < ImapCount; index++)
    {
        if (InvMap[index].imdef_flag == 0)
        {
            return (TRUE);
        }
    }
    return(FALSE);
}

/****************************** Diebold 1000 ********************************
*
*1 Function:     inv_pcheck                     Copyright (c) Diebold Inc. 1985
*
*2 Summary:      Performs an invocation driven parameter check.
*
*3 Invocation:   inv_pcheck (out_fptr, fname_ptr)
*
*4 Inputs:       out_fptr  (FILE *) = pointer to given output file
*                fname_ptr (char *) = name of source file from which definition
*                                     table was built
*
*5 Outputs:      None.
*
```

```
*6 Caveats:      None.
*
*7 Author:       Christopher J. Knouff          Date: 01 September 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*****************************************************************************/

/*
*   a) Obtain invocation of function whose definition has yet to be found.
*   b) If definition exists in map, compare with the invocation.
*/

void inv_pcheck (out_fptr, fname_ptr)
FILE *out_fptr;
char *fname_ptr;
{
    DEF   *def_ptr;                 /* pointer to a function definition     */
    FCALL *fcall_ptr;               /* pointer to a function invocation     */
    int   def_prt_flg;              /* flags if definition already printed  */
    int   error_flag;               /* signals an error in an invocation    */
    int   def_index;                /* index for the Definition Map array   */
    int   inv_index;                /* index for the Invocation Map array   */
    long  next_offset;              /* offset into file of next invocation  */

    extern DMAP DefMap[];           /* Definition Map array                 */
    extern IMAP InvMap[];           /* Invocation Map array                 */
    extern int  DmapCount;          /* number of entries in Definition Map  */
    extern int  ImapCount;          /* number of entries in Invocation Map  */

    DEF  *get_def();                /* obtains a function definition block  */
    FCALL *get_inv();               /* obtains a function invocation block  */
    int decl_chk  ();               /* checks function declarations         */
    int invoke_chk();               /* checks function calls                */
    void def_prt();                 /* prints the function definition       */
    void free_fnc_def();            /* frees a function definition block    */
    void free_fnc_inv();            /* frees a function invocation block    */
    void loc_prt();                 /* prints data on location of invocation*/

    /*
     * Print name of source file from which definition table was built.
    */
    fprintf(out_fptr,
    "\n************************************************************\n");
    fprintf(out_fptr,"* Source File : %s\n", fname_ptr);
    fprintf(out_fptr,
    "************************************************************\n");

    /*
     *  Obtain undefined function invocations.
    */
    for ( inv_index = 0; inv_index < ImapCount; inv_index++)
    {
        if (InvMap[inv_index].imdef_flag == TRUE)
        {
            continue;               /* invoked function already defined     */
        }
```

```
def_ptr = NULL;                 /* initialize definition block pointer  */
/*
 * Determine if definition exists in new table.
*/
for (def_index = 0; def_index < DmapCount; def_index++)
{
    if (!str_comp
    (DefMap[def_index].dmfunc_name,InvMap[inv_index].imfunc_name))
    {
        def_ptr = get_def (&DefMap[def_index]); /* get definition  */
        InvMap[inv_index].imdef_flag = TRUE;    /* definition found*/
        break;
    }
}
if (def_ptr != NULL)
{
/*
 *  Definition found for this invocation; perform the actual parameter
 *  checking for all invocations of this function.
*/
    def_prt_flg = 0;
    if (X_flag)
    {
        def_prt (out_fptr, def_ptr);            /* print definition */
        def_prt_flg = 1;
        fprintf(out_fptr, "\nInvocations :\n");
    }
    /*
     *  Check all invocations.
    */
    for (fcall_ptr = get_inv(InvMap[inv_index].imstrt_offset);
        fcall_ptr != NULL; fcall_ptr = get_inv(next_offset))
    {
        if (fcall_ptr->fcnum_params == -1)      /* declaration   */
        {
            error_flag = decl_chk (out_fptr, def_ptr, fcall_ptr,
                        &def_prt_flg);
        }
        else                                    /* function call */
        {
            error_flag = invoke_chk(out_fptr, def_ptr, fcall_ptr,
                        &def_prt_flg);
        }
        if (!error_flag && X_flag)
        {
            if (fcall_ptr->fcnum_params == -1)
            {
                fprintf(out_fptr, " Declaration");
            }
            else
            {
                fprintf(out_fptr, " Invocation ");
            }

            loc_prt (out_fptr, fcall_ptr);
        }
        next_offset = fcall_ptr->fcnxt_offset;
```

```
                free_fnc_inv(fcall_ptr);/* free mem held by invoc. block */
            }
            free_fnc_def (def_ptr); /* free mem held by defin. block */
        }
    }
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    print_undef                  Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Prints the names of all undefined functions.
*
*3 Invocation:  print_undef(out_fptr)
*f
*4 Inputs:      out_fptr (FILE *) = pointer to output file
*
*5 Outputs:     None
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*****************************************************************************/

/*
*   a) Search invocation map, printing undefined functions.
*/
void print_undef(out_fptr)
FILE *out_fptr;
{
    int head_prt_flg;           /* flag to print header */
    int index;                  /* used as an index     */

    long    next_offset;        /* pointer into invocation data file    */
    FCALL *fcall_ptr;           /* pointer to a function invocation     */
    extern IMAP InvMap[];       /* Invocation Map array                 */
    extern int  ImapCount;      /* number of entries in Invocation Map  */

    head_prt_flg = 1;
    /*
     *  Search for undefined functions in invocation table.
    */
    for (index = 0; index < ImapCount; index++)
    {
        if (InvMap[index].imdef_flag != TRUE)
        {
            if(head_prt_flg)
            {
                fprintf(out_fptr,"\n* * * * * * * * * * * * * *\n");
                fprintf(out_fptr,"Undefined Functions :\n");
                head_prt_flg = 0;
            }
            fprintf(out_fptr,"\n%s\n", InvMap[index].imfunc_name);
            fflush (out_fptr);
```

```
            if (X_flag)
            {
            /*
            **  Obtain invocation map entries for this function.
            */
                for (fcall_ptr = get_inv(InvMap[index].imstrt_offset);
                     fcall_ptr != NULL; fcall_ptr = get_inv(next_offset))
                {
                    if (fcall_ptr->fcnum_params == -1)
                    {
                        fprintf(out_fptr, "   Declaration");
                    }
                    else
                    {
                        fprintf(out_fptr, "   Invocation ");
                    }

                    loc_prt (out_fptr, fcall_ptr);
                    next_offset = fcall_ptr->fcnxt_offset;

                /*
                **  free memory held by invocation block
                */
                    free_fnc_inv(fcall_ptr);
                }
            }
        }
    }
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    def_prt                        Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Prints a function definition.
*
*3 Invocation:  def_prt (out_fptr, def_entry)
*
*4 Inputs:      out_fptr  (FILE *)  = pointer to output file
*              def_entry (DEF *)   = pointer to definition block
*
*5 Outputs:     None
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*              John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void def_prt (out_fptr, def_entry)
FILE *out_fptr;
DEF *def_entry;
{
    int index;                                  /* used as an index */
```

```
    int p_index;                                    /* used as an index */

    fprintf(out_fptr,
        "************************************************************\n");
    fprintf(out_fptr,"Function Name        : %s\n", def_entry->dfunc_name);
    fprintf(out_fptr,"Declaration Line     : %d\n", def_entry->dfline);
    fprintf(out_fptr,"Return Type          : ");
    for (index = 0; index < def_entry->dfp_depth; index++)
    {
        putc ('*', out_fptr);
    }
    fprintf(out_fptr,"%s", type_tbl[def_entry->dfret_type]);

    /*
     *  If type is structure, print structure information.
    */
    if ((def_entry->dfret_type >= T_STRUCT) &&
    (def_entry->dfret_type <= T_FUNION))
    {
        fprintf(out_fptr," %s",def_entry->dfs_info->s_name);
        fprintf(out_fptr," ,Bytes = %d", def_entry->dfs_info->s_bytes);
    }
    putc ('\n', out_fptr);
    fprintf(out_fptr,"Number of Parameters : %d\n", def_entry->dfnum_params);

    /*
     *  Print data for each parameter.
    */
    for(p_index =0; p_index < def_entry->dfnum_params; p_index++)
    {
        fprintf(out_fptr,"Parameter #%d : %-20s ",p_index + 1, def_entry->
                        dfparams[p_index].p_name);
        putc ('(', out_fptr);
        for (index = 0; index < DEPARAM[p_index].p_depth; index++)
        {
            putc ('*', out_fptr);
        }
        fprintf(out_fptr,"%s", type_tbl[DEPARAM[p_index].p_type]);
        /*
         *  Determine if a structure type; output data if so.
        */
        if (DEPARAM[p_index].p_type >= T_STRUCT &&
             DEPARAM[p_index].p_type <= T_FUNION)
        {
            fprintf(out_fptr," %-15s",DEPARAM[p_index].ps_info->s_name);
            fprintf(out_fptr," ,Bytes = %d",
                        DEPARAM[p_index].ps_info->s_bytes);
        }
        fprintf(out_fptr,")\n");
    }
    if (!X_flag)
    {
        fprintf(out_fptr, "\n");        /* neatness counts */
    }
}

/****************************** Diebold 1000 ******************************
```

```
*
*1 Function:    decl_chk                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Checks a function declaration.
*
*3 Invocation:  decl_chk (out_fptr, def_ptr, fcall_ptr, def_prt_flg)
*
*4 Inputs:      out_fptr    (FILE  *) = pointer to output file
*               def_ptr     (DEF *)   = pointer to definition block
*               fcall_ptr   (FCALL *) = pointer to invocation block
*               def_prt_flg (int   *) = pointer to definition printed flag
*
*5 Outputs:     None
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

/*
*   a) Compare pointer depth and return type.
*   b) If returns a structure, compare byte count.
*/
int decl_chk (out_fptr, def_ptr, fcall_ptr, def_prt_flg)
FILE  *out_fptr;
DEF   *def_ptr;
FCALL *fcall_ptr;
int   *def_prt_flg;
{
    int index;                          /* used as an index */

    void loc_prt();                     /* prints location of an invocation */

    /*
     *  Compare pointer depth and return type.
    */
    if(ret_type_chk(def_ptr, fcall_ptr))
    {
        if (!(*def_prt_flg))
        {
            /*
             *  Print function definition if first error.
            */
            fprintf(out_fptr,
                "\nDefinition of erroneously invoked function :\n");
            def_prt (out_fptr, def_ptr);
            *def_prt_flg = 1;
        }
        /*
         * Print error diagnostics.
        */
        fprintf(out_fptr,"\n*ERROR* : Mismatched Declaration :\n");
        loc_prt(out_fptr, fcall_ptr);           /* location of declaration  */
```

```
        fprintf(out_fptr," Declared return type : ");
        for(index=0; index < fcall_ptr->fcp_depth; index++)
        {
            putc('*', out_fptr);
        }
        fprintf(out_fptr,"%-10s ", type_tbl[fcall_ptr->fcdecl_type]);

        /*
         *  Print structure information.
        */
        if ((fcall_ptr->fcdecl_type >= T_STRUCT) &&
             (fcall_ptr->fcdecl_type <= T_FUNION))
        {
            fprintf(out_fptr,"%-15s, Bytes=%d", fcall_ptr->fcs_info->s_name,
                 fcall_ptr->fcs_info->s_bytes);
        }
        fprintf(out_fptr, "\n\n");
        return (TRUE);      /* error in invocation */
    }
    return (FALSE);         /* no errors detected */
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    invk_err                        Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Prints error diagnostics from an improper function call.
*
*3 Invocation:  invk_err(out_fptr, p_map, fcall_ptr, param_num)
*
*4 Inputs:      out_fptr  (FILE  *) = pointer to output file
*               p_map     (char  *) = pointer to parameter error map
*               fcall_ptr (FCALL *) = pointer to invocation block
*               param_num (int )    = number of parameters
*
*5 Outputs:     None
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff           Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void invk_err(out_fptr, p_map, fcall_ptr, param_num)
FILE  *out_fptr;
char  *p_map;
FCALL *fcall_ptr;
int   param_num;
{
    int index;                  /* used as an index */
    int p_index;                /* used as an index */

    fprintf(out_fptr, " Erroneous invocation :\n");
    /*
```

```
     *  Print data for each parameter, using p_map to determine which
     *  error message to print, if any.
    */
    for (p_index = 0; p_index < param_num; p_index++)
    {
        fprintf(out_fptr," ****Param #%d  ", p_index + 1);
        if (p_map[p_index] == MISS)
        {
            fprintf(out_fptr,"%-25s", "");/* no parameter; pad with blanks */
        }
        else
        {
            /*
             * Print parameter data.
            */
            fprintf(out_fptr,"%-15s ", FPARAM[p_index].p_name);
            for (index = 0; index < FPARAM[p_index].p_depth; index++)
            {
                putc('*', out_fptr);
            }
            fprintf(out_fptr,"%-10s", type_tbl[FPARAM[p_index].p_type]);
            if((FPARAM[p_index].p_type >= T_STRUCT) &&
                (FPARAM[p_index].p_type <= T_FUNION))
            {
                fprintf(out_fptr,"%-15s, Bytes=%d",
                            FPARAM[p_index].ps_info->s_name,
                            FPARAM[p_index].ps_info -> s_bytes);
            }
        }
        /*
         *  Print error message as determined from p_map.
        */
        fprintf(out_fptr," %-22s\n", err_tbl[p_map[p_index]]);
    }
    putc('\n', out_fptr);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    invoke_chk                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Checks the validity of a function call.
*
*3 Invocation:  invoke_chk (out_fptr, def_ptr, fcall_ptr, def_prt_flg)
*
*4 Inputs:      out_fptr     (FILE  *) = pointer to output file
*               def_ptr      (DEF   *) = pointer to definition block
*               fcall_ptr    (FCALL *) = pointer to invocation block
*               def_prt_flg  (int   *) = pointer to definition printed flag
*
*5 Outputs:     None
*
*6 Caveats:
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
```

```
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

/*
*   a) Determine maximum parameters (definition or invocation).
*   b) For each parameter used, check its definition equivalent, storing
*      in param_map the correct error code (0 if no error).
*   c) Print diagnostics if error.
*/

int invoke_chk (out_fptr, def_ptr, fcall_ptr, def_prt_flg)
FILE  *out_fptr;
DEF   *def_ptr;
FCALL *fcall_ptr;
int   *def_prt_flg;
{
    char param_map[NARGS];          /* used as a parameter error map        */
    int  invoke_err;                /* flags an invocation error            */
    int  max_params;                /* maximum number of parameters         */
    int  p_index;                   /* parameter index                      */
    int  ret_type_err;              /* flags return type mismatch           */

    void loc_prt();                 /* prints location of an invocation     */
    invoke_err = FALSE;
    ret_type_err = FALSE;

    /*
     *  Determine maximum parameters.
    */
    if (def_ptr->dfnum_params > fcall_ptr->fcnum_params)
    {
        max_params = def_ptr->dfnum_params;
    }
    else
    {
        max_params = fcall_ptr->fcnum_params;
    }
    /*
     * Determine if return types match.
    */
    if (ret_type_chk(def_ptr, fcall_ptr))
    {
        ret_type_err = TRUE;
    }
    /*
     * Check each parameter.
    */
    for (p_index = 0; (p_index < max_params) && (p_index < NARGS); p_index++)
    {
        param_map[p_index] = NO_ERR;
        if ((p_index + 1) > def_ptr->dfnum_params)
        {
            param_map[p_index] = EXTRA;          /* extra parameter */
        }
        else if ((p_index + 1) > fcall_ptr->fcnum_params)
        {
```

```
            param_map[p_index] = MISS;          /* missing parameter */
        }
        else if (DPARAM[p_index].p_depth != FPARAM[p_index].p_depth)
        {
            param_map[p_index] = TYPE_ERR;      /* pointer depth mismatch */
        }
        /*
         *  Check parameter type if pointer type mismatch not suppressed
         *  or if pointer depth equal to zero.
        */
        else if (!P_flag || FPARAM[p_index].p_depth == 0)
        {
            if  (DPARAM[p_index].p_type != FPARAM[p_index].p_type)
            {
                if (FPARAM[p_index].p_depth > 0)
                    param_map[p_index] = PNT_ERR;   /* pointer mismatch */
                else
                    param_map[p_index] = TYPE_ERR;     /* type mismatch */
            }
            else if ((FPARAM[p_index].p_type >= T_STRUCT) &&
                        (FPARAM[p_index].p_type <= T_FUNION))
            {
                /*
                 *  Check structure information.
                */
                if (FPARAM[p_index].ps_info -> s_bytes !=
                    DPARAM[p_index].ps_info ->s_bytes)
                {
                    if (FPARAM[p_index].p_depth > 0)
                        param_map[p_index] = PNT_ERR; /* pointer mismatch*/
                    else
                        param_map[p_index] = BYTE_ERR; /* byte count error */
                }
            }
        }
        if (param_map[p_index] != NO_ERR)
        {
            invoke_err = TRUE;
        }
    }
    if (invoke_err || ret_type_err)
    {
        if (!(*def_prt_flg))
        {
            fprintf(out_fptr,
                "\nDefinition of erroneously invoked function :\n");
            def_prt (out_fptr, def_ptr);
            *def_prt_flg = 1;
        }
        fprintf(out_fptr, "\n*ERROR* :");
        loc_prt (out_fptr, fcall_ptr);   /* output location of invocation */
    }
    if (ret_type_err)
    {
        fprintf(out_fptr, " Return Type Mismatch : ");
        for (p_index = 0; p_index < fcall_ptr->fcp_depth; p_index++)
        {
```

```
            putc ('*', out_fptr);
        }
        fprintf(out_fptr,"%s", type_tbl[fcall_ptr->fcdecl_type]);

        /*
         *  If type is structure, print structure information.
        */
        if ((fcall_ptr->fcdecl_type >= T_STRUCT) &&
            (fcall_ptr->fcdecl_type <= T_FUNION))
        {
            fprintf(out_fptr," %s",fcall_ptr->fcs_info->s_name);
            fprintf(out_fptr," ,Bytes = %d", fcall_ptr->fcs_info->s_bytes);
        }
        fprintf(out_fptr, "\n\n");
    }
    if (invoke_err)
    {
        invk_err(out_fptr, param_map, fcall_ptr, max_params);
    }
    if (ret_type_err || invoke_err)
    {
        return(TRUE);   /* errors detected    */
    }
    return(FALSE);      /* no errors detected */
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    parm_chk                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     This routine performs a definition driven parameter check.
*
*3 Invocation:  parm_chk (out_fptr)
*
*4 Inputs:      out_fptr (FILE *) = pointer to output file
*
*5 Outputs:     None
*
*6 Caveats:     Any improper invocations cause error diagnostics
*               to be printed.
*
*7 Author:      Christopher J. Knouff          Date: 29 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

/*
*   a) Read a definition from disk at location given by definition map.
*   b) Obtain invocations of current function from invocation file
*      starting at location given by invocation map.
*   c) If an invocation exists, compare invocation to its definition.
*   d) Print data on all definitions and invocations if cross-reference
*      map requested.
*/
```

```
void parm_chk (out_fptr)
FILE *out_fptr;
{
    DEF   *def_ptr;          /* pointer to definition block                */
    FCALL *fcall_ptr;        /* pointer to an invocation block             */
    int   def_index;         /* index for the Definition Map array         */
    int   def_prt_flg;       /* flags if definition already printed        */
    int   error_flag;        /* signals an error in current invocation     */
    int   inv_index;         /* index for the Invocation Map array         */
    int   invoke_flag;       /* signals current function has been invoked  */
    long  next_offset;       /* offset in file of next block               */

    extern char main_fname[];    /* filename buffer                        */
    extern DMAP DefMap[];        /* Definition Map array                   */
    extern IMAP InvMap[];        /* Invocation Map array                   */
    extern int  DmapCount;       /* number of entries in Definition Map    */
    extern int  ImapCount;       /* number of entries in Invocation Map    */

    char *strncpy();             /* standard string copy                   */
    DEF  *get_def();             /* obtains a function definition block    */
    FCALL *get_inv();            /* obtains a function invocation block    */
    int decl_chk  ();            /* checks function declarations           */
    int invoke_chk();            /* checks function calls                  */
    void  free_fnc_def();        /* frees a function definition block      */
    void  free_fnc_inv();        /* frees a function invocation block      */
    void  loc_prt();             /* prints location of invocation          */

    main_fname[0] = '\0';                    /* initialize filename buffer */
    /*
     *  Check each entry in the definition map.
    */
    for (def_index = 0; def_index < DmapCount; def_index++)
    {
        def_ptr = get_def (&DefMap[def_index]);     /* get definition */
        /*
         *  Print source filename if different from old one.
        */
        if (str_comp(main_fname, def_ptr->dfsrc_fname))
        {
            fprintf(out_fptr,
          "\n*************************************************************\n");
            fprintf(out_fptr,"* Source File : %s\n", def_ptr->dfsrc_fname);
            fprintf(out_fptr,
            "*************************************************************\n");
            /*
             *  Place new filename in buffer.
            */
            strncpy(main_fname, def_ptr->dfsrc_fname, NFNAME - 1);
            main_fname[NFNAME - 1] = '\0';
        }
        def_prt_flg = 0;
        invoke_flag = 0;
        if (X_flag)
        {
            def_prt(out_fptr, def_ptr);
            def_prt_flg = 1;
        }
```

```
/*
 *  Obtain invocation map entry for this function.
*/
fcall_ptr = NULL;
for (inv_index = 0; inv_index < ImapCount; inv_index++)
{
    if(!str_comp(InvMap[inv_index].imfunc_name,
         DefMap[def_index].dmfunc_name))
    {
        fcall_ptr = get_inv (InvMap[inv_index].imstrt_offset);
        break;
    }
}
if (fcall_ptr != NULL)
{
    InvMap[inv_index].imdef_flag = 1; /* function has been defined */
    if (X_flag)
    {
        fprintf(out_fptr, "\nInvocations :\n");
    }
}
while (fcall_ptr != NULL)
{
    if (fcall_ptr->fcnum_params == -1)
    {
        /*
         *  Check the function declaration.
        */
        error_flag = decl_chk(out_fptr, def_ptr, fcall_ptr,
                              &def_prt_flg);
    }
    else
    {
        invoke_flag = 1;    /* function has been called */
        /*
         *  Check function call.
        */
        error_flag = invoke_chk(out_fptr, def_ptr, fcall_ptr,
                                &def_prt_flg);
    }
    /*
     *  Print location of invocation if no errors(since location will
     *  have already been printed if so) and if cross reference flag
     *  has been set.
    */
    if (!error_flag && X_flag)
    {
        if (fcall_ptr->fcnum_params == -1)
        {
            fprintf(out_fptr, " Declaration");
        }
        else
        {
            fprintf(out_fptr, " Invocation ");
        }

        loc_prt (out_fptr, fcall_ptr);
```

```
            }
            /*
             *  Free memory allocated by current invocation block and
             *  obtain next block.
            */
            next_offset = fcall_ptr->fcnxt_offset;
            free_fnc_inv (fcall_ptr);
            fcall_ptr = get_inv (next_offset);
        }
        if (invoke_flag == 0)
        {
            /*
             *  If function is not main, print warning message that
             *  function is defined but not invoked.
            */
            if (str_comp(def_ptr->dfunc_name, "main"))
            {
                fprintf(out_fptr,
                    "\n**WARNING : function \"%s\" not called\n",
                        def_ptr->dfunc_name);
                fflush (out_fptr);
            }
        }
        free_fnc_def (def_ptr);     /* free memory held by definition block */
    }
}

void loc_prt (out_fptr, fcall_ptr)
FILE *out_fptr;
FCALL *fcall_ptr;
{
    fprintf(out_fptr, " In %-20s at LINE #%d ",
            fcall_ptr->fcsrc_fname, fcall_ptr->fcline);
    if (*fcall_ptr->fcsrc_fnc != '\0')
    {
        fprintf(out_fptr, "from %-20s\n",fcall_ptr->fcsrc_fnc);
    }
    else
    {
        fprintf(out_fptr, ": %-20s\n","<external declaration>");
    }
}


----------------------
>>> File: DSK_HNDLR.C
----------------------


#include "cc0.h"
#include "tbl_str.h"
#include "def.h"

#define DEF_FILE    ":work:def_file"
#define INV_FILE    ":work:inv_file"
#define DPARAM  (def_ptr->dfparams)
#define FPARAM  (fcall_ptr->fcparams)
```

```
extern char *BufStrtPtr;        /* pointer to parameter expression buffer   */
extern FILE *DefFilePtr;        /* Definition File pointer                  */
extern FILE *InvFilePtr;        /* Invocation File pointer                  */

char *mem_alloc();              /* allocates memory                         */
char *strcpy();                 /* standard string copy                     */
void free();                    /* frees allocated memory                   */

/****************************** Diebold 1000 ********************************
*
*1 Function:    put_string                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Outputs a string with a newline to the output file
*
*3 Invocation:  put_string (string, fp, file_name)
*
*4 Inputs:      string  - string to output
*               fp      - FILE connection
*               filename- only used in case of error
*
*5 Outputs:     Characters go out file connection
*
*6 Caveats:     Exits if error occurs on output.
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void put_string (string, fp, file_name)
char *string;
FILE *fp;
char *file_name;
{
    int *fputs();               /* writes a string to output file   */

    fputs (string, fp);
    if (ferror (fp))
    {
        printf("\nput_string\n");
        printf ("\nERROR : Output error file \"%s\"\n", file_name);
        exit(1);
    }
    putc ((int)'\n', fp);
    if (ferror (fp))
    {
        printf("\nputc\n");
        printf ("\nERROR : Output error file \"%s\"\n", file_name);
        exit(1);
    }
}

/****************************** Diebold 1000 ********************************
*
```

```
*1 Function:    get_string                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Gets a string from the input file
*
*3 Invocation:  get_string (string, fp, file_name)
*
*4 Inputs:      fp       - FILE connection
*               filename- only used in case of error
*
*5 Outputs:     string   - inputed string
*
*6 Caveats:     Exits if error occurs on input
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

void get_string (string, fp, file_name)
char *string;
FILE *fp;
char *file_name;
{
    if ((fgets (string, NCSYMB - 1, fp) == NULL) || ferror (fp))
    {
        printf("\nget_string\n");
        printf ("\nERROR : Input error in file \"%s\"\n", file_name);
        exit(1);
    }
    if (strlen(string) != 0)
        string [strlen(string) - 1] = '\0';
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    put_word                      Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Outputs a word (2- bytes) to the output file
*
*3 Invocation:  put_word (i_word, fp, file_name)
*
*4 Inputs:      i_word   - word to output
*               fp       - FILE connection
*               filename- only used in case of error
*
*5 Outputs:     Characters go out file connection
*
*6 Caveats:     Exits if error occurs on output
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/
```

```
void put_word (i_word, fp, file_name)
int i_word;
FILE *fp;
char *file_name;
{
    if ((putw (i_word, fp) != i_word) || ferror (fp))
    {
        printf("\nput_word\n");
        printf ("\nERROR : Output error in file \"%s\"\n", file_name);
        exit(1);
    }
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    get_word                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Gets a word (2- bytes) from the input file
*
*3 Invocation:  get_string (fp, file_name)
*
*4 Inputs:      fp      - FILE connection
*               filename- only used in case of error
*
*5 Outputs:     Function Returns:  i_word   - word from input
*
*6 Caveats:     Exits if error occurs on input
*
*7 Author:      Christopher J. Knouff              Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
******************************************************************************/
int get_word (fp, file_name)
FILE *fp;
char *file_name;
{
    int i_word;

    i_word = getw(fp);
    if (ferror(fp))
    {
        printf("\nget_word\n");
        printf("\nERROR : Input error in file \"%s\"\n", file_name);
        exit(1);
    }
    return (i_word);
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    put_dblword                 Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Outputs a double word (4 - bytes) to the output file
*
```

```
*3 Invocation:  put_dblword (dbl_word, fp, file_name)
*
*4 Inputs:      dbl_word    - double word to output
*               fp      - FILE connection
*               filename- only used in case of error
*
*5 Outputs:     Characters go out file connection
*
*6 Caveats:     Exits if error occurs on output
*
*7 Author:      Christopher J. Knouff                Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void put_dblword (dbl_word, fp, file_name)
long dbl_word;
FILE *fp;
char *file_name;
{
    if ((fwrite((char *)&dbl_word, sizeof(long), 1, fp) != 1) || ferror(fp))
    {
        printf("put_dblword\n");
        printf("\nERROR : Output error in file \"%s\"\n", file_name);
        exit(1);
    }
}

/****************************** Diebold 1000 ********************************
*
*1 Function:    get_dblword                  Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Gets a double word (4 - bytes) from th einput file
*
*3 Invocation:  get_dblword (fp, file_name)
*
*4 Inputs:      fp      - FILE connection
*               filename- only used in case of error
*
*5 Outputs:     Function returns:  dbl_word - double word to output
*
*6 Caveats:     Exits if error occurs on input
*
*7 Author:      Christopher J. Knouff                Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

long get_dblword (fp, file_name)
FILE *fp;
char *file_name;
{
    long dbl_word;
```

```
    if ((fread((char *)&dbl_word, sizeof(long), 1, fp) != 1) || ferror(fp))
    {
        printf("\nget_dblword\n");
        printf("\nERROR : Input error in file \"%s\"\n", file_name);
        exit(1);
    }
    return (dbl_word);
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    mem_alloc                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Allocates specified number of bytes of memory.
*
*3 Invocation:  mem_alloc (num_bytes)
*
*4 Inputs:      num_bytes (int ) = number of bytes requested
*
*5 Outputs:     Function returns:   Pointer to block of memory allocated.
*
*6 Caveats:     Exits from program if unable to allocate memory.
*
*7 Author:      Christopher J. Knouff          Date: 09 June 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

char *mem_alloc (num_bytes)
int num_bytes;
{
    char *block_ptr;                /* pointer to block of memory allocated */
    char *malloc();                 /* standard memory allocation function  */

    if((block_ptr = malloc((unsigned)num_bytes)) == NULL)
    {
        printf("\nERROR : unable to allocate memory\n");
        fflush(stdout);
        exit(1);
    }
    return(block_ptr);
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    insert_param                 Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Inserts parameter data into temporary parameter block.
*
*3 Invocation:  insert_param (tree_ptr)
*
*4 Inputs:      tree_ptr (TREE *) = pointer to the tree created in the parser
*                                   that contains data for this parameter.
*
```

```
*5 Outputs:      None
*
*6 Caveats:
*
*7 Author:       Christopher J. Knouff          Date: 09 June 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

void insert_param (tree_ptr)
TREE *tree_ptr;
{
    TPARAM *tmp_ptr;                /* pointer to temporary parameter block */

    extern TPARAM *TopTparamPtr;    /* pointer to chain of TPARAM blocks    */
    TPARAM *get_prmtr();            /* obtains a temporary parameter block  */

    /*
     * If first parameter, insert at top of chain.
    */
    if ((tmp_ptr = TopTparamPtr) == NULL)
    {
        TopTparamPtr = get_prmtr (tree_ptr);
    }
    else
    {
        /*
         *  Determine end of existing chain and insert new parameter.
        */
        while (tmp_ptr -> tp_next != NULL)
        {
            tmp_ptr = tmp_ptr -> tp_next;
        }
        tmp_ptr -> tp_next = get_prmtr (tree_ptr);
    }
}

/****************************** Diebold 1000 ********************************
*
*1 Function:     get_prmtr                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:      Obtains and initializes a temporary parameter info block.
*
*3 Invocation:   get_prmtr(tree_ptr)
*
*4 Inputs:       tree_ptr (TREE *) = pointer to tree containing parameter data
*
*5 Outputs:      Function returns:   Pointer to new parameter info block
*
*6 Caveats:
*
*7 Author:       Christopher J. Knouff          Date: 09 June 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
```

```
*
*****************************************************************************/

/*
*   a) Allocate needed memory.
*   b) Enter necessary data.
*/

TPARAM *get_prmtr(tree_ptr)
TREE *tree_ptr;
{
    DIM    *tmp_dp;             /* temporary pointer to a DIM structure     */
    TPARAM *param_ptr;          /* pointer to a temporary parameter block   */

    char *mem_alloc();              /* allocates memory                     */
    /*
     *  Allocate needed memory.
    */
    param_ptr = (TPARAM *)mem_alloc( sizeof(struct tparam_str));
    param_ptr->tp_id = mem_alloc( strlen(BufStrtPtr) + 1);
    strcpy( param_ptr -> tp_id, BufStrtPtr);    /* parameter expression */
    param_ptr -> tp_type = tree_ptr -> t_type;  /* expression type      */
    if ((tree_ptr->t_type >= T_STRUCT) && (tree_ptr->t_type <= T_FUNION))
    {
        /*
         *  t_ip will contain needed structure information.
        */
        param_ptr->tp_info = tree_ptr->t_ip;
    }
    param_ptr -> tp_pdepth = 0;
    for (tmp_dp = tree_ptr -> t_dp; tmp_dp != NULL; tmp_dp = tmp_dp -> d_dp)
    {
        /*
         * if d_type not 'function' then a pointer is represented, one for
         * each pointer level.
        */
        if (tmp_dp->d_type != D_FUNC)
        {
            (param_ptr -> tp_pdepth)++;
        }
    }
    param_ptr -> tp_next = NULL;

    return (param_ptr);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    free_param                    Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Frees a chain of temporary parameter blocks.
*
*3 Invocation:  free_param (param_ptr)
*
*4 Inputs:      param_ptr (TPARAM *) = pointer to block to be freed.
*
*5 Outputs:     None
```

```
*
*6 Caveats:      1) This is a recursive function.
*
*7 Author:       Christopher J. Knouff          Date: 09 June 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*                Christopher J. Knouff          Date: 02 September 1985
*                Function now frees memory allocated by tp_id, which is
*                the parameter expression. This is because the expression,
*                along with other parameter data, is written to the disk.
*
*******************************************************************************/

void free_param (param_ptr)
TPARAM *param_ptr;
{
    if (param_ptr != NULL)
    {
        /*
         * Free next parameter first (since current one points to it).
        */
        free_param (param_ptr -> tp_next);
        free (param_ptr->tp_id);    /* free memory for parameter expression */
        free ((char *) param_ptr);
    }
}

/******************************** Diebold 1000 *********************************
*
*1 Function:     decl_fwrite                  Copyright (c) Diebold Inc. 1985
*
*2 Summary:      Writes declaration data to the disk.
*
*3 Invocation:   decl_fwrite (sym_ptr, source_fnc)
*
*4 Inputs:       sym_ptr     (SYM *)   = pointer to symbol structure for the
*                                        function declaration.
*                source_fnc  (char *)  = pointer to source function of
*                                        declaration
*
*5 Outputs:      None
*
*6 Caveats:
*
*7 Author:       Christopher J. Knouff          Date: 28 August 1985
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void decl_fwrite (sym_ptr, source_fnc)
SYM *sym_ptr;
char *source_fnc;
{
    DIM *dim_ptr;                       /* pointer to a dimension stucture  */
```

```
    int p_depth;                            /* pointer depth                    */
    long strt_pos;                          /* starting position in file        */

    void map_inv();                         /* updates the Invocation Map       */

    p_depth = 0;
    strt_pos = ftell(InvFilePtr);           /* obtain starting position in file */
    put_word ( -1, InvFilePtr, INV_FILE);       /* signifies a declaration */
    put_string (file, InvFilePtr, INV_FILE);            /* source file      */
    put_string (source_fnc, InvFilePtr, INV_FILE);      /* source function  */
    put_word (sym_ptr->s_type, InvFilePtr, INV_FILE);   /* return type      */
    /*
     *  Determine pointer depth.
    */
    for (dim_ptr = sym_ptr -> s_dp; dim_ptr != NULL;
        dim_ptr = dim_ptr -> d_dp)
    {
        if (dim_ptr -> d_type != D_FUNC)
        {
            ++p_depth;
        }
    }
    put_word (p_depth, InvFilePtr, INV_FILE);
    put_word (sym_ptr->s_dline, InvFilePtr, INV_FILE);
    if ((sym_ptr->s_type >= T_STRUCT) && (sym_ptr->s_type <= T_FUNION))
    {
        if (sym_ptr->s_ip == NULL)
        {
            printf("ERROR : %s, no INFO struct for \"%s\"\n",
                 file, sym_ptr->s_id);
            exit(1);
        }
        /*
         * Write structure info (tag name and # bytes) to disk.
        */
        if (sym_ptr->s_ip->i_id != NULL)
        {
            put_string (sym_ptr->s_ip->i_id, InvFilePtr, INV_FILE);
        }
        else
        {
            put_string ("", InvFilePtr, INV_FILE);
        }
        put_word (sym_ptr->s_ip->i_data, InvFilePtr, INV_FILE);
    }
    map_inv (strt_pos, sym_ptr->s_id, ftell(InvFilePtr));
    put_dblword ((long)-1, InvFilePtr, INV_FILE);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    fcall_fwrite                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Writes function call data to the disk.
*
*3 Invocation:  fcall_fwrite (line_num, tree_ptr)
*
```

```
*4 Inputs:      line_num    (int)      = line number
*               tree_ptr    (TREE *)   = pointer to tree of invoked function
*
*5 Outputs:     None
*
*6 Caveats:     Exits if a parameter is of a structure type but no data
*               exists for that structure.
*
*7 Author:      Christopher J. Knouff          Date: 29 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void fcall_fwrite ( line_num, tree_ptr)
int   line_num;
TREE  *tree_ptr;
{
    DIM *dim_ptr;       /* pointer to a dimension structure                  */
    int p_depth;        /* pointer depth of given item                       */
    long strt_pos;      /* starting position in file for this block          */
    TPARAM *t_tparam;   /* pointer to temorary parameter blocks              */

    extern char *SourceFnc; /* pointer to the source of the function call    */
    extern int NumArgs;     /* number of arguments used in invocation        */
    extern TPARAM *TopTparamPtr;    /* pointer to top of linked list of      */
                                    /* parameter data blocks                 */
    void map_inv();                 /* updates invocation map                */

    p_depth = 0;
    strt_pos = ftell(InvFilePtr);       /* starting position for this block */
    put_word ( NumArgs, InvFilePtr, INV_FILE);      /* number of arguments  */
    put_string (file, InvFilePtr, INV_FILE);        /* source file          */
    put_string (SourceFnc, InvFilePtr, INV_FILE);   /* source function      */
    put_word (tree_ptr->t_type, InvFilePtr, INV_FILE);      /* return type  */
    /*
     *  Determine pointer depth.
    */
    for (dim_ptr = tree_ptr->t_dp; dim_ptr != NULL;
        dim_ptr = dim_ptr->d_dp)
    {
        if (dim_ptr->d_type != D_FUNC)
        {
            ++p_depth;
        }
    }
    put_word (p_depth, InvFilePtr, INV_FILE);   /* pointer depth        */
    put_word (line_num, InvFilePtr, INV_FILE);  /* line # of invocation */
    /*
     *  Write structure information to disk (if structure type) as provided
     *  by the tree's INFO structure.
    */
    if ((tree_ptr->t_type >= T_STRUCT) && (tree_ptr->t_type <= T_FUNION))
    {
        if (tree_ptr->t_ip->i_id != NULL)                   /* tag name */
        {
```

```
            put_string (tree_ptr->t_ip->i_id, InvFilePtr, INV_FILE);
        }
        else
        {
            put_string ("", InvFilePtr, INV_FILE);
        }
        put_word (tree_ptr->t_ip->i_data, InvFilePtr, INV_FILE);/* # bytes  */
    }
    if (NumArgs > 0)
    {
        /*
         * Write data from temporary parameter blocks to the disk.
        */
        for (t_tparam = TopTparamPtr; t_tparam != NULL; t_tparam =
            t_tparam -> tp_next)
        {
            put_string (t_tparam -> tp_id, InvFilePtr, INV_FILE);
            put_word (t_tparam -> tp_type, InvFilePtr, INV_FILE);
            put_word (t_tparam -> tp_pdepth, InvFilePtr, INV_FILE);
            /*
             *  Write structure info to disk if of structure type.
            */
            if ((t_tparam->tp_type >= T_STRUCT) &&
            (t_tparam->tp_type <= T_FUNION))
            {
                if (t_tparam->tp_info == NULL)
                {
                    printf("ERROR : %s, no INFO struct for \"%s\"\n",
                         file, t_tparam->tp_id);
                    exit(1);
                }
                if (t_tparam->tp_info->i_id != NULL)            /* tag name */
                {
                    put_string (t_tparam->tp_info->i_id, InvFilePtr,
                            INV_FILE);
                }
                else
                {
                    put_string ("", InvFilePtr, INV_FILE);
                }
                /* write # bytes for this structure */
                put_word (t_tparam->tp_info->i_data, InvFilePtr, INV_FILE);
            }
        }
    }
    /*
     *  Update invocation map and write offset of zero to disk.
    */
    map_inv (strt_pos, tree_ptr->t_sp->s_id, ftell(InvFilePtr));
    put_dblword ((long)-1, InvFilePtr, INV_FILE);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    def_fwrite              Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Writes function defintion data to the disk.
```

```
*
*3 Invocation:  def_fwrite (sp)
*
*4 Inputs:      sp (SYM *) = symbol structure
*
*5 Outputs:     Output to definition storage area
*
*6 Caveats:     None
*
*7 Author:      Christopher J. Knouff          Date: 29 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void def_fwrite (sp)
SYM *sp;
{
    int index;                  /* used as an index                          */
    int p_depth;                /* pointer depth of current object           */
    DIM *t_dp;                  /* temporary pointer to a DIM struct         */

    extern int nargs;           /* number of arguments for this function     */
    extern SYM *args[];         /* array containing pointers to symbol       */
                                /* structures for each argument              */

    p_depth = 0;
    /*
     *  Write definition data to the disk.
    */
    put_word ( nargs, DefFilePtr, DEF_FILE);    /* number of parameters      */
    put_string (file, DefFilePtr, DEF_FILE);    /* source file of definition*/

    /*
     *  Determine pointer depth.
    */
    for (t_dp = sp->s_dp; t_dp != NULL; t_dp = t_dp->d_dp)
    {
        /*
         * Anything other than D_FUNC represents a pointer.
        */
        if (t_dp->d_type != D_FUNC)
        {
            p_depth++;
        }
    }
    put_word (sp->s_dline - 1, DefFilePtr, DEF_FILE);
    put_word (p_depth, DefFilePtr, DEF_FILE);
    put_word (sp->s_type, DefFilePtr, DEF_FILE);
    /*
     *  Determine if return type is of a structure type.
    */
    if ((sp->s_type >= T_STRUCT) && (sp->s_type <= T_FUNION))
    {
        if (sp->s_ip == NULL)
        {
```

```
            printf("ERROR : %s, no INFO struct for \"%s\"\n", file, sp->s_id);
            exit(1);
        }
        /*
         * Write structure info (tag name and # bytes) to disk.
        */
        if (sp->s_ip->i_id != NULL)
        {
            put_string (sp->s_ip->i_id, DefFilePtr, DEF_FILE);
        }
        else
        {
            put_string ("", DefFilePtr, DEF_FILE);
        }
        put_word (sp->s_ip->i_data, DefFilePtr, DEF_FILE);
    }
    /*
     *  Write necessary parameter information.
    */
    for (index = 0; index < nargs; index++)
    {
        p_depth = 0;
        put_string (args[index]->s_id, DefFilePtr, DEF_FILE);
        put_word (args[index]->s_type, DefFilePtr, DEF_FILE);
        for (t_dp = args[index]->s_dp; t_dp != NULL; t_dp = t_dp->d_dp)
        {
            if (t_dp->d_type != D_FUNC)
            {
                p_depth++;
            }
        }
        put_word (p_depth, DefFilePtr, DEF_FILE);
        if ((args[index]->s_type >= T_STRUCT) &&
        (args[index]->s_type <= T_FUNION))
        {
            /*
             * Write structure info (tag name and # bytes) to disk.
            */
            if (args[index]->s_ip->i_id != NULL)
            {
                put_string (args[index]->s_ip->i_id, DefFilePtr, DEF_FILE);
            }
            else
            {
                put_string ("", DefFilePtr, DEF_FILE);
            }
            put_word (args[index]->s_ip->i_data, DefFilePtr, DEF_FILE);
        }
    }
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    map_def                          Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Puts function definition in memory index map
*
```

```
*3 Invocation:  map_def (func_name, offset)
*
*4 Inputs:      func_name (char *) = function name
*               offset (long)      = offset in Definition disk work file
*
*5 Outputs:     Put entry in memory index table
*
*6 Caveats:     Exits if a function is already present in the index table
*               or the table is full.
*
*7 Author:      Christopher J. Knouff          Date: 29 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

void map_def (func_name, offset)
char *func_name;
long offset;
{
    int index;                  /* used as an index */

    extern DMAP DefMap[];       /* Definition Map array         */
    extern int  DmapCount;      /* # entries in Definition Map  */

    if (DmapCount == DEF_MAP_SIZE)
    {
        printf("ERROR : Maximum definition entries of %c exceeded\n",
           DEF_MAP_SIZE);
        exit(ERROR);
    }
    /*
     * Determine if definition for this function already exists.
    */
    for (index = 0; index < DmapCount; index++)
    {
        if (!str_comp(func_name, DefMap[index].dmfunc_name))
        {
            printf("\nERROR in %s: function \"%s\" being redefined\n",
                file, func_name);
            exit(ABORT);
        }
    }
    strncpy (DefMap[DmapCount].dmfunc_name, func_name, FNAME_SIZE - 1);
    DefMap[DmapCount].dmfunc_name[FNAME_SIZE - 1] = '\0';
    DefMap[DmapCount].dmoffset = offset;
    DmapCount++;
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    map_inv                        Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Puts/Updates function invocation entry in memory index map
*               and updates the invocation "linked list" on disk.
*
```

```
*3 Invocation:  map_def (start, func_name, last)
*
*4 Inputs:      start (long)       = start of function entry in disk file
*               func_name (char *) = function name
*               offset (long)      = offset in Definition disk work file
*
*5 Outputs:     Put/update entry in memory index table
*
*6 Caveats:     Exits if the index table is full.

*7 Author:      Christopher J. Knouff          Date: 29 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

void map_inv (start, func_name, last)
long start;
char *func_name;
long last;
{
    int index;                  /* used as an index */

    extern IMAP InvMap[];       /* Invocation Map                    */
    extern int ImapCount;       /* # of entries in Invocation Map    */

/*
*   Determine if function exists in the map.
*/
    for (index = 0; index < ImapCount; index++)
    {
        if (!str_comp (func_name, InvMap[index].imfunc_name))
        {
            break;
        }
    }
/*
*   If function not in map, initialize new map element if maximum
*   number of entries not exceeded.
*/
    if (index == ImapCount)
    {
        if (ImapCount == INV_MAP_SIZE)
        {
            printf("ERROR : Maximum invocation entries of %d exceeded.\n",
                   INV_MAP_SIZE);
            exit(ERROR);
        }
        strncpy(InvMap[index].imfunc_name, func_name, FNAME_SIZE);
        InvMap[index].imfunc_name[FNAME_SIZE - 1] = '\0';
        InvMap[index].imstrt_offset = start;
        InvMap[index].imlast_offset = last;
        InvMap[index].imdef_flag = 0;
        ImapCount++;
    }
    else
```

```
    {
    /*
     *  Seek to position in file of last invocation's pointer to next
     *  invocation(last_invocation->next_inv) and write the offset of
     *  new invocation block; then seek back to last position in file.
    */
        fseek(InvFilePtr, InvMap[index].imlast_offset, 0);
        put_dblword (start, InvFilePtr, INV_FILE);
        fseek(InvFilePtr, last, 0);
        InvMap[index].imlast_offset = last;
    }
}

/******************************** Diebold 1000 ********************************
*
*1 Function:    get_inv                        Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Reads an invocation entry from the invocation storage file
*
*3 Invocation:  get_inv (offset)
*
*4 Inputs:      offset (long) = start of function entry in disk file
*
*5 Outputs:     Function allocs and returns a structure with the data
*               Returns a NULL if no memory available.
*
*6 Caveats:     None
*
*7 Author:      Christopher J. Knouff          Date: 29 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

FCALL *get_inv (offset)
long offset;
{
    char buffer[NCSYMB];        /* character buffer                          */
    FCALL *fcall_ptr;           /* pointer to an invocation structure        */
    int  index;                 /* used as an index                          */
    int  num_params;            /* number of parameters for this function    */

    char *mem_alloc();              /* allocates memory                      */
    /*
     *  Return NULL if offset == 0 ; represents no more invocations for
     *  current function.
    */
    if (offset == -1)
    {
        return ((FCALL *) NULL);
    }
    /*
     *  Obtain file position of invocation block.
    */
    fseek (InvFilePtr, offset, 0);
    /*
```

```
 *  Obtain number of parameters and allocate required memory for an
 *  invocation structure.
*/
num_params = get_word(InvFilePtr, INV_FILE);
if (num_params > 0)
    fcall_ptr = mem_alloc (sizeof(struct fcall_str) +
                           (num_params * sizeof(struct param_str)));
else
    fcall_ptr = mem_alloc (sizeof(struct fcall_str));
fcall_ptr->fcnum_params = num_params;
/*
 *  Obtain source filename and place in block.
*/
get_string (buffer, InvFilePtr, INV_FILE);
fcall_ptr->fcsrc_fname = mem_alloc (strlen(buffer) + 1);
strcpy(fcall_ptr->fcsrc_fname, buffer);
/*
 *  Obtain source function and place in block.
*/
get_string (buffer, InvFilePtr, INV_FILE);
fcall_ptr->fcsrc_fnc = mem_alloc(strlen(buffer) + 1);
strcpy(fcall_ptr->fcsrc_fnc, buffer);
/*
 *  Obtain return type, pointer depth, and line number.
*/
fcall_ptr->fcdecl_type = get_word (InvFilePtr, INV_FILE);
fcall_ptr->fcp_depth = get_word (InvFilePtr, INV_FILE);
fcall_ptr->fcline = get_word (InvFilePtr, INV_FILE);
/*
 *  If of a structure type, allocate a structure info block and read in
 *  required data (tag name and byte count).
*/
if ((fcall_ptr->fcdecl_type >= T_STRUCT) &&
    (fcall_ptr->fcdecl_type <= T_FUNION))
{
    get_string (buffer, InvFilePtr, INV_FILE);  /* read in tag name */
    if (buffer[0] == '\0')
    {
        strcpy(buffer, "<untagged structure>");
    }
    fcall_ptr->fcs_info =
            mem_alloc (sizeof(struct stctin_str) + strlen(buffer) + 1);
    strcpy(fcall_ptr->fcs_info->s_name, buffer);
    fcall_ptr->fcs_info->s_bytes = get_word (InvFilePtr, INV_FILE);
}
else
{
    fcall_ptr->fcs_info = NULL;
}
/*
 *  If there are parameters for this function, read in and store
 *  parameter data.
*/
for (index = 0; index < num_params; index++)
{
    get_string (buffer, InvFilePtr, INV_FILE);
    FPARAM[index].p_name = mem_alloc(strlen(buffer) + 1);
```

```
        strcpy(FPARAM[index].p_name, buffer);
        FPARAM[index].p_type = get_word(InvFilePtr, INV_FILE);
        FPARAM[index].p_depth = get_word(InvFilePtr, INV_FILE);
        /*
         *  Determine if of structure type and enter data accordingly.
        */
        if((FPARAM[index].p_type >= T_STRUCT) &&
            (FPARAM[index].p_type <= T_FUNION))
        {
            get_string (buffer, InvFilePtr, INV_FILE);  /* struct tag name  */
            /*
             *  Allocate memory for a struct info block.
            */
            if (buffer[0] == '\0')
            {
                strcpy(buffer, "<untagged structure>");
            }
            FPARAM[index].ps_info = mem_alloc (sizeof(struct stctin_str) +
                                    strlen(buffer) + 1);
            /*
             *  Enter struct info into memory block.
            */
            strcpy (FPARAM[index].ps_info->s_name, buffer);
            FPARAM[index].ps_info->s_bytes = get_word (InvFilePtr, INV_FILE);
        }
        else    /* not of structure type */
        {
            FPARAM[index].ps_info = NULL;
        }
    }
    /*
     *  Read in offset to next invocation block in file.
    */
    fcall_ptr->fcnxt_offset = get_dblword (InvFilePtr, INV_FILE);
    return (fcall_ptr);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    get_def                        Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Reads a definition entry from the definition storage file
*
*3 Invocation:  get_def (dmap_ptr)
*
*4 Inputs:      dmap_ptr = pointer to entry in memory index table
*
*5 Outputs:     Function allocs and returns a structure with the data
*               Returns a NULL if no memory available.
*
*6 Caveats:     None
*
*7 Author:      Christopher J. Knouff          Date: 29 August 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
```

```
*****************************************************************************/

DEF *get_def (dmap_ptr)
DMAP *dmap_ptr;
{

    char buffer[NCSYMB];        /* character buffer                          */
    DEF  *def_ptr;              /* pointer to a definition structure         */
    int  index;                 /* used as an index                          */
    int  num_params;            /* number of parameters for this function    */

    char *mem_alloc();              /* allocates memory                      */
    /*
     *  Obtain file position of invocation block.
    */
    fseek (DefFilePtr, dmap_ptr->dmoffset, 0);
    /*
     *  Obtain number of parameters and allocate required memory for an
     *  invocation structure.
    */
    num_params = get_word(DefFilePtr, DEF_FILE);
    def_ptr = mem_alloc (sizeof(struct def_str) +
                         (num_params * sizeof(struct param_str)));
    def_ptr->dfnum_params = num_params;
    def_ptr->dfunc_name = dmap_ptr->dmfunc_name;
    /*
     *  Obtain source filename and place in block.
    */
    get_string (buffer, DefFilePtr, DEF_FILE);
    def_ptr->dfsrc_fname = mem_alloc (strlen(buffer) + 1);
    strcpy(def_ptr->dfsrc_fname, buffer);
    /*
     *  Obtain return type, pointer depth, and line number.
    */
    def_ptr->dfline = get_word (DefFilePtr, DEF_FILE);
    def_ptr->dfp_depth = get_word (DefFilePtr, DEF_FILE);
    def_ptr->dfret_type = get_word (DefFilePtr, DEF_FILE);
    /*
     *  If of a structure type, allocate a structure info block and read in
     *  required data (tag name and byte count).
    */
    if ((def_ptr->dfret_type >= T_STRUCT) && (def_ptr->dfret_type <= T_FUNION)
    {
        get_string (buffer, DefFilePtr, DEF_FILE);
        if (buffer[0] == '\0')
        {
            strcpy(buffer, "<untagged structure>");
        }
        def_ptr->dfs_info =
                 mem_alloc (sizeof(struct stctin_str) + strlen(buffer) + 1);
        strcpy(def_ptr->dfs_info->s_name, buffer);
        def_ptr->dfs_info->s_bytes = get_word (DefFilePtr, DEF_FILE);
    }
    else
    {
        def_ptr->dfs_info = NULL;
    }
```

```
    /*
     *  If there are parameters for this function, read in and store
     *  parameter data.
    */
    for (index = 0; index < num_params; index++)
    {
        get_string (buffer, DefFilePtr, DEF_FILE);
        DPARAM[index].p_name = mem_alloc(strlen(buffer) + 1);
        strcpy(DPARAM[index].p_name, buffer);
        DPARAM[index].p_type = get_word(DefFilePtr, DEF_FILE);
        DPARAM[index].p_depth = get_word(DefFilePtr, DEF_FILE);
        /*
         *  Determine if of structure type and enter data accordingly.
        */
        if((DPARAM[index].p_type >= T_STRUCT) &&
            (DPARAM[index].p_type <= T_FUNION))
        {
            get_string (buffer, DefFilePtr, DEF_FILE);  /* struct tag name  */
            /*
             *  Allocate memory for a struct info block.
            */
            if (buffer[0] == '\0')
            {
                strcpy(buffer, "<untagged structure>");
            }
            DPARAM[index].ps_info = mem_alloc (sizeof(struct stctin_str) +
                                    strlen(buffer) + 1);
            /*
             *  Enter struct info into memory block.
            */
            strcpy (DPARAM[index].ps_info->s_name, buffer);
            DPARAM[index].ps_info->s_bytes = get_word (DefFilePtr, DEF_FILE);
        }
        else    /* not of structure type */
        {
            DPARAM[index].ps_info = NULL;
        }
    }
    return (def_ptr);
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    free_fnc_def                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Frees memory allocated by function definition block.
*
*3 Invocation:  free_fnc_def (def_ptr)
*
*4 Inputs:      def_ptr (DEF *) = pointer to a function definition block
*
*5 Outputs:     None
*
*6 Caveats:     This function might have to be modified if the structure of
*               a definition block is changed.
*
*7 Author:      Christopher J. Knouff          Date: 01 September 1985
```

```
*                John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*****************************************************************************/

/*
*   a) Free memory allocated by definition block's structure data block.
*   b) Free memory allocated by that function's parameter data.
*   c) Free memory allocated by definition block itself.
*/

void free_fnc_def (def_ptr)
DEF *def_ptr;
{
    int index;                  /* used as an index */

    if (def_ptr != NULL)
    {
        free (def_ptr->dfsrc_fname);        /* free source file name    */
        /*
         *  Free structure information block.
        */
        if (def_ptr->dfs_info != NULL)
        {
            free ((char *)def_ptr->dfs_info);
        }
        /*
         *  Free data for each parameter.
        */
        for (index = 0; index < def_ptr->dfnum_params; index++)
        {
            /*
             *  Free parameter's structure information block.
            */
            if (def_ptr->dfparams[index].ps_info != NULL)
            {
                free ((char *) def_ptr->dfparams[index].ps_info);
            }
            free (def_ptr->dfparams[index].p_name);
        }
        /*
         *  Free main definition block.
        */
        free ((char *)def_ptr);
    }
}

/******************************** Diebold 1000 *********************************
*
*1 Function:    free_fnc_inv                Copyright (c) Diebold Inc. 1985
*
*2 Summary:     Frees memory allocated by function invocation block.
*
*3 Invocation:  free_fnc_inv (inv_ptr)
*
```

```
*4 Inputs:      inv_ptr (FCALL *) = pointer to a function invocation block
*
*5 Outputs:     None
*
*6 Caveats:     This function might have to be modified if the structure of
*               an invocation block is changed.
*
*7 Author:      Christopher J. Knouff          Date: 01 September 1985
*               John A. Sexton
*
*8 Revisions: (Name, Date, Description)
*
*******************************************************************************/

/*
*   a) Free memory allocated by invocation block's structure data block.
*   b) Free memory allocated by that function's parameter data.
*   c) Free memory allocated by invocation block itself.
*/

void free_fnc_inv (inv_ptr)
FCALL *inv_ptr;
{
    int index;                  /* used as an index */

    if (inv_ptr != NULL)
    {
        free (inv_ptr->fcsrc_fname);            /* free source file name    */
        free (inv_ptr->fcsrc_fnc);              /* free source function name*/
        /*
         *  Free structure information block.
        */
        if (inv_ptr->fcs_info != NULL)
        {
            free ((char *)inv_ptr->fcs_info);
        }
        /*
         *  Free data for each parameter.
        */
        for (index = 0; index < inv_ptr->fcnum_params; index++)
        {
            /*
             *  Free parameter's structure information block.
            */
            if (inv_ptr->fcparams[index].ps_info != NULL)
            {
                free ((char *) inv_ptr->fcparams[index].ps_info);
            }
            free (inv_ptr->fcparams[index].p_name);
        }
        /*
         *  Free main invocation block.
        */
        free ((char *)inv_ptr);
    }
}
```

```
--------------------
>>> File: tbl_str.h
--------------------

/*****************************************************************************
**                                                                          **
**  LISTING #2  -   'C' Include files for the Parameter Checker             **
**                                                                          **
*****************************************************************************/

/*  Table and Map Structures Used by Parameter Checker */

#define FNAME_SIZE 30                   /* max characters in a filename */

#ifdef PC_LARGE
#define DEF_MAP_SIZE (int)350           /* max entries in definition map*/
#define INV_MAP_SIZE (int)450           /* max entries in invocation map*/
#else
#define DEF_MAP_SIZE (int)325           /* max entries in definition map*/
#define INV_MAP_SIZE (int)375           /* max entries in invocation map*/
#endif

typedef struct tparam_str
{                                       /* temporary parameter structure*/
    char *tp_id;                        /* id name                      */
    int  tp_type;                       /* expression type              */
    int  tp_pdepth;                     /* pointer depth                */
    INFO *tp_info;                      /* INFO pointer, if aggregate   */
    struct tparam_str *tp_next;         /* pointer to next parameter    */
} TPARAM;

typedef struct stctin_str
{                                       /* structure data                   */
    int  s_bytes;                       /* number of bytes for this struct  */
    char s_name[];                      /* structure tag name               */
} STCTIN;

typedef struct param_str
{                                       /* structure for parameter data     */
    char    *p_name;                    /* name of parameter                */
    int     p_type;                     /* parameter type                   */
    int     p_depth;                    /* pointer depth                    */
    STCTIN  *ps_info;                   /* pointer to struct data           */
} PARAM;

typedef struct def_str
{                                       /* function definition structure    */
    int     dfnum_params;               /* number of parameters             */
    char    *dfsrc_fname;               /* source file name                 */
    char    *dfunc_name;                /* name of function                 */
    int     dfline;                     /* line number of definition        */
    int     dfp_depth;                  /* pointer depth                    */
    int     dfret_type;                 /* return type                      */
    STCTIN  *dfs_infc;                  /* structure information pointer    */
```

```
    PARAM   dfparams[];             /* parameters for this function     */
} DEF;

typedef struct fcall_str
{                                   /* structure for a function call    */
    int     fcnum_params;           /* number of parameters in call     */
    char    *fcsrc_fname;           /* source of invocation             */
    char    *fcsrc_fnc;             /* the calling function             */
    int     fcdecl_type;            /* declaration type (-1 if a call)  */
    int     fcp_depth;              /* pointer depth for return type    */
    int     fcline;                 /* line number of invocation        */
    STCTIN  *fcs_info;              /* pointer to structure info block  */
    long    fcnxt_offset;           /* offset in file of next invocation*/
    PARAM   fcparams[];             /* parameters of this call          */
} FCALL;

typedef struct dmap_str
{                                   /* definition map entry             */
    long dmoffset;                  /* offset into definition file      */
    char dmfunc_name[FNAME_SIZE];   /* name of function                 */
} DMAP;

typedef struct imap_str
{                                   /* invocation map entry             */
    char imdef_flag;                /* flags function is defined        */
    long imstrt_offset;             /* offset to first block in invoca- */
                                    /* tion file for this function      */
    long imlast_offset;             /* offset to last block in invoca-  */
                                    /* tion file for this function      */
    char imfunc_name[FNAME_SIZE];   /* name of function                 */
} IMAP;


-----------------
>>> File: def.h
-----------------

#define ERROR           (int)-1
#ifndef OK
#define OK              (int)0
#endif
#define MAX_STR_LEN     (int)256
#define F_NEXIST        (int)0x21
#define FLAG_SIZE       (int)3
#define TRUE            (int)1
#define FALSE           (int)0
#define NULL_CHAR       '\0'
#define NULL_PTR        (char *)0
#define KEYWORD         (int)-1
#define DFLT_PREP       (int)0
#define TO_PREP         (int)1
#define OVER_PREP       (int)2
#define AFTER_PREP      (int)3

struct str_tab_tag{
    char num;
    char string[MAX_STR_LEN];
```

A.7 Make File Listing

```
----------------------
>>> File: param.make
----------------------

#/*************************************************************************
#**                                                                      *
#**   LISTING #3  - MAKE file for the Parameter Checker                  *
#**                                                                      *
#*************************************************************************

#
# This is the make file for Parameter Checker with DefMap= 325 InvMap = 375
#
# NOTE :  Alternate lqmain is used due to the parsing of
#         the command line.
#
:pe:param_chk= :po:param_chk.obj :po:com_parse.obj :po:cc0.obj :po:cpp.obj \
               :po:pcheck.obj :po:dsk_hndlr.obj :po:DIAG.obj :po:NPUT.obj  \
               :po:DOPE.obj :po:CC0SYM.obj :po:EXPR.obj :po:BIND.obj       \
               :po:DDECL.obj :po:BPUT.obj :po:STAT.obj :po:DPUT.obj        \
               :po:GDECL.obj :po:LEX.obj :po:FOLD.obj :po:ETC.obj          \
               :po:INIT.obj :po:IPUT.obj :po:LOCALS.obj :po:LPUT.obj       \
               :po:GCANDT.obj :po:NEWLAB.obj :po:NEWSEG.obj :po:SIZE.obj   \
               :po:SPUT.obj :po:lexit.obj :po:ffile.obj :po:fgetc.obj      \
               :po:fsetup.obj
    delete :pm:param_chk.map
    link86 &
    :po:param_chk.obj, :po:com_parse.obj,       &
    :po:pcheck.obj,    :po:dsk_hndlr.obj,       &
        .---\
        .    >  Insert other object files for pre-processing and parsing
        .---/
    :po:altmain.obj, &
    :sd:lib/cc86.10/chi.lib,    :sd:lib/cc86.10/ceios.lib, &
    :sd:lib/cc86.10/cbios.lib,  :sd:lib/cc86.10/cnuc.lib,  &
    :sd:lib/cc86.10/cldr.lib,   :sd:lib/cc86.10/lclib.lib, &
    :sd:lib/rmx86/hpifl.lib,    :sd:lib/rmx86/ipifl.lib,   &
    :sd:lib/rmx86/epifl.lib,    :sd:lib/rmx86/large.lib,   &
    :sd:lib/rmx86/lpifl.lib,    :sd:lib/rmx86/rpifl.lib,   &
    :sd:lib/ndp87/87null.lib                               &
    to :pe:param_chk bind segsize (stack(20000))           &
    mempool(20000, 0FFFFFh) map print(:pm:param_chk.map)

:po:param_chk.obj=  :ps:param_chk.c :pi:ctype.h :pi:udi.h \
                    :pi:cc0.h :pi:def.h :pi:tbl_str.h
    delete :po:param_chk.obj
    cc86 :ps:param_chk.c to :po:param_chk.obj large ROM include(:pi:)

:po:com_parse.obj=  :ps:com_parse.c :pi:ctype.h :pi:stdio.h :pi:def.h
    delete :po:com_parse.obj
    cc86 :ps:com_parse.c to :po:com_parse.obj large ROM include(:pi:)

:po:pcheck.obj=     :ps:pcheck.c :pi:cc0.h :pi:tbl_str.h
    delete :po:pcheck.obj
```

```
    cc86 :ps:pcheck.c to :po:pcheck.obj large ROM include(:pi:)

:po:dsk_hndlr.obj=  :ps:dsk_hndlr.c :pi:cc0.h :pi:def.h :pi:tbl_str.h
    delete :po:dsk_hndlr.obj
    cc86 :ps:dsk_hndlr.c to :po:dsk_hndlr.obj large ROM include(:pi:)
```

```
/*************************************************************************
**                                                                      **
**  Appendix B - Listing # 1                                            **
**                                                                      **
**  This is the console output of the parameter checker as it           **
**  processes the modules for Release 'D' - Executable #6               **
**                                                                      **
**  The WARNINGs listed here are caught as each module is parsed.       **
**  These warnings are due to errors that do not need to be tracked     **
**  between modules (i.e. the value to to be passed back by the return  **
**  statement does not match the definition of the function that is     **
**  being parsed).  The "missing return values" are caused by           **
**  functions that are NOT declared "void" and have no value in their   **
**  return statement.                                                   **
**                                                                      **
*************************************************************************/

/*
**  This is the invocation line that was used.
*/

 param_chk @dcspin.pc over ^data/dcspin.out include (:bi:) c r


/*
**  This is the output to the CRT.
*/

 PARAMETER CHECKER   <Version 2.12  DefTbl=360  InvTbl=450>

        The following options will be included :
                -include 'C' library functions
                -include RMX functions

        Output will be written to ^data/dcspin.out


        Processing :bs:CHSTUB.CL0

        Processing :bs:DCATIL.CL0

**WARNING : :bs:DCATIL.CL0 : REVERSE, Line# 97 - missing return value.

**WARNING : :bs:DCATIL.CL0 : ITOA, Line# 146 - missing return value.

**WARNING : :bs:DCATIL.CL0 : LTOA, Line# 196 - missing return value.

**WARNING : :bs:DCATIL.CL0 : FREEBUFR, Line# 311 - missing return value.

**WARNING : :bs:DCATIL.CL0 : FREEBUFR, Line# 317 - missing return value.

**WARNING : :bs:DCATIL.CL0 : FREEBUFR, Line# 340 - missing return value.

**WARNING : :bs:DCATIL.CL0 : RETMREC, Line# 850 - missing return value.

**WARNING : :bs:DCATIL.CL0 : RETMREC, Line# 861 - missing return value.

**WARNING : :bs:DCATIL.CL0 : RETMREC, Line# 873 - missing return value.
```

```
**WARNING : :bs:DCATIL.CL0 : RETMREC, Line# 876 - missing return value.

        Processing :bs:DCEDES.CL3

        Processing :bs:DCSBIO.CL4

**WARNING : :bs:DCSBIO.CL4 : INPKEY, Line# 74
        - return value used is not return value of defined function
Defined return value : Char
Return value used    : Integer

        Processing :bs:DCSERR.CL3

**WARNING : :bs:DCSERR.CL3 : ALRM_STAT_CHEK, Line# 296 - missing return value.

        Processing :bs:DCSFMT.CL3

        Processing :ps:DCSPIN.CL4

**WARNING : :ps:DCSPIN.CL4 : PIN_GEN, Line# 169 - missing return value.

        Processing :bs:DCSPPM.CL4

        Processing :bs:DCSREQ.CL0

        Processing :bs:DCSRPT.CL4

        Processing :bs:dcsrq2.CL3

        Processing :bs:DCSTXT.CL3

        Processing :bs:DCSUMC.CL3

        Processing :bs:DCSUTL.CL4

**WARNING : :bs:DCSUTL.CL4 : itolp, Line# 64 - missing return value.

**WARNING : :bs:DCSUTL.CL4 : itolp, Line# 80 - missing return value.

**WARNING : :bs:DCSUTL.CL4 : lptoi, Line# 125 - missing return value.

**WARNING : :bs:DCSUTL.CL4 : lptoi, Line# 133 - missing return value.

**WARNING : :bs:DCSUTL.CL4 : htoi, Line# 182 - missing return value.

        Processing :sd:param_chk/clib.def

        Processing :sd:param_chk/rmx.def

        PARAMETER CHECKING COMPLETE
```

```
/*************************************************************************
**                                                                      **
**  Appendix B - Listing # 2                                            **
**                                                                      **
**  This is the output of the parameter checker as it does the actual   **
**  parameter checking on the Release 'D' - Executable #6 source code.  **
**                                                                      **
**  This listing will give you an overall idea of the kinds of errors   **
**  the parameter checker can catch.  This listing has been edited      **
**  due to its size.  Only a subset of the actual errors are shown      **
**  here.                                                               **
**                                                                      **
*************************************************************************/


**********************************************************
* Source File : :bs:CHSTUB.CL0
**********************************************************

Definition of erroneously invoked function :
**********************************************************
Function Name         : CHK_STAT
Declaration Line      : 1
Return Type           : Integer
Number of Parameters : 1
Parameter #1 : STATUS                (Integer)


*ERROR* : In :bs:DCATIL.CL0        at LINE #308 from FREEBUFR
 Erroneous invocation :
 ****Param #1  DEL_STAT         U_Integer  TYPE MISMATCH


**********************************************************
* Source File : :bs:DCATIL.CL0
**********************************************************

Definition of erroneously invoked function :
**********************************************************
Function Name         : ITOA
Declaration Line      : 125
Return Type           : Integer
Number of Parameters : 2
Parameter #1 : STRNG                 (*Char)
Parameter #2 : NUM                   (Integer)


*ERROR* : Mismatched Declaration :
 In :ps:DCSPIN.CL4        at LINE #84 from PIN
 Declared return type : *Char


*ERROR* : In :ps:DCSPIN.CL4        at LINE #91 from PIN
 Return Type Mismatch : *Char

 Erroneous invocation :
 ****Param #1  PIN_NO           *Char
```

```
 ****Param #2  X                U_Integer  TYPE MISMATCH


**WARNING : function "LTOA" not called

Definition of erroneously invoked function :
************************************************************
Function Name         : FREEBUFR
Declaration Line      : 288
Return Type           : Integer
Number of Parameters : 2
Parameter #1 : ADDRESS              (*U_Char)
Parameter #2 : STATUS               (*U_Integer)


*ERROR* : In :bs:DCEDES.CL3        at LINE #555 from fmt_rmptdesc
 Erroneous invocation :
 ****Param #1  rmpt_record     *Struct    rmpt_str Bytes=249 POINTER MISMATCH
 ****Param #2  &status         *Integer    POINTER MISMATCH


*ERROR* : In :ps:DCSPIN.CL4        at LINE #324 from main
 Erroneous invocation :
 ****Param #1  RET_REC_PTR[0]  *Char       POINTER MISMATCH
 ****Param #2  &status         *U_Integer


*ERROR* : In :bs:DCSRPT.CL4        at LINE #807 from FIL_REPORT
 Erroneous invocation :
 ****Param #1  RPT_PTR[0]      *Char       POINTER MISMATCH
 ****Param #2  &STATUS         *Integer    POINTER MISMATCH


**WARNING : function "TD_TOSEC" not called

Definition of erroneously invoked function :
************************************************************
Function Name         : ALOC_BLK
Declaration Line      : 551
Return Type           : *Char
Number of Parameters : 2
Parameter #1 : SIZE                 (U_Integer)
Parameter #2 : STATUS_PTR           (*Integer)


*ERROR* : In :bs:DCSFMT.CL3        at LINE #384 from FMT_SCHED
 Erroneous invocation :
 ****Param #1  80*10           Integer    TYPE MISMATCH
 ****Param #2  &STATUS         *Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : STR_SIZE
Declaration Line      : 615
Return Type           : Integer
Number of Parameters : 3
```

```
Parameter #1 : STR_PTR                (*Char)
Parameter #2 : MAX_LEN                (Integer)
Parameter #3 : STAT_PTR               (*U_Integer)


*ERROR* : In :bs:DCATIL.CL0       at LINE #404 from TD_TOSEC
 Erroneous invocation :
 ****Param #1  ATD_PTR          *Char
 ****Param #2  (int)10          Integer
 ****Param #3  &STATUS          *Integer    POINTER MISMATCH


**WARNING : function "INCR_PTR" not called

**WARNING : function "GETMREC" not called

**WARNING : function "RETMREC" not called

************************************************************
* Source File : :bs:DCEDES.CL3
************************************************************

**WARNING : function "fmt_rmptdesc" not called

************************************************************
* Source File : :bs:DCSBIO.CL4
************************************************************

Definition of erroneously invoked function :
************************************************************
Function Name         : OUTCHAR
Declaration Line      : 102
Return Type           : Integer
Number of Parameters : 1
Parameter #1 : C                      (Char)


*ERROR* : In :bs:DCSBIO.CL4       at LINE #368 from READ_INPUT
 Erroneous invocation :
 ****Param #1  CH               Integer    TYPE MISMATCH


************************************************************
* Source File : :bs:DCSERR.CL3
************************************************************

Definition of erroneously invoked function :
************************************************************
Function Name         : RET_STAT_CHEK
Declaration Line      : 387
Return Type           : Integer
Number of Parameters : 3
Parameter #1 : SCHED_STATUS           (Integer)
Parameter #2 : HDR_LPOL               (Integer)
Parameter #3 : ERR_STR                (*Char)
```

```
*ERROR* : In :bs:DCSREQ.CL0        at LINE #181 from DB_REQ
 Erroneous invocation :
 ****Param #1  SCHED_PTR->ACT_MSG.SSTATUS Integer
 ****Param #2  SCHED_PTR->ACT_HDR.LE_NUM U_Integer  TYPE MISMATCH
 ****Param #3  RET_STAT_STR     *Char


************************************************************
* Source File : :bs:DCSFMT.CL3
************************************************************

 efinition of erroneously invoked function :
************************************************************
Function Name         : FMT_TCT
Declaration Line      : 83
Return Type           : Void
Number of Parameters : 2
Parameter #1 : RETFIL                (*Struct RETURN_FILE_STR ,Bytes = 6)
Parameter #2 : FMT_STRNG             (*Char)


*ERROR* : In :bs:DCSRPT.CL4        at LINE #680 from FIL_REPORT
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : FMT_BADG
Declaration Line      : 156
Return Type           : Void
Number of Parameters : 2
Parameter #1 : RETFIL                (*Struct RETURN_FILE_STR ,Bytes = 6)
Parameter #2 : FMT_STRNG             (*Char)


*ERROR* : In :bs:DCSRPT.CL4        at LINE #685 from FIL_REPORT
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : FMT_LOG
Declaration Line      : 217
Return Type           : Void
Number of Parameters : 2
Parameter #1 : RETFIL                (*Struct RETURN_FILE_STR ,Bytes = 6)
Parameter #2 : FMT_STRNG             (*Char)


*ERROR* : In :bs:DCSRPT.CL4        at LINE #690 from FIL_REPORT
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : FMT_CAL
Declaration Line      : 305
```

```
Return Type           : Void
Number of Parameters : 2
Parameter #1 : RETFIL                (*Struct RETURN_FILE_STR ,Bytes = 6)
Parameter #2 : FMT_STRNG             (*Char)


*ERROR* : In :bs:DCSRPT.CL4       at LINE #695 from FIL_REPORT
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : FMT_ALARM
Declaration Line      : 453
Return Type           : Void
Number of Parameters : 2
Parameter #1 : RETFIL                (*Struct RETURN_FILE_STR ,Bytes = 6)
Parameter #2 : FMT_STRNG             (*Char)


*ERROR* : In :bs:DCSRPT.CL4       at LINE #711 from FIL_REPORT
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : FMT_PSWD
Declaration Line      : 564
Return Type           : Void
Number of Parameters : 2
Parameter #1 : RETFIL                (*Struct RETURN_FILE_STR ,Bytes = 6)
Parameter #2 : FMT_STRNG             (*Char)


*ERROR* : In :bs:DCSRPT.CL4       at LINE #716 from FIL_REPORT
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : FMT_PIN
Declaration Line      : 627
Return Type           : Void
Number of Parameters : 3
Parameter #1 : RETFIL                (*Struct RETURN_FILE_STR ,Bytes = 6)
Parameter #2 : FMT_STRNG             (*Char)
Parameter #3 : ACSYS_PTR             (*Struct ACSYS_STR       ,Bytes = 60)


*ERROR* : In :bs:DCSRPT.CL4       at LINE #721 from FIL_REPORT
 Return Type Mismatch : Integer

 Erroneous invocation :
 ****Param #1  &RETURN_FILE    *Struct    RETURN_FILE_STR, Bytes=6
 ****Param #2  TMP_STR         *Char
 ****Param #3  SVSF_PTR[0]     *Char       POINTER MISMATCH
```

```
************************************************************
* Source File : :ps:DCSPIN.CL4
************************************************************

Definition of erroneously invoked function :
************************************************************
Function Name         : PIN
Declaration Line      : 74
Return Type           : Void
Number of Parameters : 5
Parameter #1 : COD1                    (U_Integer)
Parameter #2 : COD2                    (U_Integer)
Parameter #3 : COD3                    (U_Integer)
Parameter #4 : CARD                    (Long)
Parameter #5 : PIN_NO                  (*Char)


*ERROR* : In :bs:DCSFMT.CL3       at LINE #664 from FMT_PIN
 Return Type Mismatch : Integer

 Erroneous invocation :
 ****Param #1  PCALC1          U_Integer
 ****Param #2  PCALC2          U_Integer
 ****Param #3  PCALC3          U_Integer
 ****Param #4  CARD_NUM        U_Long     TYPE MISMATCH
 ****Param #5  PIN_NUM         *Char


************************************************************
* Source File : :bs:DCSPPM.CL4
************************************************************

************************************************************
* Source File : :bs:DCSREQ.CL0
************************************************************

Definition of erroneously invoked function :
************************************************************
Function Name         : DB_REQ
Declaration Line      : 129
Return Type           : Integer
Number of Parameters : 6
Parameter #1 : DBNAM                   (Integer)
Parameter #2 : DBFUNCT                 (Integer)
Parameter #3 : DBKEY                   (*Char)
Parameter #4 : KEYTYP                  (Integer)
Parameter #5 : DBDEST                  (Integer)
Parameter #6 : REC_FIL_PTR             (**Char)


*ERROR* : In :bs:DCSREQ.CL0       at LINE #253 from COAL_CHEK
 Erroneous invocation :
 ****Param #1  8               Integer
 ****Param #2  0               Integer
 ****Param #3  "\\\\"          *Char
 ****Param #4  0               Integer
```

```
 ****Param #5  2                  Integer
 ****Param #6  &ACSYS_PTR         **Struct  ACSYS_STR Bytes=60 POINTER MISMATCH

**WARNING : function "PRT_CNTRL" not called

************************************************************
* Source File : :bs:DCSRPT.CL4
************************************************************

************************************************************
* Source File : :bs:dcsrq2.CL3
************************************************************

**WARNING : function "db_com1" not called

Definition of erroneously invoked function :
************************************************************
Function Name         : db_req2
Declaration Line      : 175
Return Type           : Integer
Number of Parameters : 6
Parameter #1 : dbnam                  (Integer)
Parameter #2 : dbfunct                (Integer)
Parameter #3 : dbkey                  (*Char)
Parameter #4 : keytyp                 (Integer)
Parameter #5 : dbdest                 (Integer)
Parameter #6 : rec_fil_ptr            (**Char)


*ERROR* : In :bs:DCEDES.CL3       at LINE #545 from fmt_rmptdesc
 Erroneous invocation :
 ****Param #1  10              Integer
 ****Param #2  0               Integer
 ****Param #3  &line_poll      *Integer    POINTER MISMATCH
 ****Param #4  12              Integer
 ****Param #5  2               Integer
 ****Param #6  &rmpt_record    **Struct    rmpt_str Bytes=249 POINTER MISMATCH


************************************************************
* Source File : :bs:DCSUMC.CL3
************************************************************

************************************************************
* Source File : :bs:DCSUTL.CL4
************************************************************

Definition of erroneously invoked function :
************************************************************
Function Name         : itolp
Declaration Line      : 44
Return Type           : Integer
Number of Parameters : 2
Parameter #1 : integer_var          (Integer)
Parameter #2 : lp_string            (*Char)
```

```
*ERROR* : Mismatched Declaration :
 In :bs:DCSFMT.CL3       at LINE #52 : <external declaration>
 Declared return type : Void


*ERROR* : In :bs:DCSFMT.CL3       at LINE #100 from FMT_TCT
 Return Type Mismatch : Void


**WARNING : function "lptoi" not called

**WARNING : function "check_mil_time" not called

************************************************************
* Source File : :sd:param_chk/clib.def
************************************************************

Definition of erroneously invoked function :
************************************************************
Function Name         : strlen
Declaration Line      : 804
Return Type           : U_Integer
Number of Parameters : 1
Parameter #1 : source                (*Char)


*ERROR* : In :bs:DCATIL.CL0       at LINE #91 from REVERSE
 Return Type Mismatch : Integer


*ERROR* : In :bs:DCSBIO.CL4       at LINE #147 from OUTC
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : bld_ptr
Declaration Line      : 250
Return Type           : *Char
Number of Parameters : 1
Parameter #1 : sel                   (U_Integer)


*ERROR* : Mismatched Declaration :
 In :bs:DCATIL.CL0       at LINE #238 from GETBUFR
 Declared return type : *U_Char


*ERROR* : In :bs:DCATIL.CL0       at LINE #319 from FREEBUFR
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : bld_sel
Declaration Line      : 256
```

```
Return Type          : Integer
Number of Parameters : 1
Parameter #1 : ptr                    (*Char)


*ERROR* : Mismatched Declaration :
 In :bs:DCATIL.CL0       at LINE #294 from FREEBUFR
 Declared return type : U_Integer


*ERROR* : In :bs:DCATIL.CL0       at LINE #302 from FREEBUFR
 Return Type Mismatch : U_Integer

 Erroneous invocation :
 ****Param #1  ADDRESS          *U_Char     POINTER MISMATCH


*ERROR* : In :bs:DCSREQ.CL0       at LINE #66 from DB_COM
 Erroneous invocation :
 ****Param #1  SCHED_PTR        *Struct    SCHED_STR Bytes=21 POINTER MISMATCH


*ERROR* : Mismatched Declaration :
 In :bi:useful_dcl.h     at LINE #16 : <external declaration>
 Declared return type : U_Integer


Definition of erroneously invoked function :
************************************************************
Function Name        : strcpy
Declaration Line     : 788
Return Type          : *Char
Number of Parameters : 2
Parameter #1 : dest                   (*Char)
Parameter #2 : source                 (*Char)


*ERROR* : Mismatched Declaration :
 In :bs:DCEDES.CL3       at LINE #83 from fmtdes
 Declared return type : Void


*ERROR* : In :bs:DCEDES.CL3       at LINE #95 from fmtdes
 Return Type Mismatch : Void


*ERROR* : In :bs:DCSERR.CL3       at LINE #702 from RET_STAT_CHEK
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name        : rmbl
Declaration Line     : 688
Return Type          : *Char
Number of Parameters : 1
Parameter #1 : in_p                   (*Char)
```

```
*ERROR* : In :bs:DCEDES.CL3        at LINE #554 from fmt_rmptdesc
 Return Type Mismatch : Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : strncpy
Declaration Line      : 860
Return Type           : *Char
Number of Parameters  : 3
Parameter #1 : dest                    (*Char)
Parameter #2 : source                  (*Char)
Parameter #3 : len                     (U_Integer)


*ERROR* : In :bs:DCSUTL.CL4        at LINE #372 from pad_mtime_fields
 Erroneous invocation :
 ****Param #1  second           *Char
 ****Param #2  &time_string[start_pos] *Char
 ****Param #3  field_length    Integer    TYPE MISMATCH


Definition of erroneously invoked function :
************************************************************
Function Name         : isdigits
Declaration Line      : 441
Return Type           : U_Integer
Number of Parameters  : 1
Parameter #1 : source                  (*Char)


*ERROR* : Mismatched Declaration :
 In :bi:useful_dcl.h      at LINE #44 : <external declaration>
 Declared return type : Integer


*ERROR* : In :bs:DCSUTL.CL4        at LINE #439 from check_mil_time
 Return Type Mismatch : Integer


************************************************************
* Source File : :sd:param_chk/rmx.def
************************************************************

Definition of erroneously invoked function :
************************************************************
Function Name         : rqsleep
Declaration Line      : 752
Return Type           : Void
Number of Parameters  : 2
Parameter #1 : timelimit               (U_Integer)
Parameter #2 : exceptptr               (*U_Integer)


*ERROR* : In :bs:DCATIL.CL0        at LINE #566 from ALOC_BLK
```

```
 Return Type Mismatch : Integer

 Erroneous invocation :
 ****Param #1  10                Integer     TYPE MISMATCH
 ****Param #2  &ALOC_STAT        *U_Integer


Definition of erroneously invoked function :
************************************************************
Function Name         : rqreceivecontrol
Declaration Line      : 675
Return Type           : Void
Number of Parameters : 2
Parameter #1 : regiont                (U_Integer)
Parameter #2 : exceptptr              (*U_Integer)


*ERROR* : In :bs:DCATIL.CL0       at LINE #779 from GETMREC
 Return Type Mismatch : Integer

 Erroneous invocation :
 ****Param #1  FILREG_PTR->REGION_TOK U_Integer
 ****Param #2  STATUS            *Integer     POINTER MISMATCH


Definition of erroneously invoked function :
************************************************************
Function Name         : rqsendcontrol
Declaration Line      : 699
Return Type           : Void
Number of Parameters : 1
Parameter #1 : exceptptr              (*U_Integer)


*ERROR* : In :bs:DCATIL.CL0       at LINE #869 from RETMREC
 Return Type Mismatch : Integer

 Erroneous invocation :
 ****Param #1  STATUS            *Integer     POINTER MISMATCH


Definition of erroneously invoked function :
************************************************************
Function Name         : rqgetexceptionhandler
Declaration Line      : 632
Return Type           : Void
Number of Parameters : 2
Parameter #1 : exceptioninfoptr       (*Struct eh_str           ,Bytes = 5)
Parameter #2 : exceptptr              (*U_Integer)


*ERROR* : In :bs:DCEDES.CL3       at LINE #415 from fmtdes
 Erroneous invocation :
 ****Param #1  &exception_ptr.EXC_HNDLR **Char        TYPE MISMATCH
 ****Param #2  &status           *Integer     POINTER MISMATCH
```

```
*ERROR* : In :bs:DCSERR.CL3        at LINE #786 from RET_STAT_CHEK
 Return Type Mismatch : Integer

 Erroneous invocation :
 ****Param #1  &EXCEPT_INFO     *Struct    INFO            , Bytes=5
 ****Param #2  &EXCEPT_STAT     *Integer    POINTER MISMATCH


Definition of erroneously invoked function :
************************************************************
Function Name         : rqsetexceptionhandler
Declaration Line      : 713
Return Type           : Void
Number of Parameters : 2
Parameter #1 : exceptioninfoptr     (*Struct eh_str          ,Bytes = 5)
Parameter #2 : exceptptr            (*U_Integer)


*ERROR* : In :bs:DCEDES.CL3        at LINE #418 from fmtdes
 Erroneous invocation :
 ****Param #1  &exception_ptr.EXC_HNDLR **Char       TYPE MISMATCH
 ****Param #2  &status          *Integer    POINTER MISMATCH



* * * * * * * * * * * * * *
Undefined Functions :
"sprint"
"sprintf"
```

```
#ifndef VIO_HDW
#define VIO_HDW
/*
*************************************************************************
** $Source: /users2/prism/dss/revm/include/RCS/vio_hdw.h,v $
** $Revision: 2.3 $
** $Date: 89/07/21 15:19:00 $
** $Author: mandel $
** $Locker:  $
*************************************************************************
**/
#include "dssdefs.h"
#include "icm_cppdef.h"

/* Video IO poweron constants */
#define VIO_R0          CPP_HEX(0000)
#define VIO_R1          CPP_HEX(0011)
#define VIO_R2_US       CPP_HEX(0045)
#define VIO_R2_EUR      CPP_HEX(004D)
#define VIO_R3          CPP_HEX(0001)
#define VIO_R4_US       CPP_HEX(00F8)
#define VIO_R4_EUR      CPP_HEX(00FA)
#define VIO_R5          CPP_HEX(0000)
#define VIO_R6          CPP_HEX(0000)
#define VIO_R7          CPP_HEX(0000)

/* miscellaneous video IO constants */
#define VIOR1MASK       CPP_HEX(000D)
#define VIONFSCENMSK    CPP_HEX(0040)
#define VIOPLAYMAPS     CPP_HEX(00F1)
#define VIOPLAYCTRL     CPP_HEX(00E0)
#define CAMERA          0
#define MITSUBISHI      1

/* Video I/O quadrants */

#define ALL_QUAD        CPP_HEX(000F)

/* Vio hardware address */

#define VIO_HDW_WRITE_ADDR  0xA2A000
#define VIO_HDW_READ_ADDR   0xA2A000

/* video IO register flag write word definition */
#define VIOALLREGS      CPP_HEX(00FF)
typedef struct {
    unsigned_16     :8;
    unsigned_16     reg7:1;
    unsigned_16     reg6:1;
    unsigned_16     reg5:1;
    unsigned_16     reg4:1;
    unsigned_16     reg3:1;
    unsigned_16     reg2:1;
    unsigned_16     reg1:1;
    unsigned_16     reg0:1;
} VIO_WR_STRUCT;
```

```
typedef union {
     VIO_WR_STRUCT     b;
     unsigned_16       w;
} VIO_WR_TYPE;

/* Video IO register mapping type definitions */

/* write register 0 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     q1mode:2;       /* bit 6-7 */
        unsigned_16     intcam1:1;      /* bit 5 */
        unsigned_16     q2mode:2;       /* bit 3-4 */
        unsigned_16     intcam2:1;      /* bit 2 */
        unsigned_16     cineq1on:1;     /* bit 1 */
        unsigned_16     cineq2on:1;     /* bit 0 */
} VIO_REG0_B;

/* write register 1 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     nredmap:1;      /* bit 7 */
        unsigned_16     ngrnmap:1;      /* bit 6 */
        unsigned_16     nblumap:1;      /* bit 5 */
        unsigned_16     nwrcol:1;       /* bit 4 */
        unsigned_16     nrbwen:1;       /* bit 3 */
        unsigned_16     ngbwen:1;       /* bit 2 */
        unsigned_16     nbbwen:1;       /* bit 1 */
        unsigned_16     nwrbw:1;        /* bit 0 */
} VIO_REG1_B;

/* write register 2 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     nsysvtr:1;      /* bit 7 */
        unsigned_16     nfscen:1;       /* bit 6 */
        unsigned_16     syncsel:1;      /* bit 5 */
        unsigned_16     blnkds:1;       /* bit 4 */
        unsigned_16     eur_nus:1;      /* bit 3 */
        unsigned_16     nwrthr:1;       /* bit 2 */
        unsigned_16     cineon:1;       /* bit 1 */
        unsigned_16     ndisgrp:1;      /* bit 0 */
} VIO_REG2_B;

/* write register 3 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     pflgen:1;       /* bit 7 */
        unsigned_16     pflg2:1;        /* bit 6 */
        unsigned_16     pflg1:1;        /* bit 5 */
        unsigned_16     pflg0:1;        /* bit 4 */
        unsigned_16     flgsel:1;       /* bit 3 */
        unsigned_16     cflgen:1;       /* bit 2 */
        unsigned_16     hflip:1;        /* bit 1 */
        unsigned_16     tsten:1;        /* bit 0 */
```

```
} VIO_REG3_B;

/* write register 4 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     nvtrclkst:1;    /* bit 7 */
        unsigned_16     nvtrclkcl:1;    /* bit 6 */
        unsigned_16     nfrcelokd:1;    /* bit 5 */
        unsigned_16     lockden:1;      /* bit 4 */
        unsigned_16     ntestvid:1;     /* bit 3 */
        unsigned_16     svhson:1;       /* bit 2 */
        unsigned_16     ncombon:1;      /* bit 1 */
        unsigned_16     auxcol:1;       /* bit 0 */
} VIO_REG4_B;

/* write register 5 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     colordat7:1;    /* bit 7 */
        unsigned_16     colordat6:1;    /* bit 6 */
        unsigned_16     colordat5:1;    /* bit 5 */
        unsigned_16     colordat4:1;    /* bit 4 */
        unsigned_16     colordat3:1;    /* bit 3 */
        unsigned_16     colordat2:1;    /* bit 2 */
        unsigned_16     colordat1:1;    /* bit 1 */
        unsigned_16     colordat0:1;    /* bit 0 */
} VIO_REG5_B;

/* write register 6 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     bwdat7:1;       /* bit 7 */
        unsigned_16     bwdat8:1;       /* bit 6 */
        unsigned_16     bwdat5:1;       /* bit 5 */
        unsigned_16     bwdat4:1;       /* bit 4 */
        unsigned_16     bwdat3:1;       /* bit 3 */
        unsigned_16     bwdat2:1;       /* bit 2 */
        unsigned_16     bwdat1:1;       /* bit 1 */
        unsigned_16     bwdat0:1;       /* bit 0 */
} VIO_REG6_B;

/* write register 7 bit definitions */
typedef struct {
        unsigned_16     :8;             /* bits 15-8 in a 16 bit word */
        unsigned_16     q3mode:2;       /* bit 6-7 */
        unsigned_16     extcam1:1;      /* bit 5 */
        unsigned_16     q4mode:2;       /* bit 3-4 */
        unsigned_16     extcam2:1;      /* bit 2 */
        unsigned_16     cineq3on:1;     /* bit 1 */
        unsigned_16     cineq4on:1;     /* bit 0 */
} VIO_REG7_B;

typedef union {
        unsigned_16     w;
        VIO_REG0_B      b;
} VIO_REG0_U;
```

```
typedef union {
        unsigned_16     w;
        VIO_REG1_B      b;
   VIO_REG1_U;

typedef union {
        unsigned_16     w;
        VIO_REG2_B      b;
} VIO_REG2_U;

typedef union {
        unsigned_16     w;
        VIO_REG3_B      b;
} VIO_REG3_U;

typedef union {
        unsigned_16     w;
        VIO_REG4_B      b;
} VIO_REG4_U;

typedef union {
        unsigned_16     w;
        VIO_REG5_B      b;
} VIO_REG5_U;

typedef union {
        unsigned_16     w;
        VIO_REG6_B      b;
} VIO_REG6_U;

typedef union {
        unsigned_16     w;
        VIO_REG7_B      b;
} VIO_REG7_U;

typedef struct {
        VIO_REG0_U      vio_reg0;
        VIO_REG1_U      vio_reg1;
        VIO_REG2_U      vio_reg2;
        VIO_REG3_U      vio_reg3;
        VIO_REG4_U      vio_reg4;
        VIO_REG5_U      vio_reg5;
        VIO_REG6_U      vio_reg6;
        VIO_REG7_U      vio_reg7;
} VIO_REG_TYPE;

/* Vio Read register descriptions */

typedef struct {
    unsigned_16         :16;        /* bits 0-15 */
 } VIO_READ_REG0_B;

typedef struct {
    unsigned_16         :16;        /* bits 0-15 */
 } VIO_READ_REG1_B;
```

```
typedef struct {
    unsigned_16         :16;        /* bits 0-15 */
 } VIO_READ_REG2_B;

typedef struct {
    unsigned_16         :16;        /* bits 0-15 */
 } VIO_READ_REG3_B;

typedef struct {
    unsigned_16         :8;         /* bits 8-15 */
    unsigned_16    nfscoff:1;       /* bit 7 */
    unsigned_16     locked:1;       /* bit 6 */
    unsigned_16         :6;         /* bits 0-5 */
 } VIO_READ_REG4_B;

typedef struct {
    unsigned_16         :16;        /* bits 0-15 */
 } VIO_READ_REG5_B;

typedef struct {
    unsigned_16         :16;        /* bits 0-15 */
 } VIO_READ_REG6_B;

typedef struct {
    unsigned_16         :16;        /* bits 0-15 */
 } VIO_READ_REG7_B;

typedef union {
        unsigned_16      w;
        VIO_READ_REG0_B  b;
 } VIO_READ_REG0_U;

typedef union {
        unsigned_16      w;
        VIO_READ_REG1_B  b;
 } VIO_READ_REG1_U;

typedef union {
        unsigned_16      w;
        VIO_READ_REG2_B  b;
 } VIO_READ_REG2_U;

typedef union {
        unsigned_16      w;
        VIO_READ_REG3_B  b;
 } VIO_READ_REG3_U;

typedef union {
        unsigned_16      w;
        VIO_READ_REG4_B  b;
 } VIO_READ_REG4_U;

typedef union {
        unsigned_16      w;
        VIO_READ_REG5_B  b;
```

```
 } VIO_READ_REG5_U;

typedef union {
        unsigned_16      w;
        VIO_READ_REG6_B b;
 } VIO_READ_REG6_U;

typedef union {
        unsigned_16      w;
        VIO_READ_REG7_B b;
 } VIO_READ_REG7_U;

typedef struct {
      VIO_READ_REG0_U    rd_reg0;
      VIO_READ_REG1_U    rd_reg1;
      VIO_READ_REG2_U    rd_reg2;
      VIO_READ_REG3_U    rd_reg3;
      VIO_READ_REG4_U    rd_reg4;
      VIO_READ_REG5_U    rd_reg5;
      VIO_READ_REG6_U    rd_reg6;
      VIO_READ_REG7_U    rd_reg7;
 } VIO_READ_REG_TYPE;

/* Define a type to pass boolean status of quadrants */

typedef union {
     struct{
        unsigned_16         :12;   /* bits 4-15 not used */
        unsigned_16   quad4:1;     /* bit 3 lower_right quad */
        unsigned_16   quad3:1;     /* bit 2 lower_left quad */
        unsigned_16   quad2:1;     /* bit 1 upper_right quad */
        unsigned_16   quad1:1;     /* bit 0 upper_left quad */
        } b;
     unsigned_16 w;
               } QUAD_STATUS_TYPE;


#ifdef DECLARE

/* assign hardware register pointer */

VIO_REG_TYPE *const Vio_regs = (VIO_REG_TYPE *)VIO_HDW_WRITE_ADDR;
volatile VIO_READ_REG_TYPE *const Vio_rd_regs = (VIO_READ_REG_TYPE *)VIO_HDW_READ_
VIO_REG_TYPE  Vio_shdw;
VIO_WR_TYPE   Regs_writ;

#else

extern VIO_REG_TYPE *Vio_regs;
extern volatile VIO_READ_REG_TYPE *Vio_rd_regs;
extern VIO_REG_TYPE Vio_shdw;

#endif DECLARE

extern unsigned_16  Vio_maps[];
```

```
#ifndef NOEXTERN

extern void Vio_hdw_poweron();
extern void Vio_hdw_begin_reformat();
extern void Vio_hdw_end_reformat();
extern void Vio_cineoff();
extern void Vio_cineon();
extern void Vio_hdw_color();
extern void Vio_hdw_disable_color();
extern void Vio_hdw_enable_color();
extern boolean Vio_hdw_phaselockedloop();
extern void Vio_playoff();
extern void Vio_playon();
extern void Vio_hdw_quadmode();
extern void Vio_hdw_set_graphics();
extern void Vio_hdw_setcolor();
extern boolean Vio_hdw_vtrcoloron();
extern void Vio_hdw_vtr_type();
extern void Vio_hdw_trigger_frame();
extern void Vio_hdw_write_regs();
extern void Vio_hdw_reset();

extern void Vio_hdw_bw_image();
extern void Vio_hdw_bw_reslevels();
extern void Vio_hdw_bw_rgbtriplet();
extern void Vio_hdw_bw_triplet();
extern void Vio_hdw_col_triplet();
extern void Vio_hdw_flowturb();
extern void Vio_hdw_testcleanup();
extern void Vio_hdw_velocity();

#endif NOEXTERN

#endif VIO_HDW
```

```
/*
*************************************************************************
** $Source: /users2/prism/dss/revm/src/hdw/vio/RCS/vio_hdw.c,v $
** $Revision: 2.1 $
** $Date: 89/06/19 13:45:05 $
** $Author: goodnow $
** $Locker: mandel $
*************************************************************************
*************************************************************************
**
** DESCRIPTION: This file contains the main entry points for the
**              video i/o hardware
**
** ENTRYPOINT(S):  void Vio_hdw_poweron()
**                 void Vio_hdw_begin_reformat()
**                 void Vio_hdw_end_reformat()
**                 void Vio_cineoff()
**                 void Vio_cineon()
**                 void Vio_hdw_color()
**                 void Vio_hdw_disable_color()
**                 void Vio_hdw_enable_color()
**                 boolean Vio_hdw_phaselockedloop()
**                 void Vio_playoff()
**                 void Vio_playon()
**                 void Vio_hdw_quadmode()
**                 void Vio_hdw_set_graphics()
**                 void Vio_hdw_setcolor()
**                 boolean Vio_hdw_vtrcoloron()
**                 void Vio_hdw_vtr_type()
**                 void Vio_hdw_trigger_frame()
**                 void Vio_hdw_write_regs()
**
*************************************************************************
**/
#include "dssdefs.h"
#include "scrn.h"
#include "cine.h"
#include "proc.h"
#include "status.h"
#define DECLARE
#include "vio_hdw.h"

#define THRESHOLD 3
#define SUMMING 0

/*
*************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_poweron()
**
** INPUTS:  none
**
** OUTPUTS:  All the Vio hardware registers are given initial values
**
** CAVEATS:
**
```

```
**************************************************************************
*/

void Vio_hdw_poweron()
     {

/* update all the shadow registers  */

     Vio_shdw.vio_reg0.w = VIO_R0;
     Vio_shdw.vio_reg1.w = VIO_R1;
     Vio_shdw.vio_reg2.w = us_eur_mode() ? VIO_R2_EUR : VIO_R2_US ;
     Vio_shdw.vio_reg3.w = VIO_R3;
     Vio_shdw.vio_reg4.w = us_eur_mode() ? VIO_R4_EUR : VIO_R4_US ;
     Vio_shdw.vio_reg5.w = VIO_R5;
     Vio_shdw.vio_reg6.w = VIO_R6;
     Vio_shdw.vio_reg7.w = VIO_R7;

/* Update the actual hardware */

     Regs_writ.w = 0x0FF;
     Vio_hdw_write_regs();
     )

/*
**************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_begin_reformat()
**
** INPUTS:  none
**
** OUTPUTS: none....just a stub for now
**
** CAVEATS:
**
**************************************************************************
*/

void Vio_hdw_begin_reformat()
   {
   }

/*
**************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_end_reformat()
**
** INPUTS:  none
**
** OUTPUTS:
**
** CAVEATS:
**
**************************************************************************
*/

void Vio_hdw_end_reformat()
```

```
{
     Regs_writ.w = FALSE;

     /* set color burst on or off depending on current mode */
     Vio_hdw_color();
     Vio_hdw_write_regs();
}

/*
**************************************************************************
**
** ENTRYPOINT(S): void Vio_cineoff()
**
** INPUTS:  none
**
** OUTPUTS:  Update the vio hardware to real time
**
** CAVEATS:
**
**************************************************************************
*/

void Vio_cineoff()
{
     Regs_writ.w = FALSE;  /* clear write-reg flag word */

     Vio_shdw.vio_reg2.b.cineon = 0;       /* use system bus */
     Regs_writ.b.reg2 = TRUE;

     /* if we are exiting cine into playback, set up for that  */
     if ( Play_on() )
        Vio_playon();

     /* update the vio registers that may have changed */
     Vio_hdw_write_regs();
}

/*
**************************************************************************
**
** ENTRYPOINT(S): void Vio_cineon()
**
** INPUTS: none
**
** OUTPUTS:  set up Vio for cine display on
**
** CAVEATS:
**
**************************************************************************
*/

void Vio_cineon()
{
   Vio_shdw.vio_reg2.b.nsysvtr = 0;       /* no vtr playback */
   Vio_shdw.vio_reg2.b.syncsel = 0;       /* use system clock */
   Vio_shdw.vio_reg2.b.cineon = 1;        /* Cine bus */
```

```
   /* Update the hardware */
   Vio_regs->vio_reg2.w = Vio_shdw.vio_reg2.w;
}

/*
**********************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_color()
**
** INPUTS:  none
**
** OUTPUTS:  setup colorburst for color flow imaging if color video
**           is enabled or for BW imaging if not.
**
** CAVEATS:  This routine does not update the registers since writing to
**           reg2 causes a flash.  Do it once for all times this routine is
**           called during a screen change.
**
**********************************************************************
*/

void Vio_hdw_color()
{
   boolean color_flag;

/*************************************************/
/* Check if color video is enabled.  If it is,   */
/* update the proper registers.                  */
/*************************************************/
   if (Cine_playthru_on()) {
      if (Play_on())
         color_flag = TRUE;
      else
         color_flag = Scrn_q_colorvideo() || Cine_q_colorvideo();
   }
   else if (Cine_display_on()) {
      color_flag = Cine_q_colorvideo();
   }
   else {
      if (Play_on())
         color_flag = TRUE;
      else
         color_flag = Scrn_q_colorvideo();
   }

   if (color_flag) {
      /* 1 = color burst off; if off then turn it on. */
      if (Vio_shdw.vio_reg2.b.nfscen) {
         Vio_shdw.vio_reg2.b.nfscen = 0;
         Regs_writ.b.reg2 = TRUE;
      }
   }
   else { /* bw mode */
      /* 0 = color burst on; if on then turn it off. */
      if (!Vio_shdw.vio_reg2.b.nfscen) {
```

```
        Vio_shdw.vio_reg2.b.nfscen = 1;
        Regs_writ.b.reg2 = TRUE;



/*
**********************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_disable_color()
**
** INPUTS:  none
**
** OUTPUTS:  disable color by forcing the latched R,G,B maps data to
**           0 independent of the map contents
**
** CAVEATS: Cine or Play must NOT be on if this is called
**
**********************************************************************
*/

void Vio_hdw_disable_color()
   {
   Vio_shdw.vio_reg1.b.nredmap = 1;
   Vio_shdw.vio_reg1.b.ngrnmap = 1;
   Vio_shdw.vio_reg1.b.nblumap = 1;

 * force summing maps */

   Vio_shdw.vio_reg0.b.q1mode = 0;
   Vio_shdw.vio_reg0.b.q2mode = 0;
   Vio_shdw.vio_reg7.b.q3mode = 0;
   Vio_shdw.vio_reg7.b.q4mode = 0;

   Vio_regs->vio_reg0.w = Vio_shdw.vio_reg0.w;
   Vio_regs->vio_reg1.w = Vio_shdw.vio_reg1.w;
   Vio_regs->vio_reg7.w = Vio_shdw.vio_reg7.w;
   }

/*
**********************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_enable_color()
**
** INPUTS:  none
**
** OUTPUTS:  enable color by enabling R,G,B maps
**
** CAVEATS: this routine does NOT check to see if Cine or Play is on!
**
**********************************************************************
*/

void Vio_hdw_enable_color()
   {
   QUAD_STATUS_TYPE *thresholdmap;
```

```
   unsigned_16 status_word;

   Vio_shdw.vio_reg1.b.nredmap = 0;
   Vio_shdw.vio_reg1.b.ngrnmap = 0;
   Vio_shdw.vio_reg1.b.nblumap = 0;

   Regs_writ.b.reg1 = TRUE;

   status_word = Scrn_thres_on();
   thresholdmap = (QUAD_STATUS_TYPE *)&status_word;
   if (thresholdmap->b.quad1)
      {
      Vio_shdw.vio_reg0.b.q1mode = 3;
      Regs_writ.b.reg0 = TRUE;
      }
   if (thresholdmap->b.quad2)
      {
      Vio_shdw.vio_reg0.b.q2mode = 3;
      Regs_writ.b.reg0 = TRUE;
      }
   if (thresholdmap->b.quad3)
      {
      Vio_shdw.vio_reg7.b.q3mode = 3;
      Regs_writ.b.reg7 = TRUE;
      }
   if (thresholdmap->b.quad4)
      {
      Vio_shdw.vio_reg7.b.q4mode = 3;
      Regs_writ.b.reg7 = TRUE;
      }

   Vio_hdw_write_regs();
   }

/*
**************************************************************************
**
** ENTRYPOINT(S): boolean Vio_hdw_phaselockedloop
**
** INPUTS:
**
** OUTPUTS:  1 = phased locked playback
**           0 = no phase lock
**
** CAVEATS:
**
**************************************************************************
*/

boolean Vio_hdw_phaselockedloop()
   {
   VIO_READ_REG4_U reg4;

   reg4.w = Vio_rd_regs->rd_reg4.w;
   return (reg4.b.locked);
   }
```

```
/*
**********************************************************************
**
** ENTRYPOINT(S): void Vio_playoff()
**
** INPUTS:  none
**
** OUTPUTS:  setup vio hardware for realtime imaging.
**
** CAVEATS:
**
**********************************************************************
*/

void Vio_playoff()
{
   Regs_writ.w = FALSE;

   Vio_shdw.vio_reg2.b.nsysvtr = 0;
   Vio_shdw.vio_reg2.b.syncsel = 0;

   /* Set up video for color or bw */
   Vio_hdw_color();

   /* update reg2 manually */
   Regs_writ.b.reg2 = FALSE;
   Vio_regs->vio_reg2.w = Vio_shdw.vio_reg2.w;
}

/*
**********************************************************************
**
** ENTRYPOINT(S): void Vio_playon()
**
** INPUTS:  none
**
** OUTPUTS:  setup VIO hardware for vtr playback
**
** CAVEATS: cine overrides vtr playback !
**          NOTE: reg2 & reg4 are updated here !
**
**********************************************************************
*/

void Vio_playon()
{
   if (Cine_playthru_on() || Cine_display_on()) {
      Vio_shdw.vio_reg2.b.syncsel = 0;
      Vio_shdw.vio_reg2.b.nsysvtr = 0;
   }
   else { /* In VTR playback ONLY. */
      Vio_shdw.vio_reg2.b.nsysvtr = 1;
      Vio_shdw.vio_reg2.b.syncsel = 1;

      if (Cine_quick_review_on()) {
```

```
            Cine_off_quick_review();
            Vio_shdw.vio_reg2.b.cineon = 0;      /* use system bus */
         )
      )
      /* set up colorburst */
      Vio_hdw_color();

      /* Update reg2 manually */
      Regs_writ.b.reg2 = FALSE;
      Vio_regs->vio_reg2.w = Vio_shdw.vio_reg2.w;
)

/*
*************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_quadmode( qaud, maptype, cinevtrcolor )
**
** INPUTS:  unsigned_16 quad = 0 for upper left
**                           = 1 for upper right
**                           = 2 for lower left
**                           = 3 for lower right
**                           = 4 for ALL quadrants
**
**          unsigned_16 maptype = 0 for summing
**                              = 1 for threshold
**
**          boolean cinevtrcolor = TRUE if cinevtrcolor
**
** OUTPUTS:
**
** CAVEATS:
**
*************************************************************************
*/

void Vio_hdw_quadmode( quad, maptype, cinevtrcolor )
unsigned_16 quad;
unsigned_16 maptype;
boolean cinevtrcolor;
   {
   switch (quad)
      {

      case 0 :       /* upper left quad */
        if (Vio_shdw.vio_reg0.b.q1mode != (maptype?THRESHOLD:SUMMING))
           {
           Vio_shdw.vio_reg0.b.q1mode = (maptype?THRESHOLD:SUMMING);
           Regs_writ.b.reg0 = TRUE;
           )
        if (Vio_shdw.vio_reg0.b.cineq1on != cinevtrcolor)
           {
           Vio_shdw.vio_reg0.b.cineq1on = cinevtrcolor;
           Regs_writ.b.reg0 = TRUE;
           }
        break;
```

```
      case 1 :        /* upper right quad */
        if (Vio_shdw.vio_reg0.b.q2mode != (maptype?THRESHOLD:SUMMING))
           {
           Vio_shdw.vio_reg0.b.q2mode = (maptype?THRESHOLD:SUMMING);
           Regs_writ.b.reg0 = TRUE;
           }
        if (Vio_shdw.vio_reg0.b.cineq2on != cinevtrcolor)
           {
           Vio_shdw.vio_reg0.b.cineq2on = cinevtrcolor;
           Regs_writ.b.reg0 = TRUE;
           }
        break;

      case 2 :        /* lower left quad */
        if (Vio_shdw.vio_reg7.b.q3mode != (maptype?THRESHOLD:SUMMING))
           {
           Vio_shdw.vio_reg7.b.q3mode = (maptype?THRESHOLD:SUMMING);
           Regs_writ.b.reg7 = TRUE;
           }
        if (Vio_shdw.vio_reg7.b.cineq3on != cinevtrcolor)
           {
           Vio_shdw.vio_reg7.b.cineq3on = cinevtrcolor;
           Regs_writ.b.reg7 = TRUE;
           }
        break;

      case 3 :        /* lower right quad */
        if (Vio_shdw.vio_reg7.b.q4mode != (maptype?THRESHOLD:SUMMING))
           {
           Vio_shdw.vio_reg7.b.q4mode = (maptype?THRESHOLD:SUMMING);
           Regs_writ.b.reg7 = TRUE;
           }
        if (Vio_shdw.vio_reg7.b.cineq4on != cinevtrcolor)
           {
           Vio_shdw.vio_reg7.b.cineq4on = cinevtrcolor;
           Regs_writ.b.reg7 = TRUE;
           }
        break;

      case 4 :        /* ALL quadrants */
        if (Vio_shdw.vio_reg0.b.q1mode != (maptype?THRESHOLD:SUMMING))
           {
           Vio_shdw.vio_reg0.b.q1mode = (maptype?THRESHOLD:SUMMING);
           Regs_writ.b.reg0 = TRUE;
           }
        if (Vio_shdw.vio_reg0.b.cineq1on != cinevtrcolor)
           {
           Vio_shdw.vio_reg0.b.cineq1on = cinevtrcolor;
           Regs_writ.b.reg0 = TRUE;
           }
        if (Vio_shdw.vio_reg0.b.q2mode != (maptype?THRESHOLD:SUMMING))
           {
           Vio_shdw.vio_reg0.b.q2mode = (maptype?THRESHOLD:SUMMING);
           Regs_writ.b.reg0 = TRUE;
           }
        if (Vio_shdw.vio_reg0.b.cineq2on != cinevtrcolor)
```

```
            {
            Vio_shdw.vio_reg0.b.cineq2on = cinevtrcolor;
            Regs_writ.b.reg0 = TRUE;
            }
         if (Vio_shdw.vio_reg7.b.q3mode != (maptype?THRESHOLD:SUMMING))
            {
            Vio_shdw.vio_reg7.b.q3mode = (maptype?THRESHOLD:SUMMING);
            Regs_writ.b.reg7 = TRUE;
            }
         if (Vio_shdw.vio_reg7.b.cineq3on != cinevtrcolor)
            {
            Vio_shdw.vio_reg7.b.cineq3on = cinevtrcolor;
            Regs_writ.b.reg7 = TRUE;
            }
         if (Vio_shdw.vio_reg7.b.q4mode != (maptype?THRESHOLD:SUMMING))
            {
            Vio_shdw.vio_reg7.b.q4mode = (maptype?THRESHOLD:SUMMING);
            Regs_writ.b.reg7 = TRUE;
            }
         if (Vio_shdw.vio_reg7.b.cineq4on != cinevtrcolor)
            {
            Vio_shdw.vio_reg7.b.cineq4on = cinevtrcolor;
            Regs_writ.b.reg7 = TRUE;
            }
         break;
      }
   Vio_hdw_write_regs();
   }


/*
**************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_set_graphics( on_off )
**
** INPUTS:  unsigned_8 type  = 0  for off
**                           = 1  for on
**
** OUTPUTS:  VIO hardware reg2 is set (ON) or cleared (OFF)
**
** CAVEATS: The hardware IS updated here!
**
**************************************************************************
*/

void Vio_hdw_set_graphics( on_off )
unsigned_8 on_off;
   {
   Vio_shdw.vio_reg2.b.ndisgrp = on_off;
   Vio_regs->vio_reg2.w = Vio_shdw.vio_reg2.w;
   Regs_writ.b.reg2 = FALSE;
   }

/*
**************************************************************************
**
```

```
** ENTRYPOINT(S): void Vio_hdw_setcolor()
**
** INPUTS:  none
**
** OUTPUTS:  Vio hardware setup for color or bw
**
** CAVEATS: This entry point is from non-members so must
**          call the Vio_hdw_write_regs() from here
**
*****************************************************************************
*/

void Vio_hdw_setcolor()
{
   Regs_writ.w = FALSE;
   Vio_hdw_color();
   if ( Sys_realtime() )
      Vio_hdw_write_regs();
}

/*
*****************************************************************************
**
** ENTRYPOINT(S): boolean Vio_hdw_vtrcoloron()
**
** INPUTS:  none
**
** OUTPUTS:  1 = Vtr color playback
**           0 = otherwise
**
** CAVEATS:
**
*****************************************************************************
*/

boolean Vio_hdw_vtrcoloron()
  {
  VIO_READ_REG4_U reg4;

  reg4.w = Vio_rd_regs->rd_reg4.w;
  return (reg4.b.nfscoff);
  }

/*
*****************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_vtr_type(type)
**
** INPUTS:  unsigned_8 type  = 0  for regular VHS
**                           = 1  for SUPER VHS
**
** OUTPUTS:  VIO hardware reg4 is updated
**
** CAVEATS: the hardware IS modified here
**
*****************************************************************************
```

```
*/

void Vio_hdw_vtr_type( type )
unsigned_8 type;
   {
   /* set the vtr type */
   Vio_shdw.vio_reg4.b.svhson = type;
   Vio_regs->vio_reg4.w = Vio_shdw.vio_reg4.w;
   Regs_writ.b.reg4 = FALSE;
   }


/*
**************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_trigger_frame()
**
** INPUTS: option - trigger camera or mitsubishi
**
** OUTPUTS:  a frame is taken by the mitsubishi or the camera
**
** CAVEATS:
**
**************************************************************************
*/

void Vio_hdw_trigger_frame( option )
     unsigned_8 option;

     unsigned_32 msec_to_tick();
     signed_16 sync=FALSE;

     /* wait for freeze to stabilize */
     pause_p( msec_to_tick( 1000 ) );

     /* trigger the shutter */
     if ( option == CAMERA ) {
          /* select system sync if necessary */
          if ( sync = Vio_shdw.vio_reg2.b.syncsel ) {
               Vio_shdw.vio_reg2.b.syncsel = FALSE;
               Vio_regs->vio_reg2.w = Vio_shdw.vio_reg2.w;
          }
          /* open the shutter */
          Vio_shdw.vio_reg0.b.intcam1 = TRUE;
     }
     else
          Vio_shdw.vio_reg0.b.intcam2 = TRUE;
     Vio_regs->vio_reg0.w = Vio_shdw.vio_reg0.w;

     /* wait one second for picture to be taken */
     pause_p( msec_to_tick( 1000 ) );

     /* stop the shutter */
     if ( option == CAMERA )
          /* close the shutter */
          Vio_shdw.vio_reg0.b.intcam1 = FALSE;
```

```
     else
          Vio_shdw.vio_reg0.b.intcam2 = FALSE;
     Vio_regs->vio_reg0.w = Vio_shdw.vio_reg0.w;

     /* wait one or 12 seconds (depending on camera type) for */
     /* refresh */
     if ( option != MITSUBISHI )
          if ( Proc_switchstate(CAM_TYPE_SWITCH) )
               pause_p(msec_to_tick(us_eur_mode() ? 14000:12000));  /* Aspect */
          else
               pause_p( msec_to_tick( 3000 ) );   /* Matrix */
     else
          pause_p( msec_to_tick( 1000 ) );   /* Mitsubishi */

     /* restore system sync if changed */
     if ( sync ) {
          Vio_shdw.vio_reg2.b.syncsel = TRUE;
          Vio_regs->vio_reg2.w = Vio_shdw.vio_reg2.w;
     }

     /* signal controller that we're done */
     request_service(SYS_STATUS_FRAMETRIGDONE);
}


/*
*************************************************************************
**
** ENTRYPOINT(S): void Vio_hdw_write_regs()
**
** INPUTS:
**
** OUTPUTS:  The hardware registers are conditionaly updated
**
** CAVEATS:
**
*************************************************************************
*/

void Vio_hdw_write_regs()
{
   if (!Regs_writ.w)  return;

   if (Regs_writ.b.reg0)
      Vio_regs->vio_reg0.w = Vio_shdw.vio_reg0.w;
   if (Regs_writ.b.reg1)
      Vio_regs->vio_reg1.w = Vio_shdw.vio_reg1.w;
   if (Regs_writ.b.reg2)
      Vio_regs->vio_reg2.w = Vio_shdw.vio_reg2.w;
   if (Regs_writ.b.reg3)
      Vio_regs->vio_reg3.w = Vio_shdw.vio_reg3.w;
   if (Regs_writ.b.reg4)
      Vio_regs->vio_reg4.w = Vio_shdw.vio_reg4.w;
   if (Regs_writ.b.reg5)
      Vio_regs->vio_reg5.w = Vio_shdw.vio_reg5.w;
   if (Regs_writ.b.reg6)
```

```
      Vio_regs->vio_reg6.w = Vio_shdw.vio_reg6.w;
   if (Regs_writ.b.reg7)
      Vio_regs->vio_reg7.w = Vio_shdw.vio_reg7.w;

   Regs_writ.w = 0;
}
```