

**An Analysis of
"The Definition of a Production Quality Ada Compiler"**

**Volume II
PQAC Test Suite**

AD-A219 484

Prepared by
B. A. PETRICK
Software Development Department
S. J. YANKE
Systems Software Engineering Department
Engineering Group
The Aerospace Corporation
El Segundo, CA 90245

13 March 1989

DTIC
ELECTE
S **D** **Co** **D**
MAR 14 1990

Prepared for
SPACE SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
Los Angeles Air Force Base
P.O. Box 92960
Los Angeles, CA 90009-2960

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

90 03 13 065

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TR-0089(4902-03)-1 Vol II			5. MONITORING ORGANIZATION REPORT NUMBER(S) SSD-TR-89-82	
6a. NAME OF PERFORMING ORGANIZATION The Aerospace Corporation Engineering Group		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Air Force Systems Command Space Systems Division	
6c. ADDRESS (City, State, and ZIP Code) 2350 E. El Segundo Blvd. El Segundo, CA 90245			7b. ADDRESS (City, State, and ZIP Code) Los Angeles Air Force Base Los Angeles, CA 90009-2960	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F04701-88-C-0089	
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.	
			PROJECT NO.	
			TASK NO.	
			WORK UNIT ACCESSION NO.	
11. TITLE (Include Security Classification) An Analysis of "The Definition of a Production Quality Ada Compiler" Volume II PQAC Test Suite				
12. PERSONAL AUTHOR(S) B. A. Petrick, S. J. Yanke				
13a. TYPE OF REPORT		13b. TIME COVERED FROM TO		15. PAGE COUNT 316
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP		
			Ada compiler, selection; Ada compiler, procuring;	
			Ada compiler, specifications; (cont.)	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
This volume contains the Production Quality Ada Compiler (PQAC) test suite source code and operating instructions. This test suite was derived from the requirements in "The Definition of a Production Quality Ada Compiler", SD-TR-87-29.				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS REPORT ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Capt. John Brill			22b. TELEPHONE (Include Area Code) (213) 643-2532	
			22c. OFFICE SYMBOL SSD/ALR	

18. SUBJECT TERMS (Continued)

Ada compiler, evaluating; Ada Compiler, requirements;
Ada compiler, test suite; production quality Ada compiler;
Project Ada compiler.

TABLE OF CONTENTS

1. INTRODUCTION	1
2. PQAC TEST SUITE FILES	3
3. ADDING A COMPILER TO THE PQAC TEST SUITE DOMAIN	5
4. COMPILING THE PQAC TEST SUITE SUPPORT SOFTWARE	7
5. TEST SUITE EXECUTION	9
6. TEST FILE EXECUTION	11
7. SUPPORT SOFTWARE PACKAGE DESCRIPTIONS	17
8. EXAMPLE TEST WEIGHTING FILE	19
9. OPERATING SYSTEM COMMAND INTERPRETER	23
10. PQAC SUPPORT SOFTWARE PACKAGES (Alphabetical)	29
11. PQAC TEST FILES (T000000 through T080800)	137

A-1	
1. INTRODUCTION	✓
2. PQAC TEST SUITE FILES	
3. ADDING A COMPILER TO THE PQAC TEST SUITE DOMAIN	
4. COMPILING THE PQAC TEST SUITE SUPPORT SOFTWARE	
5. TEST SUITE EXECUTION	
6. TEST FILE EXECUTION	
7. SUPPORT SOFTWARE PACKAGE DESCRIPTIONS	
8. EXAMPLE TEST WEIGHTING FILE	
9. OPERATING SYSTEM COMMAND INTERPRETER	
10. PQAC SUPPORT SOFTWARE PACKAGES (Alphabetical)	
11. PQAC TEST FILES (T000000 through T080800)	



1. INTRODUCTION

This document contains a complete description of the Production Quality Ada Compiler Test Suite (PQAC). This test suite has been created to test the requirements set forth in "The Definition of a Production Quality Ada Compiler".

The PQAC test suite contains 150 individual tests and the support software needed to execute the tests. This volume will attempt to explain all of the components of the support software as well as providing instructions for creating new tests, compiling the support software, running the tests, and summarizing the PQAC test suite results. The source code for the test suite and test files is contained in Sections 9, 10, and 11 of this volume.

The PQAC test suite was designed to be independent of the Ada compiler and environment under test. In other words, it was desired that each of the 150 tests could be run without modification for all compilers. In practice, this is an almost impossible task. However, by standardizing both the operating system interface and the compiler option syntax, the amount of effort required to rehost the test suite has been minimized.

Tables in the support software capture compiler and operating system dependent information. Before running the test suite, information about the current compiler and environment must be entered into these tables. However, once this data has been entered it becomes a permanent part of the test suite data base. Currently, data for the DEC VAX V1.4 and Telesoft TeleGen2 V3.15 Ada compilers both running under VAX VMS has been included in the test suite. Examples used throughout this volume will be specific to one of these two compilers. As new compilers are added to the test suite, the capability for running the test suite on these compilers will remain.

After all of the compiler and environment specific information has been incorporated into the test suite and support software, the support software may be compiled. Each of the individual tests may then be executed. With the exception of a few special tests, the tests may be executed in any order. Each test automatically records its results in the test suite data base. After all of the tests have been executed,

an analysis of the results may be obtained.

Several special purpose functions operating on text files have been developed in the support software. They include functions to parse a test file, count the number of Ada source code lines in a file, retrieve the size of a file, expand a section of test code containing embedded special symbols, and a function that reads in the test result data with assigned weights for each test and creates a test report. Each of these functions is discussed at the appropriate place in the following pages.

2. PQAC TEST SUITE FILES

The following is a list of every file included in the PQAC test suite.
Other temporary files will be created by executing the test suite.

-- Test Suite Software Description (This File):

DESCRIPTION.TXT

-- Operating System Interpreter:

PERFORM.COM

-- Test Weighting Information Data File:

WEIGHTS.DAT

-- Support Software Source Code:

-- File_Name contains either a package or procedure body.

-- File_Name_ contains a package specification.

COMMON.ADA	EXPAND.ADA	PQAC_IO.ADA	SCRIPT.ADA	TIMES.ADA
COMMON.ADA	EXPAND.ADA	PQAC_IO.ADA	SCRIPT.ADA	TIMES.ADA
COMPARE.ADA	NAMES.ADA	RATING.ADA	SUPPORT.ADA	TWINE.ADA
COMPARE.ADA	PARSE.ADA	RATING.ADA	SYNTAX.ADA	TWINE.ADA
COUNT.ADA	PARSE.ADA	RESULT.ADA	SYNTAX.ADA	
COUNT.ADA		RESULT.ADA	TABLES.ADA	

-- 150 Test Files:

T000000.TST	T030305.TST	T030803.TST	T050300.TST	T060703.TST
T010100.TST	T030306.TST	T030804.TST	T060100.TST	T060801.TST
T020100.TST	T030307.TST	T040101.TST	T060201.TST	T060802.TST
T020200.TST	T030308.TST	T040102.TST	T060202.TST	T060900.TST
T020300.TST	T030309.TST	T040103.TST	T060203.TST	T061001.TST
T020401.TST	T030310.TST	T040104.TST	T060301.TST	T061002.TST
T020402.TST	T030311.TST	T040105.TST	T060302.TST	T061003.TST
T020403.TST	T030401.TST	T040106.TST	T060303.TST	T061004.TST
T020501.TST	T030402.TST	T040201.TST	T060304.TST	T061101.TST
T020502.TST	T030403.TST	T040202.TST	T060305.TST	T061102.TST
T030101.TST	T030404.TST	T040203.TST	T060306.TST	T061201.TST
T030102.TST	T030405.TST	T040204.TST	T060307.TST	T061202.TST
T030103.TST	T030406.TST	T040205.TST	T060308.TST	T061203.TST
T030104.TST	T030407.TST	T040206.TST	T060309.TST	T061204.TST
T030105.TST	T030408.TST	T040207.TST	T060310.TST	T061205.TST
T030106.TST	T030501.TST	T040208.TST	T060401.TST	T061206.TST
T030201.TST	T030502.TST	T040209.TST	T060402.TST	T061207.TST
T030202.TST	T030601.TST	T040301.TST	T060403.TST	T061208.TST
T030203.TST	T030602.TST	T040302.TST	T060404.TST	T070100.TST
T030204.TST	T030701.TST	T040303.TST	T060501.TST	T070200.TST
T030205.TST	T030702.TST	T040304.TST	T060502.TST	T070300.TST
T030206.TST	T030703.TST	T040305.TST	T060503.TST	T070400.TST
T030207.TST	T030704.TST	T050101.TST	T060504.TST	T070500.TST
T030208.TST	T030705.TST	T050102.TST	T060505.TST	T080100.TST
T030209.TST	T030706.TST	T050103.TST	T060506.TST	T080200.TST
T030301.TST	T030707.TST	T050104.TST	T060601.TST	T080300.TST
T030302.TST	T030708.TST	T050201.TST	T060602.TST	T080400.TST
T030303.TST	T030709.TST	T050202.TST	T060603.TST	T080500.TST
T030304.TST	T030801.TST	T050203.TST	T060701.TST	T080600.TST
T030802.TST	T050204.TST	T060702.TST	T080700.TST	T080800.TST

3. ADDING A COMPILER TO THE PQAC TEST SUITE DOMAIN

Currently, the PQAC test suite is capable of testing two compilers. Both of these compilers are hosted on a VAX running VAX VMS. In order to add a compiler to the test suite domain, several of the files listed in the previous section must be modified.

First, if the new compiler is being run under a system other than VAX VMS then a new command interpreter for that system must be created. Without knowing what the system is, it is impossible to say here exactly what procedure should be followed to develop this interpreter. However, its functionality is clearly defined by the current interpreter and in the accompanying comments. If the new compiler is being run under VAX VMS, then modifying the current interpreter, PERFORM.COM, will be sufficient. The areas that must be modified are marked, and comments on what to change are included in the header comments inside the interpreter file. In short, the VAX VMS DCL variable called Test\$Compiler is set inside PERFORM.COM. Only sections of PERFORM.COM that test this variable, e.g.,

```
$ IF Test$Compiler .EQS. "DEC_VAX_V1_4" THEN ...
```

need be modified. These sections are easily found using text searches.

Second, the package specification TABLES found in the file TABLES.ADA must be modified. The comments at the beginning of that file explain how to add a compiler to the tables contained there. For each new compiler, a name must be added to the enumeration type Compiler_Domain in the TABLES package. An entry for this new name must then be added to the data tables contained in that package. These tables contain compiler invocation syntax for the standard compiler options as well as other information.

Finally, some of the 150 test files may need to be modified. All of the tests that contain information specific to a compiler should be examined to determine what other changes need be made for a new compiler. All of these tests contain statements starting with "--X BEGIN" and "--X END". These statements are special meta symbols the test parsing procedure recognizes. They delineate portions of a test that are specific to a particular compiler. Only a few of the tests contain these statements.

The complete syntax of these and other parse commands is discussed in detail later in this volume.

For some of the tests, it will only become apparent that they need to be modified after they have been executed. This will usually be indicated by a test taking too much time or failing completely. Some tests actually print out messages stating they have to be modified. When any of these results occur, the test can simply be modified and rerun. The results will be rerecorded in the result data base. When multiple results for a test are found, the latest result recorded will be used in the result analysis.

4. COMPILING THE PQAC TEST SUITE SUPPORT SOFTWARE

After a compiler has been included into the test suite support software tables, the support software may be compiled. The order of compilation is as follows:

```
COMPILE TWINE_.ADA
COMPILE NAMES_.ADA
COMPILE TABLES_.ADA
COMPILE PQAC_IO_.ADA
COMPILE COMMON_.ADA
COMPILE SYNTAX_.ADA
COMPILE SCRIPT_.ADA
COMPILE TIMES_.ADA
COMPILE RATING_.ADA
COMPILE COUNT_.ADA
COMPILE EXPAND_.ADA
COMPILE PARSE_.ADA
COMPILE RESULT_.ADA
COMPILE COMPARE_.ADA
COMPILE TWINE.ADA
COMPILE PQAC_IO.ADA
COMPILE COMMON.ADA
COMPILE SYNTAX.ADA
COMPILE SCRIPT.ADA
COMPILE TIMES.ADA
COMPILE RATING.ADA
COMPILE RESULT.ADA
COMPILE COUNT.ADA
COMPILE EXPAND.ADA
COMPILE PARSE.ADA
COMPILE COMPARE.ADA
COMPILE SUPPORT.ADA
LINK      SUPPORT
```

After the procedure SUPPORT has been linked by this last statement, the executable file SUPPORT.EXE will be created. When executed, this procedure reads its parameters from a predetermined file. These parameters are then used to execute one of the possible functions that may be performed. Each of these functions has its own unique parameters. The possible functions and their syntax are fully documented in both the SUPPORT.ADA file, and in each of the separate package specifications for each of the functions. For the most part, a user of the test suite does not have to worry about these details. The calling of these functions is performed automatically by the support software, or by the command interpreter PERFORM.COM.

5. TEST SUITE EXECUTION

Once the test suite support software has been compiled, the test suite is ready for execution. The execution of the test suite is driven by the PERFORM.COM command interpreter. The allowed arguments to this interpreter are SETUP, RATING, or a test name such as T010100. The descriptions of each of these options are described below.

Several directories may be set up to help maintain the database. Logical variables at the beginning of PERFORM.COM are set to define the current working Ada library, a home directory, the directory containing SUPPORT.EXE, the directory containing all of the T?????.TST files, a directory for writing out all of the T?????.OUT files, and finally a directory containing the support software. Some of the packages of the support software must be recompiled whenever the current working Ada library needs to be purged.

The @PERFORM SETUP must be the first command that is executed. This causes a STATE.DAT file to be created and the library to be initialized. When this command is executed, the user will be presented with a list of possible compilers. He must then enter the desired compiler name. The STATE.DAT file contains status information about the current Ada library as well as the name of the current compiler. This file must be present for the rest of the execution of the test suite. If the state of the test suite becomes corrupted or the working Ada library exceeds a capacity, @PERFORM SETUP may be called again. The results obtained up to that point will still be retained.

The first test to be executed should be T000000. This file contains functionally identical Ada and FORTRAN code segments. Special commands in this file direct the software to compile and execute the Ada code four times, each time using a different compiler option. The FORTRAN code is also compiled and executed. The information from all of these compiles is stored in the file COMPARE.DAT. This test is unique in the fact that it is the only test that is strictly test overhead. No results are directly generated by this test. However, the file COMPARE.DAT is read by the tests T020401, T020402, T020403, T020501, and T020502. These tests generate pass/fail results.

After @PERFORM SETUP and @PERFORM T000000 have been completed, the 149 tests that have not been executed may be executed in any order. This is accomplished using @PERFORM T010100 through @PERFORM T080800. A command file may be easily set up using VAX VMS DCL to automate this procedure. There will most certainly be some tests that do not execute correctly without modification by the tester. When these tests are identified, they can be modified and rerun at any time without consequence. Duplicate results for these tests may then be recorded in the result data base, but only the latest result is used when generating the result analysis. The execution of each T?????.TST test file is recorded in the corresponding T?????.OUT file. When a test fails, or when more information about a test is needed, this T?????.OUT file may be consulted.

The execution of each test causes a line to be added to the results database file. The name of this file will be the name of the current compiler appended with ".DAT" (in the case of VAX VMS), e.g., DEC_VAX_V1_4.DAT. Once every test has been completed without error, this file will contain a complete set of all the results.

After all of the tests have been executed, @PERFORM RATING may be called. This causes a file with the name compiler.LIS to be created, e.g., DEC_VAX_V1_4.LIS. The rating procedure reads in the file DEC_VAX_V1_4.DAT and the file WEIGHTS.DAT. The format for the WEIGHTS.DAT file is described fully in the package specification file RATING.ADA. The weights assigned to each of the tests may be modified by changing this file.

The RATING package specification file should be read for a full description of the operation of this procedure, and for a description of the actions to perform to complete the execution of the test suite. Examining the compiler.LIS file may indicate that further actions need be taken for some of the tests. These actions may then be taken, the tests rerun, and @PERFORM RATING be repeated until the results are valid. When this happens, the execution of the test suite has been completed. The results of the test suite will be contained in all of the T?????.OUT files and summarized in the compiler.LIS file.

6. TEST FILE EXECUTION

This section will describe the test file format and what happens when a test is executed using the @PERFORM T????? command. The largest components of the support software are the PARSE and EXPAND packages. A description of their operations will be included here also. If a more detailed description of any of the procedures described in this section is needed, the package specification file for the procedure may be examined. The package specifications in the support software are fully commented.

When PERFORM.COM executes a test such as T020401 the following steps occur:

1. The file T020401.TST is copied to the home directory.
2. The PARSE procedure is called with T020401 as an argument.

Each test file must have the following format:

```
-- Test Number, e.g., T020401
--
-- A reiteration of the requirement in chapter 2, section 4.1
--
-- Method:
--
-- Test method description
--
Test code sections with embedded PARSE and EXPAND meta symbols.
```

A complete description of the meta symbol syntax is given in the package specification files PARSE.ADA and EXPAND.ADA. The meta symbols recognized by the PARSE procedure are:

```
--* BEGIN Compiler_1 Compiler_2 ...
--* END
--* COMPILE File_Name Option_1 Option_2 ...
--* FORTRAN File_Name
--* COMPARE Option File_Prefix
--* EXECUTE Procedure_Name
--* NEW_LIBRARY
```

When a file is being parsed, all text between the two commands --* BEGIN and --* END is simply ignored if the current compiler is not one of the compilers specified in the --* BEGIN compiler list. If the begin command does not contain any compiler names, then all text in the file until the next --* END statement will be ignored for all compilers. This construct allows a file to contain code for more than one compiler at a time. When adding a new compiler to the test suite domain, all tests containing such a construct should be examined to

determine if a section for the new compiler needs to be inserted into the test. For an example of this construct, see test T060301.TST.

The COMPILE, FORTRAN, and COMPARE commands all cause an auxiliary file to be produced. Test code from that point until the next COMPILE or FORTRAN command or end of file is written to the given file name. This command also causes a line to be written to the script file that will causes the specified file to be compiled using the given options, if any. In the case of the COMPARE function, which is only used by the T000000.TST test, the same code after the COMPILE command is duplicated as many times as there are COMPARE commands. But each file is then compiled using the different compiler Options. The possible compiler options are contained in the package NAMES.

The EXECUTE command causes the specified procedure name to be executed. The name specified must be an Ada or FORTRAN procedure that is found in the test code following the statement. This will causes a command to execute the appropriate .EXE file to be written to the script file.

The NEW_LIBRARY command is used to purge the current library. This command is needed for some tests that test library capacities. It is also called routinely to make sure that the library capacity does not become exceeded because many tests are being executed. This command may be placed before a COMPILE command or at the end of the file. When this command is called, the appropriate operating system primitives are written to the script file to cause the desired action. Also, the current state of the library is set to Uninitialized. When the next Ada code segment is compiled, the library status will be examined. When a library status of Uninitialized is found, a new library is automatically created and the state is set to Initialized. In addition, if a WITH statement in the current code segment is found for one of the support software packages, and the packages have not yet been compiled in the new library, then they will be automatically recompiled.

There are three meta symbols defined below that are recognized by the EXPAND procedure. There is one special case in which the PARSE procedure will also recognize these symbols. This is when a --* COMPILE

or --* FORTRAN statement is found directly after an unnested EXPAND procedure LOOP statement. This would look something like this:

```
--! LOOP 5 [1]
--* COMPILE TEMP NO_OPTIMIZE
Some large code segment
--! END [1]
```

This would be treated as:

```
--* COMPILE TEMP1 NO_OPTIMIZE
--! LOOP 1 START 1 [1]
Some large code segment
--! END [1]

--* COMPILE TEMP2 NO_OPTIMIZE
--! LOOP 1 START 2 [1]
Some large code segment
--! END [1]

--* COMPILE TEMP3 NO_OPTIMIZE
--! LOOP 1 START 3 [1]
Some large code segment
--! END [1]

--* COMPILE TEMP4 NO_OPTIMIZE
--! LOOP 1 START 4 [1]
Some large code segment
--! END [1]

--* COMPILE TEMP5 NO_OPTIMIZE
--! LOOP 1 START 5 [1]
Some large code segment
--! END [1]
```

In other words, the loop statement would be parsed according to the loop definition given below. Then the compile statement will be duplicated the specified loop number of times. Therefore, if the iteration count of the loop was five, then five files will be created and compiled. The construct was needed to avoid some of the file limitations of some compilers. The code inside the five separate files would be identical to the code created for the one big file if the --* COMPILE statement was placed before, instead of after, the --! LOOP construct. The only difference is that the code has been split into five files. For an example of this construct, see test T030103.TST.

If any EXPAND meta symbols are found embedded in either the Ada or FORTRAN test code segments, then the code segments will first be written to a file File_Name.EXP. Otherwise the appropriate File_Name.ADA or File_Name.FOR files will be created. If the .EXP

file has been created, then commands will be sent to the script file to EXPAND the .EXP file to either a .ADA or .FOR file directly before the .ADA or .FOR file is compiled.

The capability to expand files was created to help test several of the repetitive requirements. It allows some tests requiring thousands of lines of code to be compactly stored in a few lines when not in use. The EXPAND procedure takes as input a file containing EXPAND meta symbols. The commands recognized by the EXPAND procedure are:

```
--! EQUATE symbol IS expression
--! LOOP x STEP y START z [n]
--! END [n]
```

The complete syntax for these statements is described in the file EXPAND_.ADA. Many of the tests contain these constructs. The three reserved words LOOP, STEP, and START in the loop statement may be placed in any order. In addition, if any of them are missing a default of 1 is assumed. A symbol may be equated to a value, e.g.,

```
--! EQUATE Size IS 10 * 2 / Another_Symbol
```

The values for x, y, and z may be numbers or symbols or expression. The [n] value indicates the level of the loop. This number may be from 1 .. 9. Loops may be nested up to nine levels. The LOOP statement and its corresponding END statement must match up and have the correct loop level number.

When a file is expanded, the code between a LOOP and END statement is replicated x times. The implicit loop counter begins at z and is incremented by y on each iteration. The value of the implicit loop counter may be used inside the loop to change the semantic meaning for the code fragment being replicated on each iteration. The value of the counter is accessed by using [n], [n-i], [n+i], where i is an integer offset value. If an offset value is specified, then this value is added to the implicit loop counter before being printed.

```

--! START 10 LOOP 3 STEP 2 [1]
  PROCEDURE Test_11 IS
  BEGIN
    Item( [1-2] ) := Item( [1+1] );
  END Test_11;

--! END [1]

```

This fragment in the .EXP file would be replace in the .ADA file by:

```

PROCEDURE Test_10 IS
BEGIN
  Item( 8 ) := Item( 11 );
END Test_10;

PROCEDURE Test_12 IS
BEGIN
  Item( 10 ) := Item( 13 );
END Test_12;

PROCEDURE Test_14 IS
BEGIN
  Item( 12 ) := Item( 15 );
END Test_14;

```

3. After the PARSE procedure has been called with T020401 as an argument, a script file T020401.SCR will have been created. In addition, auxiliary .ADA, .FOR, or .EXP files will have been created from the code segments in the test separated by COMPILE and FORTRAN commands. The .EXP files will be EXPANDED into .ADA or .FOR files before being compiled by command written to the script file. The script file contains a delete file command for each of the temporary files created during the execution of a test.
4. PERFORM.COM routes all further output to the T020401.OUT file.
5. The T020401.SCR file is opened by PERFORM.COM.
6. If End_Of_File(T020401.SCR) THEN GO TO Step 10.
7. A line is read from the T020401.SCR File.
8. The line is processed. The allowed commands are as follows:
 PRINT, DELETE, COMPILE, FORTRAN, LINK, LINK_FORTRAN, EXECUTE,
 LIST, EXPAND, STORE_TIME, COMPUTE_RATE, CODE_SIZE, COUNT,
 REMOVE_LIBRARY, and CREATE_LIBRARY. A description of these
 commands may be found in PERFORM.COM, SUPPORT.ADA, NAMES_.ADA,
 and SCRIPT_.ADA.

9. The specified operating system primitive is executed, control returns to Step 6.
10. Close T020401.SCR, delete T020401.SCR, and delete T020401.TST.
11. Close the input stream to T020401.OUT.
12. Execution ends.

7. SUPPORT SOFTWARE PACKAGE DESCRIPTIONS

- TWINE** This is a string manipulation package. Dynamic string variables and lists are defined in this package and used throughout the other packages for building tables and doing general string manipulations. The name Twine was chosen because it is only five letters long and is a synonym for String.
- PQAC_IO** This package was created to provide a central Input/Output mechanism for the entire system, allowing input and output to be modified without editing every other package.
- NAMES** This specification contains the enumeration definitions of several of the features of the system including the possible operating system primitives and file types.
- TABLES** This specification contains the compiler and operating system dependent features of the system. Information about each compiler is saved in these tables.
- COMMON** Contains the system database and utilities used throughout the system. It controls the access to the dependent information in the TABLES package. It performs such actions as building file names and keeping track of the state of the Ada library.
- SYNTAX** This package contains utilities used by the PARSE and EXPAND programs. The syntax of the meta symbols used by these two utilities is controlled here.
- SCRIPT** This package is used by the PARSE program. It controls how the script file built by the PARSE program gets developed and printed out.
- TIMES** This package contains procedures for timing events used by both the support software and some of the test procedures.
- RESULT** This package is used by the test procedures to record their success or failure. This package also contains subprograms to allow the test procedures to display messages without Text_IO.
- COMPARE** This package contains a procedure to examine the results of running test T000000. This test contains identical versions

of Ada and FORTRAN code. This test is the first test run, and creates statistics about the performance of the compilers that are used by several other tests.

COUNT This package contains a procedure that counts the number of Ada source lines in a file.

EXPAND This package contains a procedure that produces code from templates with embedded meta symbols. These meta symbols inform the procedure to duplicate code using a loop syntax that may be nested. This allows very large code bodies from very small templates to be generated.

PARSE This package contains a procedure for parsing the test files. Meta symbols in the test files can be used to create multiple files for compilation. Information such as compilation unit names are also passed using these meta symbols. When a test file is parsed, one or more code files are created, and a script file is produced. This script file is used by an operating system interpreter to perform the actions required by the test.

RATING This package contains a procedure to be used after all the tests have been run. It uses output from the RESULT package to analyze and produce results. Weights assigned for each of the tests are also input to this procedure. This tool will allow the user to input the results of those tests that required manual intervention. It automatically has access to the results of those tests that did not require manual intervention. A compiler rating will be produced once all the data has been collected.

SUPPORT This procedure is the driver for all of the utility programs in the system. It provides access to the COUNT, PARSE, EXPAND, and RATING programs in addition to several of the timing procedures contained in the TIMES package.

8. EXAMPLE TEST WEIGHTING FILE

The following pages contain an example data file used for storing the weights used by the rating program. The format of this file is discussed in detail in the package specification file `RATING_.ADA`.

The first field contains the test number. There must be one line in the file for each of the 150 tests. The character after the test number may be either an 'M' or ' '. If 'M', the test is a minimal test as defined in the Definition. The third field contains the test weight. This is the weight the test will contribute if it passes 100%. The last column contains a percentage cutoff value between 0 and 100. Tests that pass at a percentage less than this cutoff value will be awarded 0 points.

Source File: WEIGHTS.DAT

T000000	0	100
T010100	0	100
T020100	0	100
T020200	0	100
T020300	0	100
T020401M	10	50
T020402M	10	50
T020403	10	50
T020501M	10	50
T020502M	10	50
T030101	2	0
T030102	2	0
T030103	2	0
T030104	2	0
T030105	1	100
T030106	1	100
T030201	1	100
T030202	2	0
T030203	2	0
T030204	2	0
T030205	2	0
T030206	2	0
T030207	1	100
T030208	1	100
T030209	1	100
T030301	1	100
T030302	1	100
T030303	1	100
T030304	1	100
T030305	1	100
T030306	1	100
T030307	1	100
T030308	1	100
T030309	1	100
T030310	1	100
T030311	2	100
T030401	1	100
T030402	2	0
T030403	1	100
T030404	1	100
T030405	1	100
T030406	1	100
T030407	1	100
T030408	1	100
T030501	2	100
T030502	2	100
T030601	2	0
T030602	2	0
T030701	2	0
T030702	1	100
T030703	1	100
T030704	2	100
T030705	1	100
T030706	1	100
T030707	1	100
T030708	2	100
T030709	2	100
T030801	1	100
T030802	1	100
T030803	1	100
T030804	2	100
T040101	10	100
T040102	10	100
T040103	4	0
T040104	4	0
T040105	2	100
T040106	2	100
T040201	10	100
T040202	10	100
T040203	5	100
T040204	5	100
T040205	5	100
T040206	5	100
T040207	4	0

Source File: WEIGHTS.DAT

T040208	1	100
T040209	1	100
T040301	2	0
T040302	2	100
T040303	2	0
T040304	2	100
T040305	2	100
T050101	2	100
T050102	2	100
T050103	6	100
T050104	6	100
T050201	5	100
T050202	5	100
T050203	4	100
T050204	4	100
T050300	10	100
T060100	4	100
T060201	4	100
T060202	3	0
T060203	4	100
T060301	4	100
T060302	4	100
T060303	4	100
T060304	4	100
T060305	6	0
T060306	2	100
T060307	4	100
T060308	4	100
T060309	4	100
T060310	4	100
T060401	8	100
T060402	0	100
T060403	2	100
T060404	4	100
T060501	4	100
T060502	2	100
T060503	1	100
T060504	4	100
T060505	2	100
T060506	2	100
T060601	2	100
T060602	2	100
T060603	2	100
T060701	2	100
T060702	2	100
T060703	2	100
T060801	2	100
T060802	2	100
T060900	4	0
T061001	2	100
T061002	4	0
T061003	4	0
T061004	2	100
T061101	4	100
T061102	2	100
T061201	2	100
T061202	2	100
T061203	1	100
T061204	2	100
T061205	1	100
T061206	1	100
T061207	1	100
T061208	1	100
T070100M	10	100
T070200M	8	100
T070300	10	100
T070400	8	100
T070500M	8	100
T080100M	8	100
T080200M	10	100
T080300M	10	100
T080400M	10	100
T080500M	8	100
T080600M	8	100

Source File: WEIGHTS.DAT

T080700 8 100
T080800M 8 100

9. OPERATING SYSTEM COMMAND INTERPRETER

The next few pages contain a listing of the PERFORM.COM file. This file is the command interpreter used for the DEC VAX and Telesoft Ada compiler evaluations. The file is written in VAX VMS DCL language. This file must be modified in order to apply the test suite to a new compiler.

```

$ !
$ !
$ !           The Aerospace Corporation
$ !
$ !           Production Quality Ada Compiler Test Suite Support Software
$ !
$ !
$ !           Author:  BAP
$ !           Date:    10/01/88
$ !           File:    Perform.Com
$ !           Component: VAX VMS Command Procedure Perform
$ !           Description: Operating System Primitives Interpreter
$ !           Parameters: P1 = Test Number, e.g. T010101, or SETUP, or RATING
$ !
$ !
$ ! If P1 = "SETUP" then a directory for the current working Ada library
$ ! is created if it does not exist. Any files in the directory are deleted.
$ ! Then the Ada procedure SUPPORT is called with an argument of "SET_UP".
$ ! This initializes the state of the test suite. Execution Ends.
$ !
$ ! If P1 = "RATING" then a report of the results of the test suite will be
$ ! created. A file "WEIGHTS.DAT" must exist containing the weighting scheme
$ ! to be used by the report. The current result file, e.g. Compiler_Name.DAT
$ ! will also be read. The report will be written to Compiler_Name.LIS.
$ ! Execution Ends.
$ !
$ ! If P1 = Test Number, e.g. T010101, then the following steps occur:
$ !
$ !     1. T010101.TST is copied from the test directory to the home directory
$ !     2. The output stream is directed to the file T010101.OUT
$ !     3. Ada procedure SUPPORT is called with arguments "PARSE T010101"
$ !     4. Parsing T010101.TST creates a script file T010101.SCR and other files.
$ !     5. The T010101.SCR file is opened.
$ !     6. If end of file T010101.SCR then go to step 10.
$ !     7. A line is read from T010101.SCR
$ !     8. The line is interpreted, the allowed commands are listed below
$ !     9. Go to Step 6.
$ !    10. Close T010101.SCR
$ !    11. Delete T010101.SCR
$ !    12. Delete T010101.TST
$ !    13. Close the input stream to T010101.OUT
$ !    14. Test is finished
$ !
$ ! Allowed Commands: Defined in Ada Package Names.OS_Primitives
$ !
$ ! PRINT      Args  -- Send args to output stream
$ ! DELETE     Args  -- Delete args file
$ ! COMPILE    Args  -- Args contains compiler invocation string and file
$ ! FORTRAN    Args  -- Invoke the FORTRAN compiler with optimization on
$ ! LINK       Args  -- Link the specified args using the Ada library
$ ! LINK_FORTRAN Args  -- Link the specified args FORTRAN program
$ ! EXECUTE    Args  -- Run the specified args executable code file
$ ! LIST       Args  -- Send a listing of the file args to the output stream
$ ! EXPAND     Args  -- Call SUPPORT with parameters "EXPAND args"
$ ! STORE_TIME Args  -- Call SUPPORT with parameters "STORE_TIME args"
$ ! COMPUTE_RATE Args  -- Call SUPPORT with parameters "COMPUTE_RATE args"
$ ! CODE_SIZE  Args  -- Call SUPPORT with parameters "CODE_SIZE args"
$ ! COUNT      Args  -- Call SUPPORT with parameters "COUNT args"
$ ! REMOVE_LIBRARY -- Delete all files in the working Ada library directory
$ ! CREATE_LIBRARY -- Create a new working Ada library
$ !
$ !
$ ! ON Control_Y THEN GOTO Stopped
$ ! ON Warning THEN GOTO AB_End
$ ! Status = "OK"
$ !
$ ! Directory Information:
$ !
$ ! Assign/NoLog PUBLIC:[U18579.TMPLIB] Current$Lib
$ ! ! Working Ada Library
$ ! Assign/NoLog PUBLIC:[U18579.REPORTS.PQACS] Home$Lib
$ ! ! Directory where results and status files are kept
$ ! Assign/NoLog PUBLIC:[U18579.EXECUTE] Execute$Lib
$ ! ! Directory where SUPPORT.EXE resides
$ ! Assign/NoLog PUBLIC:[U18579.REPORTS.PQACS.TESTS] Tests$Lib

```

```

$ ! Directory containing test files, i.e. T010100.TST
$ Assign/NoLog PUBLIC:[U18579.REPORTS.PQACS.OUTPUT] Output$Lib
$ ! Directory for test execution results, i.e. T010100.OUT
$ Assign/NoLog PUBLIC:[U18579.REPORTS.PQACS.SOURCE] Source$Lib
$ ! Directory containing support software source, i.e. COUNT.ADA
$ !
$ ! Test for "SETUP" or "RATING" or "Test Number" argument in P1
$ !
$ SET DEFAULT Home$Lib
$ IF P1 .EQS. "SETUP" THEN GOTO Setup
$ IF F$Search( "STATE.DAT" ) .EQS. "" THEN GOTO Setup
$ !
$ ! Test$Compiler is set to the current compiler.
$ ! Possible Test Compilers: See Package Tables.Compiler_Domain
$ ! Make sure there is an option for each possible Test$Compiler
$ ! whenever the contents of Test$Compiler are checked in this file.
$ !
$ OPEN/READ IN STATE.DAT
$ READ IN Test$Compiler
$ CLOSE IN
$ IF F$Extract( 0, 1, P1 ) .EQS. "T" THEN GOTO Run_Test
$ IF P1 .EQS. "RATING" THEN GOTO Rating
$ WRITE SYS$OUTPUT "Undefined Action: ", P1
$ EXIT
$ !
$ ! Setup: Initializes the PQAC Test Suite.
$ !
$Setup:
$ IF F$Parse( "Current$Lib" ) .EQS. "" THEN Create/Directory Current$Lib
$ IF F$Search( "Current$Lib:*.x;*" ) .NES. "" THEN DELETE Current$Lib:*.x;*
$ Action = "SET_UP"
$ Record = ""
$ ASSIGN/USER SYS$COMMAND SYS$INPUT
$ GOSUB Run_Program
$ EXIT
$ !
$ ! Rating: Read Weights and Compiler results and produce a report.
$ !
$Rating:
$ Action = "RATING"
$ Record = "WEIGHTS " + Test$Compiler
$ GOSUB Run_Program
$ EXIT
$ !
$ ! PARSE parses the .TST file and creates a script file in a .SCR file.
$ !
$Run_Test:
$ COPY Tests$Lib:'P1'.TST 'P1'.TST
$ ASSIGN/NoLog Output$Lib:'P1'.OUT SYS$OUTPUT
$ Action = "PARSE"
$ Record = P1
$ GOSUB Run_Program
$ OPEN/READ IN 'P1'.SCR
$ !
$ ! Read Loop
$ !
$Continue:
$ READ/End_Of_File = Finished IN Record
$ IF P2 .NES. "" THEN WRITE SYS$OUTPUT Record
$ Space = F$LOCATE( " ", Record ) ! Temporary Variable
$ Action = F$EXTRACT( 0, Space, Record ) ! Action Command Name
$ Record = F$EXTRACT( Space + 1, 80, Record ) ! Command Arguments
$ !
$ IF Action .EQS. "PRINT" THEN GOTO Print
$ IF Action .EQS. "DELETE" THEN GOTO Delete
$ IF Status .NES. "OK" THEN GOTO Continue
$ IF Action .EQS. "COMPILE" THEN GOTO Compile
$ IF Action .EQS. "FORTRAN" THEN GOTO Fortran
$ IF Action .EQS. "LINK" THEN GOTO Link
$ IF Action .EQS. "LINK_FORTRAN" THEN GOTO Link_Fortran
$ IF Action .EQS. "EXECUTE" THEN GOTO Execute
$ IF Action .EQS. "LIST" THEN GOTO List
$ IF Action .EQS. "EXPAND" THEN GOTO Program
$ IF Action .EQS. "STORE_TIME" THEN GOTO Program

```

Source File: PERFORM.COM

```
$ IF Action .EQS. "COMPUTE_RATE" THEN GOTO Program
$ IF Action .EQS. "CODE_SIZE" THEN GOTO Program
$ IF Action .EQS. "COUNT" THEN GOTO Program
$ IF Action .EQS. "REMOVE_LIBRARY" THEN GOTO Remove_Library
$ IF Action .EQS. "CREATE_LIBRARY" THEN GOTO Create_Library
$ WRITE SYS$OUTPUT "Undefined Action: ", Action, Record
$ GOTO Continue
$ !
$ ! Subroutine Run_Program: Calls Ada procedure SUPPORT with arguments
$ !
$Run_Program:
$ OPEN/WRITE OUT PARAM.DAT ! SUPPORT.EXE reads arguments from PARAM.DAT
$ WRITE OUT Action, " ", Record
$ CLOSE OUT
$ RUN Execute$Lib:Support
$ DELETE Param.Dat;*
$ RETURN
$ !
$ ! Program <Parameters>
$ !
$Program:
$ GOSUB Run_Program
$ GOTO Continue
$ !
$ ! Print ...
$ !
$Print:
$ WRITE SYS$OUTPUT Record
$ GOTO Continue
$ !
$ ! Delete <File Name>
$ !
$Delete:
$ IF F$Search( Record ) .NES. "" THEN DELETE 'Record';*
$ GOTO Continue
$ !
$ ! List <File Name>
$ !
$List:
$ TYPE 'Record'
$ GOTO Continue
$ !
$ ! Compile <Compiler Command String> <File Name>
$ !
$Compile:
$ Name = F$Extract( F$Locate( " ", Record ) + 1, 80, Record )
$ IF F$Search( "Source$Lib:" + Name ) .NES. "" THEN SET DEFAULT Source$Lib
$ ! Source$Lib contains support software needed by the tests. If the file
$ ! to be compiled is one of these then set default to source library.
$ 'Record'
$ SET DEFAULT Home$Lib
$ GOTO Continue
$ !
$ ! FORTRAN
$ !
$FORTRAN:
$ FOR/NOLIST/SHOW=NONE/OPTIMIZE 'Record'
$ GOTO Continue
$ !
$ ! Link <Compilation Unit Name>
$ !
$Link:
$ IF Test$Compiler .EQS. "DEC VAX V1_4" THEN GOTO LINK_DEC_VAX_V1_4
$ IF Test$Compiler .EQS. "TELEGEN2_V3_15" THEN TSADA/BIND 'Record'
$ GOTO Continue
$Link_DEC_VAX_V1_4:
$ ACS LINK 'Record'/COMMAND=XXXXXX.COM
$ @xxxxxx
$ DELETE XXXXXX.COM;*
$ GOTO Continue
$ !
$ ! Link_Fortran <Compilation Unit Name>
$ !
$Link_Fortran:
```

Source File: PERFORM.COM

```
$ LINK 'Record'
$ IF F$Search( Record + ".MAP" ) .NES. "" THEN DELETE 'Record'.MAP;*
$ GOTO Continue
$ !
$ ! Execute <Compilation Unit Name>
$ !
$Execute:
$ RUN 'Record'
$ GOTO Continue
$ !
$ ! Create_Library
$ !
$Create_Library:
$ IF F$Search( "Current$Lib:*.x;*") .NES. "" THEN DELETE Current$Lib:*.x;*
$ IF Test$Compiler.EQS."DEC_VAX_V1_4" THEN GOTO New_DEC_VAX_V1_4
$ IF Test$Compiler.EQS."TELEGEN2_V3_15" THEN GOTO New_TELEGEN2_V3_15
$New_DEC_VAX_V1_4:
$ ACS CREATE LIBRARY/NoLog Current$Lib
$ ACS SET LIBRARY/NoLog Current$Lib
$ GOTO Continue
$New_TELEGEN2_V3_15:
$ OPEN/WRITE OUT LIBLST.ALB
$ WRITE OUT "NAME: Current$Lib:ADALIB"
$ WRITE OUT "NAME: TSADA$DIR:TSADARTL"
$ CLOSE OUT
$ IF F$Search( "ADALIB.OLB" ) .EQS. "" THEN TSADA/Create ADALIB
$ COPY ADALIB.*;1 Current$Lib
$ GOTO Continue
$ !
$ ! Remove_Library
$ !
$Remove_Library:
$ IF F$Search( "Current$Lib:*.x;*") .NES. "" THEN DELETE Current$Lib:*.x;*
$ IF Test$Compiler .EQS. "TELEGEN2_V3_15" THEN DELETE LIBLST.ALB;*
$ GOTO Continue
$ !
$ ! Abnormal Termination
$ !
$AB_End:
$ Status = "NOTOK"
$ SET DEFAULT Home$Lib
$ ON Warning THEN GOTO Finished
$ GOTO Continue
$ !
$ ! Stopped Execution with Control Y
$ !
$Stopped:
$ Status = "NOTOK"
$ SET DEFAULT Home$Lib
$ CLOSE IN
$ DEASSIGN SYS$OUTPUT
$ EXIT
$ !
$ ! Normal Termination
$ !
$Finished:
$ CLOSE IN
$ DELETE 'P1'.TST;*
$ DELETE 'P1'.SCR;*
$ DEASSIGN SYS$OUTPUT
$ EXIT
```


10. PQAC SUPPORT SOFTWARE PACKAGES (Alphabetical)

The following pages contain a listing of the PQAC Ada support software packages. See Section 4 for a description of the compilation order. A brief description of the functionality of each of these packages is given in Section 7 of this volume. The files are listed in the following alphabetical order:

COMMON_.ADA
COMMON.ADA
COMPARE_.ADA
COMPARE.ADA
COUNT_.ADA
COUNT.ADA
EXPAND_.ADA
EXPAND.ADA
NAMES_.ADA
PARSE_.ADA
PARSE.ADA
PQAC_IO_.ADA
PQAC_IO.ADA
RATING_.ADA
RATING.ADA
RESULT_.ADA
RESULT.ADA
SCRIPT_.ADA
SCRIPT.ADA
SUPPORT.ADA
SYNTAX_.ADA
SYNTAX.ADA
TABLES_.ADA
TIMES_.ADA
TIMES.ADA
TWINE_.ADA
TWINE.ADA

Source File: COMMON_.ADA

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:  BAP
--       Date:   10/01/88
--       File:   Common_.Ada
--       Component: Package Specification Common
--       Description: This package provides the interface to the compiler and
--                   host dependant package Tables.
--
--                   Subprograms dependent on the compiler and host environment
--                   are included here. File names are built using this package.
--                   Actual arguments for compiler options and operating system
--                   actions are returned from this package.
--
--                   This package is also used to keep track of the current
--                   state of the suport software and Ada library. Several
--                   of the tests examine library capacities, so the capability
--                   of compiling with an initially empty library is needed.
--
--
```

WITH Names; -- Enumeration Declarations

PACKAGE Common IS

```
TYPE Library_Status IS
( UnInitialized, -- Library has not been created or has been removed.
  Initialized,   -- Library exists, support software not compiled.
  Support_Compiled ); -- Library exists, support software is compiled.
```

```
TYPE System_Attributes IS
( Current_Test,      -- Current Test Name (e.g. "T010100" )
  Current_Compiler,  -- Current Compiler Name
  Host_Machine,      -- Host Machine Name
  Target_Machine,    -- Target Machine Name
  Host_Banner,        -- Description of Host Machine, MIPS
  Target_Banner,      -- Description of Target Machine, MIPS
  Base_Compiler_Option ); -- Command for invoking the compiler without
-- any of the special compiler options.
```

Undefined_Error : EXCEPTION;

```
PROCEDURE Initialize;
-- Called by the main Support procedure before parsing each test.
-- The current state of the test suite is read from a file.
-- Undefined_Error will be raised if the status file cannot be found
-- or the data in it is unreadable.
```

```
PROCEDURE Shut_Down;
-- Called by the main Support procedure after parsing each test.
-- The current state of the test suite is written to a file.
```

```
PROCEDURE Create_Status_File;
-- This procedure must be called initially before any tests have
-- been performed. It queries the user as to the current configuration
-- of compiler and host. This information is then written to the
-- status file for use in parsing the tests. The Initialize procedure
-- above will not work unless this has been called once.
```

```
FUNCTION Host_Rated_MIPS RETURN Float;
-- Returns the Rated MIPS of the Host computer.
```

```
FUNCTION Target_Rated_MIPS RETURN Float;
-- Returns the Rated MIPS of the Target computer.
```

```
FUNCTION Is_Support_Package( Name : String ) RETURN Boolean;
-- Each of the tests requires that a subset of the support software
```

```
-- must be compiled. These package are used by the tests to record
-- information about the test, or to perform timings or sizings.
-- The function returns True if the supplied name is one of these
-- required support software packages.
```

```
FUNCTION Support_Size RETURN Natural;
-- Returns the number of package in the support software subset
-- required to run each of the tests.
```

```
FUNCTION Support_Package( Number : Positive ) RETURN String;
-- Returns the file name of the Nth support software package required
-- to run each of the tests. If the state of the library is not
-- Support_Compiled then this function will be used to get the file
-- names of all the packages that need to be compiled before the
-- test may be compiled.
```

```
PROCEDURE Set_Current_Test( Test : String );
-- Sets the current test name, e.g. "T010100". This value may be
-- retrieved by calling Common.Image( Common.Current_Test ).
```

```
FUNCTION Is_Current_Compiler( Name : String ) RETURN Boolean;
-- Returns true if the given Name is the current compiler. This
-- is used for determining whether to ignore code between
-- " --X BEGIN Compiler_Name" and " --X END".
-- Undefined_Error will be raised if the given Name is not one of
-- the possible compilers.
```

```
FUNCTION Option_Of( Option : String ) RETURN Names.Compiler_Options;
-- Converts the given Option to the enumeration type.
-- Undefined_Error will be raised if the given Option is not one of
-- the possible options.
```

```
FUNCTION Image( Option : Names.Compiler_Options ) RETURN String;
-- Returns the option string for the standardized enumeration option.
-- The image of these options will be different for different compilers.
```

```
FUNCTION Image( Special_File : Names.Transfer_Files ) RETURN String;
-- Several files are used for transferring information between
-- the tests and the support software. Standard file names are used
-- to hold time values, size values, test results, comparison results,
-- and the state of the support software. This function returns the
-- file name of the type specified.
```

```
FUNCTION Image( Attribute : System_Attributes ) RETURN String;
-- Returns the attribute image as defined in the System_Attributes
-- enumeration declaration given above.
```

```
FUNCTION Image( Primitive : Names.OS_Primitives ) RETURN String;
-- Returns a string of the representing the standard defined primitives
-- in the enumeration type Names.OS_Primitives. Each line in the
-- script file produced by parsing a test will begin with one of
-- these strings.
```

```
FUNCTION Build_Name( Prefix : String; Suffix : Names.File_Category )
RETURN String;
-- Returns a correct file name for the given file Prefix and standard
-- defined file type suffix. The syntax for file names may be different
-- for different host machines.
```

```
FUNCTION Library_State RETURN Library_Status;
-- Returns the state of the library as defined in the Library_Status
-- enumeration declaration given above.
```

```
PROCEDURE Set_Library_State( State : Library_Status );
-- Sets the current state of the library to the given value.
```

```
FUNCTION Library_Test_Count RETURN Natural;
-- Returns the number of tests performed since the creation or
-- reinitialization of the working Ada library.
```

END Common;

Source File: COMMON.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Common.Ada
--       Component: Package Body Common
--       Description: This package provides the interface to the compiler and
--                   host dependant package Tables.
--
```

```
WITH Twine;      -- String Manipulation Package
WITH Tables;     -- Compiler and Host Specific Information
WITH PQAC_IO;    -- Centralized Input and Output Package
```

PACKAGE BODY Common IS

```
-- Format for the support software status file:
--
--       Data Description      --> Example Data
--
-- Line 1, Compiler Name      --> "DEC_VAX_V1_4"
-- Line 2, Current Test       --> "T000000"
-- Line 3, Test Count         --> " 0"
-- Line 4, Library Status     --> "UNINITIALIZED"
```

```
TYPE Current_State_Record IS RECORD
    Current_Compiler   : Tables.Compiler_Domain;
    Current_Test       : Twine.Series;
    Library_Test_Count : Natural := 0;
    State_Of_Library   : Library_Status;
END RECORD;
```

```
Current_State : Current_State_Record;
Previous_State : Current_State_Record;
```

```
FUNCTION Current_Compiler RETURN Tables.Compiler_Domain IS
BEGIN
    RETURN Current_State.Current_Compiler;
END Current_Compiler;
```

```
FUNCTION Current_Host RETURN Tables.Host_Architecture IS
BEGIN
    RETURN Tables.Compiler_Table( Current_Compiler ).Host;
END Current_Host;
```

```
FUNCTION Current_Target RETURN Tables.Target_Architecture IS
BEGIN
    RETURN Tables.Compiler_Table( Current_Compiler ).Target;
END Current_Target;
```

```
FUNCTION Get_Current_Test RETURN String IS
BEGIN
    RETURN Twine.Image( Current_State.Current_Test );
END Get_Current_Test;
```

PROCEDURE Initialize IS

Source File: COMMON.ADA

```
File   : PQAC_IO.File_Type;
Buffer : Twine.Input_Buffer;
Last   : Natural := 0;

PROCEDURE Save_Current_Compiler( Name : String ) IS
BEGIN
    Current_State.Current_Compiler := Tables.Compiler_Domain'VALUE(Name);
EXCEPTION
    WHEN OTHERS =>
        PQAC_IO.Record_Error( "Unknown Compiler: " & Name );
        RAISE Undefined_Error;
END Save_Current_Compiler;

PROCEDURE Save_Current_Library( State : String ) IS
BEGIN
    Current_State.State_Of_Library := Library_Status'VALUE( State );
EXCEPTION
    WHEN OTHERS =>
        PQAC_IO.Record_Error( "Unknown Library Status: " & State );
        RAISE Undefined_Error;
END Save_Current_Library;

BEGIN
    PQAC_IO.Open_Input( File, Image( Names.PQAC_State ) );
    PQAC_IO.Get_Line( File, Buffer, Last );
    Save_Current_Compiler( Buffer( 1 .. Last ) );
    PQAC_IO.Get_Line( File, Buffer, Last );
    Current_State.Current_Test := Twine.Create( Buffer( 1 .. Last ) );
    PQAC_IO.Get_Line( File, Buffer, Last );
    Current_State.Library_Test_Count := Integer'VALUE( Buffer( 1 .. Last ) );
    PQAC_IO.Get_Line( File, Buffer, Last );
    Save_Current_Library( Buffer( 1 .. Last ) );
    PQAC_IO.Close( File );
    Previous_State := Current_State;
EXCEPTION
    WHEN OTHERS =>
        PQAC_IO.Record_Error
            ( "Error reading " & Image( Names.PQAC_State ) & " Status." );
        RAISE Undefined_Error;
END Initialize;

PROCEDURE Shut_Down IS
    File   : PQAC_IO.File_Type;
    Line   : Twine.Series;

    PROCEDURE Remove_File( Name : String ) IS
    BEGIN
        PQAC_IO.Delete_File( Name );
    EXCEPTION
        WHEN OTHERS => NULL; -- If it doesn't exist yet, that's OK.
    END Remove_File;

    BEGIN
        IF Previous_State = Current_State AND THEN
            Twine.Equal( Previous_State.Current_Test, Current_State.Current_Test )
        THEN
            RETURN; -- Nothing has changed, so don't bother writing out status.
        END IF;
        Remove_File( Image( Names.PQAC_State ) );
        PQAC_IO.Open_Output( File, Image( Names.PQAC_State ) );
        PQAC_IO.Put_Line
            ( File, Tables.Compiler_Domain'IMAGE( Current_State.Current_Compiler ) );
        PQAC_IO.Put_Line( File, Twine.Image( Current_State.Current_Test ) );
        PQAC_IO.Put_Line
            ( File, Integer'IMAGE( Current_State.Library_Test_Count ) );
        PQAC_IO.Put_Line
            ( File, Library_Status'IMAGE( Current_State.State_Of_Library ) );
        PQAC_IO.Close( File );
    END Shut_Down;
```

PROCEDURE Create_Status_File IS

```

FUNCTION Get_Response RETURN Tables.Compiler_Domain IS
    -- The user is queried for the current compiler name.
    -- The user is first presented with a list of possible compiler
    -- names, and is then prompted to type one in. The User
    -- will be prompted to type in names until one matches a
    -- given choice exactly.

    Compiler : Tables.Compiler_Domain;
    Buffer    : Twine.Input_Buffer;
    Last     : Natural := 0;

    FUNCTION Valid_Compiler( Name : String ) RETURN Boolean IS
    BEGIN
        Compiler := Tables.Compiler_Domain'VALUE( Name );
        RETURN True;
    EXCEPTION
        WHEN OTHERS => RETURN False;
    END Valid_Compiler;

BEGIN
    PQAC_IO.Put_Line( "" );
    PQAC_IO.Put_Line( "Possible Compilers:" );
    PQAC_IO.Put_Line( "" );
    FOR Index IN Tables.Compiler_Domain LOOP
        PQAC_IO.Put_Line( "  " & Tables.Compiler_Domain'IMAGE( Index ) );
    END LOOP;
    PQAC_IO.Put_Line( "" );
    LOOP
        PQAC_IO.Get_Line( "Enter Desired Compiler: ", Buffer, Last );
        EXIT WHEN Valid_Compiler( Buffer( 1 .. Last ) );
        PQAC_IO.Put_Line( "Unknown Compiler: Redo" );
    END LOOP;
    RETURN Compiler;
END Get_Response;

BEGIN
    Current_State.Current_Compiler := Get_Response;
    Current_State.Current_Test     := Twine.Create( "T000000" );
    Current_State.Library_Test_Count := 0;
    Current_State.State_Of_Library := UnInitialized;
    Shut_Down;
    PQAC_IO.Append
        ( Image( Names.Test_Result ),
          Image( Current_Compiler ) & " Ada Compiler" );
END Create_Status_File;

```

```

FUNCTION Host_Rated_MIPS RETURN Float IS
BEGIN
    RETURN Tables.Host_Table( Current_Host ).Rated_MIPS;
END Host_Rated_MIPS;

```

```

FUNCTION Target_Rated_MIPS RETURN Float IS
BEGIN
    RETURN Tables.Target_Table( Current_Target ).Rated_MIPS;
END Target_Rated_MIPS;

```

```

FUNCTION Is_Support_Package( Name : String ) RETURN Boolean IS
BEGIN
    FOR Index IN Tables.Support_Packages'RANGE LOOP
        IF Twine.Equal( Name, Tables.Support_Packages( Index ) ) THEN
            RETURN True;
        END IF;
    END LOOP;

```

Source File: COMMON.ADA

```
    RETURN False;
END Is_Support_Package;
```

```
FUNCTION Support_Package( Number : Positive ) RETURN String IS
BEGIN
    IF Number IN Tables.Support_Packages'RANGE THEN
        RETURN Twine.Image( Tables.Support_Packages( Number ) );
    ELSE
        RETURN "";
    END IF;
END Support_Package;
```

```
FUNCTION Support_Size RETURN Natural IS
BEGIN
    RETURN Tables.Support_Packages'LAST;
END Support_Size;
```

```
PROCEDURE Set_Current_Test( Test : String ) IS
BEGIN
    IF Twine.Length( Current_State.Current_Test ) = Test'LENGTH THEN
        Twine.Copy( Current_State.Current_Test, Test );
    ELSE
        Current_State.Current_Test := Twine.Create( Test );
    END IF;
    Current_State.Library_Test_Count := Current_State.Library_Test_Count + 1;
END Set_Current_Test;
```

```
FUNCTION Is_Current_Compiler( Name : String ) RETURN Boolean IS
    Current : Tables.Compiler_Domain;
BEGIN
    Current := Tables.Compiler_Domain'VALUE( Name );
    RETURN Tables."="( Current, Current_Compiler );
EXCEPTION
    WHEN OTHERS => RAISE Undefined_Error;
END Is_Current_Compiler;
```

```
FUNCTION Option_Of( Option : String ) RETURN Names.Compiler_Options IS
BEGIN
    RETURN Names.Compiler_Options'VALUE( Option );
EXCEPTION
    WHEN OTHERS => RAISE Undefined_Error;
END Option_Of;
```

```
FUNCTION Image( Option : Names.Compiler_Options ) RETURN String IS
BEGIN
    RETURN Twine.Image
        ( Tables.Compiler_Table( Current_Compiler ).Options( Option ) );
END Image;
```

```
FUNCTION Image( Special_File : Names.Transfer_Files ) RETURN String IS
BEGIN
    IF Names."="( Special_File, Names.Test_Result ) THEN
        RETURN Build_Name
            ( Tables.Compiler_Domain'IMAGE( Current_Compiler ),
              Tables.Special_Names( Special_File ).Kind );
    ELSE
        RETURN Build_Name
            ( Twine.Image( Tables.Special_Names( Special_File ).Name ),
              Tables.Special_Names( Special_File ).Kind );
    END IF;
```

Source File: COMMON.ADA

END Image;

```
FUNCTION Image( Attribute : System_Attributes ) RETURN String IS
BEGIN
  CASE Attribute IS
    WHEN Current_Test =>
      RETURN Get_Current_Test;
    WHEN Current_Compiler =>
      RETURN Twine.Image( Tables.Compiler_Table( Current_Compiler ).Name );
    WHEN Host_Machine =>
      RETURN Twine.Image( Tables.Host_Table( Current_Host ).Name );
    WHEN Target_Machine =>
      RETURN Twine.Image
        ( Tables.Target_Table( Current_Target ).Name );
    WHEN Host_Banner =>
      RETURN Image( Host_Machine ) & " Rated at " &
        Twine.Image( Host_Rated_MIPS, 5, 2 ) & " MIPS.";
    WHEN Target_Banner =>
      RETURN Image( Target_Machine ) & " Rated at " &
        Twine.Image( Target_Rated_MIPS, 5, 2 ) & " MIPS.";
    WHEN Base_Compiler_Option =>
      RETURN Twine.Image
        ( Tables.Compiler_Table( Current_Compiler ).Basic_Command );
  END CASE;
END Image;
```

```
FUNCTION Image( Primitive : Names.OS_Primitives ) RETURN String IS
BEGIN
  RETURN Names.OS_Primitives'IMAGE( Primitive );
END Image;
```

```
FUNCTION Build_Name( Prefix : String; Suffix : Names.File_Category )
  RETURN String IS
  Value : Twine.Series
    := Tables.Host_Table( Current_Host ).Suffix( Suffix );

  FUNCTION New_Name( Name : String; Last : String ) RETURN String IS
  BEGIN
    IF Last = Name( Name'LAST - Last'LENGTH + 1 .. Name'LAST ) THEN
      RETURN Name;
    ELSE
      RETURN Name & Last;
    END IF;
  END New_Name;

  BEGIN
    RETURN New_Name( Twine.Clip( Prefix ), Twine.Image( Value ) );
  END Build_Name;
```

```
FUNCTION Library_State RETURN Library_Status IS
BEGIN
  RETURN Current_State.State_Of_Library;
END Library_State;
```

```
PROCEDURE Set_Library_State( State : Library_Status ) IS
BEGIN
  Current_State.State_Of_Library := State;
END Set_Library_State;
```

```
FUNCTION Library_Test_Count RETURN Natural IS
BEGIN
```


Source File: COMMON.ADA

```
    RETURN Current_State.Library_Test_Count;  
END Library_Test_Count;
```

```
END Common;
```

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Compare_.Ada
--       Component: Package Specification Compare
--       Description:
--
--               This package is used by the tests in Chapter 2 that
--               require the comparison of an Ada compiler verses
--               an optimized FORTRAN compiler. ( This requirement
--               originally stated hand optimized assembly code but
--               has been modified to use FORTRAN. )
--
--               Test T000000 must be ran to create the data file containing
--               the results of the compilations. If this test has not been
--               ran then calling Percentage will raise the Undefined_Data
--               exception.
--
--               Test T000000 compiles and executes functionally identical
--               FORTRAN and Ada programs. Five compilations and executions
--               are made: one FORTRAN, an Ada compilation for each of the
--               four Compiler_Version options listed below. The Ada program
--               does not contain any WITH statements.
--
--               Calling Percentage causes a list of the the observed
--               results to be output to the test output stream in addition
--               to returning the percentage value.
--
```

PACKAGE Compare IS

```
TYPE Compiler_Version IS
  ( Optimize_Space, Optimize_Time, No_Optimize, Syntax_Only );
```

```
Undefined_Data : EXCEPTION;
```

```
FUNCTION Percentage
  ( Compiler_Version : Compiler_Version;
    Minimum Compile_Rate : Natural;
    Minimum_Size_Percent : Natural;
    Minimum_Time_Percent : Natural ) RETURN Natural;
--
--   If any of the minimum criteria are 0, then no minimum is required
--   for that statistic, i.e. it satisfies 100% of the criteria.
--   Undefined_Data will be raised if T000000 has not been executed.
--
--   Each time this function is called, the results are also printed out
--   to the test output stream.
--
--   For the given compiler_option and specified minimum values,
--   a pass percentage is returned based on these values and the
--   observed compilation results. The result returned will be
--   between 0 and 100 (percent). For example:
--
--       Observed Ada Optimize_Space
--       Compile Rate:    500 Lines/Minute/MIP
--       Code Size:      1200 Words
--       Execution Time:  20.0 Seconds
--
--       Observed FORTRAN
--       Compile Rate:    0 Lines/Minute/MIP -- N/A to FORTRAN
--       Code Size:      1000 Words
--       Execution Time:  25.0 Seconds
--
--       Combined Observed Results for Optimize_Space:
--       Compile Rate:    500 Lines/Minute/MIP
--       Size Percent:    120% --> 1200 Words / 1000 Words
```

```

--      Time Percent:      80% --> 20.0 Seconds / 25.0 Seconds
--
--      With this data, example results are:
--
--      Percentage( Optimize_Space, 500, 120, 80 ) = 100%
--      100% pass on compile rate ( Requires 500 >= Observed 500 )
--      100% pass on size percent ( Observed 120 >= Required 120 )
--      100% pass on time percent ( Observed 80 >= Required 80 )
--      ----
--      100% total pass ( 100% X 100% X 100% )
--
--      Percentage( Optimize_Space, 500, 100, 100 ) = 80%
--      100% pass on compile rate ( Required 500 >= Observed 500 )
--      100% pass on size percent ( Observed 120 >= Required 100 )
--      80% pass on time percent ( Observed 80 / Required 100 )
--      ----
--      80% total pass ( 100% X 100% X 80% )
--
--      Percentage( Optimize_Space, 250, 100, 160 ) = 25%
--      50% pass on compile rate ( Required 250 / Observed 500 )
--      100% pass on size percent ( Observed 120 >= Required 100 )
--      50% pass on time percent ( Observed 80 / Required 160 )
--      ----
--      25% total pass ( 50% X 100% X 50% )
--
--      Percentage( Optimize_Space, 0, 0, 100 ) = 80%
--      100% pass on compile rate ( Required 0 --> Not required )
--      100% pass on size percent ( Required 0 --> Not required )
--      80% pass on time percent ( Observed 80 / Required 100 )
--      ----
--      80% total pass ( 100% X 100% X 80% )
--

```

FUNCTION Result_File RETURN String;

```

--      Returns the name of the file used to store the compilation results.
--
--      Example format for the compare data file:
--
--      A: compiler version
--      B: lines/minute/MIP
--      C: hundredths of seconds execution time
--      D: size of executable file in machine words
--
--      A      B      C      D
--      Line 1: "OPTIMIZE_SPACE    401    450    16384"
--      Line 2: "OPTIMIZE_TIME     337    429    16384"
--      Line 3: "NO_OPTIMIZE       413    944    56320"
--      Line 4: "SYNTAX_ONLY      1230    939    56320"
--      Line 5: "FORTRAN           0     601    16384"

```

END Compare;

Source File: COMPARE.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Compare.Ada
--       Component: Package Body Compare
--       Description: Package for retrieving and manipulating stored compiler
--                   comparison data. ( See Specification Descriptions )
--
--
WITH Names;      -- Enumeration Declarations
WITH Result;     -- Records Test Results
WITH Common;     -- Interface to Compiler Specific Information and Status
WITH PQAC_IO;    -- Centralized Input and Output Package

PACKAGE BODY Compare IS

    -- Example format for the compare data file:
    --
    --   A: compiler version
    --   B: lines/minute/MIP
    --   C: hundreths of seconds execution time
    --   D: size of executable file in machine words
    --
    --
    --           A           B           C           D
    -- Line 1: "OPTIMIZE_SPACE      401      450      16384"
    -- Line 2: "OPTIMIZE_TIME       337      429      16384"
    -- Line 3: "NO_OPTIMIZE         413      944      56320"
    -- Line 4: "SYNTAX_ONLY        1230      939      56320"
    -- Line 5: "FORTRAN             0        601      16384"
    --
    --
    TYPE Metric_Record IS RECORD
        Compile_Speed : Natural := 0;
        Execute_Time  : Natural := 0;
        Execute_Size  : Natural := 0;
        Alias_Time    : Float   := 0.0;
    END RECORD;

    FORTRAN_Results : Metric_Record;
    Ada_Results      : ARRAY( Compiler_Version ) OF Metric_Record;
    Initialized      : Boolean := False;

    FUNCTION "&"( Text : String; Value : Integer ) RETURN String IS
    BEGIN
        RETURN Text & Result.Image( Value, 8 );
    END "&";

    FUNCTION "&"( Text : String; Value : Float ) RETURN String IS
    BEGIN
        RETURN Text & Result.Image( Value, 8, 2 );
    END "&";

    FUNCTION "&"( Text : String; Version : Compiler_Version ) RETURN String IS
    BEGIN
        CASE Version IS
            WHEN Syntax_Only      => RETURN Text & "Syntax Only";
            WHEN No_Optimize      => RETURN Text & "No Optimization";
            WHEN Optimize_Space   => RETURN Text & "Space Optimized";
            WHEN Optimize_Time    => RETURN Text & "Time Optimized";
        END CASE;
    END "&";
END Compare;
```

```

PROCEDURE Load_Results IS
  Buffer : String( 1 .. 132 );
  File   : PQAC_IO.File_Type;

  PROCEDURE Read_Record( Name : String; Metric : IN OUT Metric_Record ) IS
    Last : Natural := 0;
    Next : Natural := 0;
  BEGIN
    PQAC_IO.Get_Line( File, Buffer, Last );
    WHILE Next < Last AND THEN Buffer( Next + 1 ) /= ' ' LOOP
      Next := Next + 1;
    END LOOP;
    IF Name /= Buffer( 1 .. Next ) THEN
      Result.Print( Name & " /= " & Buffer( 1 .. Next ) );
      RAISE Undefined_Data;
    END IF;
    PQAC_IO.Get( Buffer( Next + 1 .. Last ), Metric.Compile_Speed, Next );
    PQAC_IO.Get( Buffer( Next + 1 .. Last ), Metric.Execute_Time, Next );
    PQAC_IO.Get( Buffer( Next + 1 .. Last ), Metric.Execute_Size, Next );
    Metric.Alias_Time := Float( Metric.Execute_Time ) / 100.0;
  END Read_Record;

BEGIN
  PQAC_IO.Open_Input( File, Result_File );
  FOR Index IN Compiler_Version LOOP
    Read_Record( Compiler_Version'IMAGE( Index ), Ada_Results( Index ) );
  END LOOP;
  Read_Record( "FORTRAN", Fortran_Results );
  PQAC_IO.Close( File );
  Initialized := True;
EXCEPTION
  WHEN OTHERS => RAISE Undefined_Data;
END Load_Results;

FUNCTION Compile( Version : Compiler_Version ) RETURN Natural IS
BEGIN
  RETURN Ada_Results( Version ).Compile_Speed;
END Compile;

FUNCTION Time( Version : Compiler_Version ) RETURN Natural IS
BEGIN
  RETURN 100 * Ada_Results( Version ).Execute_Time /
    FORTRAN_Results.Execute_Time;
END Time;

FUNCTION Size( Version : Compiler_Version ) RETURN Natural IS
BEGIN
  RETURN 100 * Ada_Results( Version ).Execute_Size /
    FORTRAN_Results.Execute_Size;
END Size;

PROCEDURE Print_Metric( Metric : Metric_Record; Name : String ) IS
  Tag1 : CONSTANT String( 1 .. 17 ) := " Lines/Minute/MIP";
  Tag2 : CONSTANT String( 1 .. 8 ) := " Seconds";
  Tag3 : CONSTANT String( 1 .. 6 ) := " Words";
BEGIN
  Result.Print( "" );
  Result.Print( "Compilation Metrics: " & Name );
  Result.Print( "  Compilation Speed: " & Metric.Compile_Speed & Tag1 );
  Result.Print( "  Object Code Time: " & Metric.Alias_Time & Tag2 );
  Result.Print( "  Object Code Size: " & Metric.Execute_Size & Tag3 );
END Print_Metric;

```

```

PROCEDURE Show( V : Compiler_Version ) IS
BEGIN
  Print_Metric( FORTRAN_Results, "FORTRAN Code - Optimized" );
  Print_Metric( Ada_Results( V ), "Ada Code - " & V );
  Result.Print( "" );
  Result.Print( "  Object Code Time Percentage: " & Time( V ) & "%" );
  Result.Print( "  Object Code Size Percentage: " & Size( V ) & "%" );
END Show;

FUNCTION Normal( Name : String; X : Natural; Min : Natural )
RETURN Natural IS
  Percent : Natural := 0;
BEGIN
  IF X <= Min OR ELSE Min = 0 THEN
    Percent := 100;
  ELSE
    Percent := 100 * Min / X;
  END IF;
  Result.Print( "" & Percent & "% " & Name & " Success" );
  RETURN Percent;
END Normal;

FUNCTION Percentage
( Compiler_Option      : Compiler_Version;
  Minimum_Compile_Rate : Natural;
  Minimum_Size_Percent : Natural;
  Minimum_Time_Percent : Natural ) RETURN Natural IS
  Option : Compiler_Version := Compiler_Option;
  Percent : Natural;

  FUNCTION Switch( N : Natural ) RETURN String IS
  BEGIN
    IF N = 0 THEN
      RETURN " must be greater than" & N & "% of equivalent FORTRAN.";
    ELSE
      RETURN " must be less than " & N & "% of equivalent FORTRAN.";
    END IF;
  END Switch;

  BEGIN
    IF NOT Initialized THEN
      Load_Results;
    END IF;
    Show( Option );
    Result.Print( "" );
    Result.Print( "Compilation Speed must be greater than"
      & Minimum_Compile_Rate & " Lines/Minute/MIP" );
    Result.Print( "Code Execute Time" & Switch( Minimum_Time_Percent ) );
    Result.Print( "Code Object Size " & Switch( Minimum_Size_Percent ) );
    Result.Print( "" );
    Percent :=
      Normal( "Compile Speed", Minimum_Compile_Rate, Compile( Option ) ) *
      Normal( "Code Time", Time( Option ), Minimum_Time_Percent ) *
      Normal( "Code Size", Size( Option ), Minimum_Size_Percent ) / 10000;
    Result.Print( "" & Percent & "% Total Success" );
    RETURN Percent;
  END Percentage;

FUNCTION Result_File RETURN String IS
BEGIN
  RETURN Common.Image( Names.Comparison );
END Result_File;

```

END Compare;

Source File: COUNT_.ADA

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Count.Ada
--       Component: Package Specification Count
--       Description: This package contains two subprograms. One counts the
--                   number of Ada source lines in a text file, and one computes
--                   the size of a file in machine words.
--
--       Ada Source Lines Definition:
--
--       Any statement terminated with ';' counts as one source
--       line except any ';'s between matched parentheses such
--       as in a subprogram parameter list. Text to the right
--       of the comment delimiter "--" is ignored. Text embedded
--       in character '?' or string "??..." literals is also
--       ignored.
```

PACKAGE Count IS

Count_Error : EXCEPTION;

PROCEDURE Count_File(Input_File : String; Output_File : String);

```
-- This procedure counts the number of Ada source lines in the Input_File.
-- The count of the number of lines is written to the Output_File.
-- Count_Error will be raised if Input_File does not exist or if the
-- Input_File contains invalid syntax such as unmatched parenthesis.
-- The Output_File will contain a single value denoting the number
-- of Ada source lines in the Input_File. This procedure will work
-- on non-Ada text ( probably finding 0 lines ) without error unless
-- the parenthesis in the file are not matched.
```

PROCEDURE Code_Size(Input_File : String; Output_File : String);

```
-- This procedure counts the number of machine words in the Input_File.
-- The count of the number of lines is written to the Output_File.
-- Count_Error will be raised if Input_File does not exist.
-- The Output_File will contain a single value denoting the number
-- of machine words in the Input_File.
```

END Count;

Source File: COUNT.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Count.Ada
--       Component: Package Body Count
--       Description: This package contains two subprograms. One counts the
--                   number of Ada source lines in a text file, and one computes
--                   the size of a file in machine words.
--
```

```
WITH Twine;      -- String Manipulation Package
WITH PQAC_IO;    -- Centralized Input and Output Package
WITH Sequential_IO;
```

PACKAGE BODY Count IS

PROCEDURE Count_File(Input_File : String; Output_File : String) IS

```
Input      : PQAC_IO.File_Type;
Finished   : Boolean := False;
Pair       : String( 1 .. 2 ) := " ";
Buffer     : Twine.Input_Buffer;
Pointer    : Natural := 0;
Length     : Natural := 0;
```

```
Source_Lines : Natural := 0;
Text_Lines   : Natural := 0;
Comments     : Natural := 0;
```

FUNCTION My_Get RETURN Character IS

-- Acts as a character stream.

```
BEGIN
  IF Pointer >= Length AND PQAC_IO.End_Of_File( Input ) THEN
    Finished := True;
    RETURN ' ';
  ELSIF Pointer >= Length THEN
    PQAC_IO.Get_Line( Input, Buffer, Length );
    Text_Lines := Text_Lines + 1;
    Pointer := 0;
  END IF;
  Pointer := Pointer + 1;
  RETURN Buffer( Pointer );
END My_Get;
```

PROCEDURE Fill_Buffer IS

```
BEGIN
  Pair( 1 ) := Pair( 2 );
  Pair( 2 ) := My_Get;
END Fill_Buffer;
```

PROCEDURE Flush_String(Char : IN Character) IS

-- Flushes characters on the line until Char is found.

```
BEGIN
  LOOP
    Fill_Buffer;
    IF Pair = Char & Char THEN
      Fill_Buffer;
    ELSIF Pair( 1 ) = Char THEN
      EXIT;
    END IF;
    IF Finished THEN
      PQAC_IO.Record_Error( "String not matched." );
    END IF;
  END LOOP;
```



```

        RAISE Count_Error;
    END IF;
END LOOP;
END Flush_String;

```

```

PROCEDURE Check_For_Comment IS
BEGIN
    IF Pair = "--" THEN
        Pointer := Length;
        Pair := " ";
        Comments := Comments + 1;
    END IF;
END Check_For_Comment;

```

```

FUNCTION Check_For_Character RETURN Boolean IS
    -- Returns true if the ' is part of a character literal '?'.
    Found : Boolean;
BEGIN
    Fill_Buffer;
    Found := Pair( 2 ) = "'";
    IF Found THEN
        Fill_Buffer;
    END IF;
    RETURN Found;
END Check_For_Character;

```

```

PROCEDURE Paren_Error IS
BEGIN
    PQAC_IO.Record_Error( "Parenthesis not matched." );
    RAISE Count_Error;
END Paren_Error;

```

```

PROCEDURE Flush_Parens IS
    -- Reads characters until a matching right paren is found.

    Level      : Natural := 1;
    Need_New   : Boolean := True;
    -- Need_New is used to keep track if an ' was found.
    -- If so, then another character must be scanned to determine
    -- if the ' is a matched '?' or an attribute something'attr.
BEGIN
    WHILE Level > 0 AND NOT Finished LOOP
        IF Need_New THEN
            Fill_Buffer;
        END IF;
        Need_New := True;
        CASE Pair( 1 ) IS
            WHEN '-' => Check_For_Comment;
            WHEN "'" => Need_New := Check_For_Character;
            WHEN '"' ! '%' => Flush_String( Pair( 1 ) );
            WHEN '(' => Level := Level + 1;
            WHEN ')' => Level := Level - 1;
            WHEN OTHERS => NULL;
        END CASE;
    END LOOP;
    IF Level > 0 THEN
        Paren_Error;
    END IF;
END Flush_Parens;

```

```

PROCEDURE Count_Lines IS
    Need_New : Boolean := True;
    -- Need_New is used to keep track if an ' was found.
    -- If so, then another character must be scanned to determine

```

Source File: COUNT.ADA

```

    -- if the ' is a matched '?' or an attribute something'attr.
BEGIN
    WHILE NOT Finished LOOP
        IF Need_New THEN
            Fill_Buffer;
        END IF;
        Need_New := True;
        CASE Pair( 1 ) IS
            WHEN '-' => Check_For_Comment;
            WHEN '(' => Flush_Parens;
            WHEN ')' => Paren_Error;
            WHEN '"' => Need_New := Check_For_Character;
            WHEN '"' ! '%' => Flush_String( Pair( 1 ) );
            WHEN ';' => Source_Lines := Source_Lines + 1;
            WHEN OTHERS => NULL;
        END CASE;
    END LOOP;
END Count_Lines;

BEGIN
    PQAC_IO.Open_Input( Input, Input_File );
    Count_Lines;
    PQAC_IO.Close( Input );
    PQAC_IO.Put_Value( Output_File, Source_Lines );
EXCEPTION
    WHEN Count_Error => RAISE;
    WHEN OTHERS =>
        PQAC_IO.Record_Error( "Error reading " & Input_File & "." );
        RAISE Count_Error;
END Count_File;

PROCEDURE Code_Size( Input_File : String; Output_File : String ) IS
    SUBTYPE New_String IS String( 1 .. 512 );
    PACKAGE Seq_IO IS NEW Sequential_IO( New_String );
    Input : Seq_IO.File_Type;
    Buffer : New_String;
    Total : Natural := 0;
BEGIN
    Seq_IO.Open( Input, Seq_IO.In_File, Input_File );
    WHILE NOT Seq_IO.End_Of_File( Input ) LOOP
        Seq_IO.Read( Input, Buffer );
        Total := Total + New_String'LENGTH;
    END LOOP;
    Seq_IO.Close( Input );
    PQAC_IO.Put_Value( Output_File, Total );
EXCEPTION
    WHEN OTHERS =>
        PQAC_IO.Record_Error( "Error reading " & Input_File & "." );
        RAISE Count_Error;
END Code_Size;

END Count;
```

```
--
--
--           The Aerospace Corporation
--
-- Production Quality Ada Compiler Test Suite Support Software
--
--
-- Author:  BAP
-- Date:   10/01/88
-- File:   Expand_.Ada
-- Component: Package Specification Expand
-- Description:
--           Generates text from a supplied input file containing text
--           templates with embedded meta symbols. The text does not
--           need to be Ada code, but that is the intent of the package.
--
--           This package relies heavily on the Syntax package. Tests
--           are first Parsed, then Expanded. The Syntax package contains
--           syntax information for both of these actions. Expand uses
--           only a subset of the meta symbols operated on by the package
--           Syntax. If a meta symbol is found that is used only by Parse
--           and not by Expand, then Expand_Error will be raised.
--
--           If any syntax errors are found in the Input_File, a message
--           informing the user of the problem will be written to the
--           output stream and Expand_Error will be raised.
--
--           Valid meta symbols for Expand are the first non-blank
--           characters on a line that begin with --!. There are three
--           meta commands recognized by Expand: EQUATE, LOOP, and END.
--           The syntax for these commands are as follows:
--
--           --! EQUATE Value IS 20
--           --! LOOP 10 STEP 1 START 1 [1]
--           --! END [1]
--
--           Values from the EQUATE statement may be used as LOOP statement
--           parameters. Text between statements of the form
--
--           --! LOOP x STEP y START z [1]
--           Lines of Text
--           --! END [1]
--
--           is repeated x times. The implicit loop counter is initially
--           set to z, and incremented by y after every iteration.
--           The entire range of the implicit loop counter must remain
--           positive, although it may decrease by setting step < 0.
--
--           Loop Statement Syntax:
--
--           The order of the three reserved words in the LOOP statement
--           doesn't matter. In addition, all but one of the three
--           fields may be omitted. A default value of 1 will be used.
--
--           --! LOOP 10 [1]           == --! LOOP 10 STEP 1 START 1 [1]
--           --! START 10 [1]          == --! LOOP 1 STEP 1 START 10 [1]
--           --! STEP 2 LOOP 5 [1]     == --! LOOP 5 STEP 2 START 1 [1]
--
--           The [1] designates the level of the loop. Loop levels can
--           be from 1 to 9. Each LOOP and END statement must contain
--           a loop level. The loop level must correspond to the actual
--           loop level. Examples:
--
--           --! LOOP 10 [1]           -- Legal
--           Some text
--           --! LOOP 5 [2]
--           Some more text
--           --! END [2]
--           --! LOOP 5 [2]
--           Some more text
--           --! END [2]
--           --! END [1]
--
--           --! LOOP 10 [1]
```

```

--      Some text
--      --! LOOP 5 [2]      -- Illegal: Doesn't match --!
--      Some more text    --
--      --! END [1]        -- <-----!
--      --! END [2]
--
--      --! LOOP 10 [2]    -- Illegal: This [2] must be [1]
--      Some text
--      --! LOOP 5 [2]    -- Illegal: [2] already used
--      Some more text
--      --! END [2]
--      --! END [2]
--
--      --! LOOP 10 [1]
--      Some text
--      --! LOOP 5 [3]    -- Illegal: This [3] must be [2]
--      Some more text
--      --! END [3]
--      --! END [1]

```

The values for LOOP x STEP y START z may be EQUATED names.

```

--      --! EQUATE Iterations IS 20
--      --! EQUATE Beginning IS 10 * 2
--      --! EQUATE Jump_Size IS Iterations / 5 + 1
--      --! LOOP Iterations STEP Jump_Size START Beginning [1]
--      Some Text
--      --! END [1]

```

Implicit Loop Counter:

The value of the implicit loop counter may be accessed by the text inside the loop using [expression]. This entire expression will be replaced by the expression value. The implicit loop counter does not have to be accessed.

The valid expressions are [x], [x+y], and [x-y]. Here, x is from 1 .. 9 denoting the loop level, and y is an offset.

Example Program:

```

--      --! EQUATE Size IS 2
--      --! LOOP Size STEP -10 START 100 [1]
--      --! LOOP Size [2]
--
--      PROCEDURE Temp_[1]_[2] IS
--      BEGIN
--          Perform( [1-10], [2+3] );
--      END Temp_[1]_[2];
--
--      --! END [2]
--      --! END [1]

```

Here we have a procedure inside two levels of loops. The outer loop LOOPS 2 (Size) times as does the inner loop. Therefore, 2 * 2 or 4 copies of the procedure will be made. The outer loop counter is accessed by [1] and [1-10]. The inner loop counter is accessed by [2] and [2+3].

Sequence for [1]:	100	90	(Start 100, Step -10)
Sequence for [2]:	1	2	(Start 1, Step 1)
Sequence for [1-10]:	90	80	(100-10, 90-10)
Sequence for [2+3]:	4	5	(1+3, 2+3)

Expanded Becomes:

```

--      PROCEDURE Temp_100_1 IS
--      BEGIN
--          Perform( 90, 4 );

```

Source File: EXPAND.ADA

```
--          END Temp_100_1;
--
--          PROCEDURE Temp_100_2 IS
--          BEGIN
--              Perform( 90, 5 );
--          END Temp_100_2;
--
--          PROCEDURE Temp_90_1 IS
--          BEGIN
--              Perform( 80, 4 );
--          END Temp_90_1;
--
--          PROCEDURE Temp_90_2 IS
--          BEGIN
--              Perform( 80, 5 );
--          END Temp_90_2;
--
--
PACKAGE Expand IS

    Expand_Error : EXCEPTION;

    PROCEDURE Expand_File( Input_File : String; Output_File : String );
        -- The Input_File containing templates and meta symbols is read in.
        -- The expanded templates are written to the given Output_File.
        -- Expand_Error will be raised if there is a problem with the Input_File.

END Expand;
```

Source File: EXPAND.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:  BAP
--       Date:    10/01/88
--       File:    Expand.Ada
--       Component: Package Body Expand
--       Description: ( See package specification description )
--
WITH Twine;      -- String Manipulation Package
WITH Syntax;     -- Meta Symbol Parsing Package
WITH PQAC_IO;    -- Centralized Input and Output Package

PACKAGE BODY Expand IS

  Left_Character   : CONSTANT Character := '[';
  Right_Character  : CONSTANT Character := ']';

  Max_Actions      : CONSTANT := 100;  -- maximum number of loops
  Max_Text_Lines   : CONSTANT := 500;  -- maximum lines of input
  Max_Variables    : CONSTANT := 50;   -- maximum number implicit counter
                                         -- accesses per loop

  TYPE Boolean_List IS ARRAY( Positive RANGE <> ) OF Boolean;

  TYPE Boolean_List_Access IS ACCESS Boolean_List;

  TYPE Coordinate IS RECORD
    Line       : Natural := 0;  -- Text Line Number
    Position   : Natural := 0;  -- Character position in the line
    Offset     : Integer := 0;  -- Offset from the counter value
  END RECORD;

  TYPE Coordinate_Array IS ARRAY( 1 .. Max_Variables ) OF Coordinate;

  TYPE Action_Type IS RECORD
    Level       : Natural := 0;  -- level of the loop: 1 .. 9
    Var_Count   : Natural := 0;  -- Number of counter accesses
    First       : Integer := 1;  -- First line of loop
    Last        : Integer := 1;  -- Last line of loop
    Start       : Integer := 1;  -- First value of loop counter
    Copies      : Natural := 1;  -- Number of loop iterations
    Step        : Integer := 1;  -- Counter step size
    Width       : Natural := 0;  -- Maximum width of counter image
    Var_Position : Coordinate_Array; -- List of counter accesses
    Start_Image : Twine.Series;  -- Initial image of counter
    Index_Image : Twine.Series;  -- Current image of counter
  END RECORD;

  TYPE Line_Descriptor IS RECORD
    Line        : Twine.Series;  -- Text Line
    Template_At : Boolean_List_Access; -- Counter access position on/off list
    Error_Lines : Natural := 0;  -- Saves original line # for messages
  END RECORD;

  Lines : ARRAY( 1 .. Max_Text_Lines ) OF Line_Descriptor;
  Action : ARRAY( 0 .. Max_Actions ) OF Action_Type;

  Number_Of_Actions : Natural := 0;
  Number_Of_Lines   : Natural := 0;
  Error_Line_Number : Natural := 0;
  Error_Line_Position : Natural := 0;
  Error_Last        : Natural := 0;
  Error_Line        : Twine.Input_Buffer;
```

PACKAGE Stack IS

```

Overflow : EXCEPTION;
Underflow : EXCEPTION;

PROCEDURE Push( Value : Natural );
FUNCTION Pop RETURN Natural;

```

END Stack;

PACKAGE BODY Stack IS

```

Stack      : ARRAY( 1 .. Max_Actions ) OF Natural;
Stack_Pointer : Positive := 1;
Full       : Boolean := False;

```

```

PROCEDURE Push( Value : Natural ) IS
BEGIN
  IF Full THEN
    RAISE Overflow;
  END IF;
  Stack( Stack_Pointer ) := Value;
  IF Stack_Pointer < Max_Actions THEN
    Stack_Pointer := Stack_Pointer + 1;
  ELSE
    Full := True;
  END IF;
END Push;

```

```

FUNCTION Pop RETURN Natural IS
BEGIN
  IF Stack_Pointer = 1 THEN
    RAISE Underflow;
  END IF;
  IF Full THEN
    Full := False;
  ELSE
    Stack_Pointer := Stack_Pointer - 1;
  END IF;
  RETURN Stack( Stack_Pointer );
END Pop;

```

END Stack;

```

PROCEDURE Process_Error( Message : String; Position : Natural := 0 ) IS
  Blanks : CONSTANT Twine.Input_Buffer := ( OTHERS => ' ' );
BEGIN
  IF Position > 0 THEN
    Error_Line_Position := Position;
  END IF;
  PQAC_IO.Record_Error( "" );
  PQAC_IO.Record_Error( "Error in Code Expander Program:" );
  PQAC_IO.Record_Error( "" );
  PQAC_IO.Record_Error( Message );
  IF Error_Line_Number /= 0 THEN
    PQAC_IO.Record_Error
      ( "Line Number:" & Twine.Image( Error_Line_Number, 8 ) );
  END IF;
  IF Error_Last > 0 THEN
    PQAC_IO.Record_Error( Error_Line( 1 .. Error_Last ) );
  END IF;
  IF Error_Line_Position /= 0 THEN
    PQAC_IO.Record_Error
      ( Blanks( 1..Error_Line_Position - 1 ) & " <---- Offending Item" );
  END IF;
  PQAC_IO.Record_Error( "" );

```

```

    RAISE Expand_Error;
END Process_Error;

```

```

FUNCTION Loop_Level( Char : Character; Err : Natural := 0 ) RETURN Natural IS
    -- Char must be in 1 .. 9 or an exception is raised.
BEGIN
    IF Char NOT IN '1' .. '9' THEN
        Error_Line_Position := Err;
        Process_Error( "[N] expected, N from 1 .. 9." );
    END IF;
    RETURN Integer'VALUE( Char & "" );
END Loop_Level;

```

```

PROCEDURE Read_In_Templates( File_Name : String ) IS

```

```

    -- The input file is read into the array Lines.
    -- All of the meta symbols are found and loop information
    -- is saved with each line.

```

```

    File      : PQAC_IO.File_Type;
    Buffer     : Twine.Input_Buffer;
    Last      : Natural := 0;
    Current_Line : Natural := 0;
    Current_Item : Natural := 0;
    Current_Level : Natural := 0;

```

```

PROCEDURE Check_Level( Line : String; Level : Natural ) IS
    -- Raises an Exception if the loop level of the line is not [Level]
BEGIN
    FOR Index IN Line'FIRST .. Line'LAST - 2 LOOP
        IF Line( Index ) = Left_Character AND THEN
            Line( Index + 2 ) = Right_Character THEN
                IF Loop_Level( Line( Index + 1 ), Index + 1 ) = Level THEN
                    RETURN;
                ELSE
                    Process_Error( "Incorrect loop level.", Index + 1 );
                END IF;
            END IF;
        END LOOP;
        Process_Error( "Loop level not designated.", Line'LAST );
    END Check_Level;

```

```

PROCEDURE Parse_Line( Line : String; Item : IN OUT Action_Type ) IS
BEGIN
    Syntax.Parse_Loop
        ( Line, Item.Copies, Item.Start, Item.Step, Item.Width );
    Item.Start_Image :=
        Twine.Create( Twine.Zeroed_Image( Item.Start, Item.Width ) );
    Item.Index_Image :=
        Twine.Create( Twine.Zeroed_Image( Item.Start, Item.Width ) );
EXCEPTION
    WHEN Syntax.Count_Error =>
        Process_Error( "Iteration step must be non-zero." );
    WHEN Syntax.Step_Error =>
        Process_Error( "Iteration step must be non-zero." );
    WHEN Syntax.Range_Error =>
        Process_Error( "Range of loop values must be non-negative." );
    WHEN Syntax.Name_Error =>
        Process_Error( "Identifier not defined." );
    WHEN Syntax.Value_Error =>
        Process_Error( "Integer value expected here." );
END Parse_Line;

```

```

PROCEDURE Parse_Equivalence_Line( Text : String ) IS
BEGIN
    Syntax.Parse_Equivalence( Text );
EXCEPTION

```



```

    WHEN Syntax.Statement_Error =>
        Process_Error( "Reserved word IS not found." );
    WHEN Syntax.Capacity_Error =>
        Process_Error( "Exceeded equivalence capacities." );
    WHEN Syntax.Duplicate_Error =>
        Process_Error( "Equivalence name used twice." );
    WHEN Syntax.Name_Error =>
        Process_Error( "Identifier not defined." );
    WHEN Syntax.Value_Error =>
        Process_Error( "Integer value expected here." );
END Parse_Equivalence_Line;

PROCEDURE Set_Line( Current : Natural; Line : String; Error : Natural ) IS
    All_False : Boolean_List( Twine.Input_Buffer'RANGE )
        := ( OTHERS => False );
BEGIN
    Lines( Current ).Line := Twine.Create( Line );
    Lines( Current ).Template_At := NEW Boolean_List'( All_False );
    Lines( Current ).Error_Lines := Error;
END Set_Line;

PROCEDURE Process_Line( Line : String ) IS
    -- If the line is normal text or a comment it is simply added to the
    -- to the text buffer. If it is an EQUATE, LOOP, or END statement
    -- then this statement is parsed and the information is saved.
    -- If it is some other special line, then an exception is raised.
    Text : String( Line'RANGE ) := Line;
BEGIN
    Twine.Upper_Case( Text );
    CASE Syntax.Process_Value_Of( Text ) IS
        WHEN Syntax.Normal_Text ! Syntax.Comment_Line =>
            IF Current_Line = Max_Text_Lines THEN
                Process_Error( "Input file too large." );
            END IF;
            Current_Line := Current_Line + 1;
            Set_Line( Current_Line, Line, Error_Line_Number );
        WHEN Syntax.Equivalence =>
            Parse_Equivalence_Line( Text );
        WHEN Syntax.Start_Loop =>
            Current_Level := Current_Level + 1;
            Current_Item := Current_Item + 1;
            IF Current_Item > Action'LAST THEN
                Process_Error( "Maximum number of loops exceeded." );
            END IF;
            Check_Level( Line, Current_Level );
            Parse_Line( Line, Action( Current_Item ) );
            Action( Current_Item ).Level := Current_Level;
            Action( Current_Item ).First := Current_Line + 1;
            Stack.Push( Current_Item );
        WHEN Syntax.End_Loop =>
            Check_Level( Line, Current_Level );
            Action( Stack.Pop ).Last := Current_Line;
            Current_Level := Current_Level - 1;
        WHEN Syntax.In_Error =>
            Process_Error( "Unknown Command." );
        WHEN OTHERS =>
            Process_Error( "Unexpected Command." );
    END CASE;
END Process_Line;

BEGIN
    PQAC_IO.Open_Input( File, File_Name );
    IF PQAC_IO.End_Of_File( File ) THEN
        Process_Error( "No Text In File." );
    END IF;
    WHILE NOT PQAC_IO.End_Of_File( File ) LOOP
        Error_Line_Number := Error_Line_Number + 1;
        PQAC_IO.Get_Line( File, Buffer, Last );
        Error_Last := Last;
    END LOOP;

```

```

        Error_Line := Buffer;
        Process_Line( Buffer( 1 .. Last ) );
    END LOOP;
    IF Current_Level > 0 THEN
        Process_Error( "Loop construct not closed." );
    END IF;
    IF Current_Line = 0 THEN
        Process_Error( "No Text In File." );
    END IF;
    Number_Of_Lines := Current_Line;
    Number_Of_Actions := Current_Item;
    Action( 0 ).Last := Number_Of_Lines;
    PQAC_IO.Close( File );
EXCEPTION
    WHEN Stack.Underflow =>
        Process_Error( "End Of Loop encountered with no begin." );
END Read_In_Templates;

```

PROCEDURE Initialize_Templates IS

```

-- Each of the implicit loop counters are initialized to their first
-- value. The saved normal text is examined for the presence of
-- accesses to the implicit loop counter. Enough room in the text
-- line is then made for the maximum width of the counter image.
-- The positions of each of the accesses is recorded.

```

```

Save_Item      : ARRAY( 1 .. Max_Actions ) OF Natural;
Item           : Natural := 0;
Current_Level  : Natural := 0;

```

PROCEDURE Prepare(Current_Line : Integer; Max_Level : Integer) IS

```

Old_Line      : Twine.Input_Buffer := ( OTHERS => ' ' );
New_Line      : Twine.Input_Buffer := ( OTHERS => ' ' );
Old_Last      : Natural := 0;
Old_Pointer   : Natural := 0;
New_Pointer   : Natural := 0;
Level         : Natural := 0;
Offset        : Integer := 0;
Char          : Character := ' ';

```

FUNCTION Read_Char RETURN Character IS

```

-- Reads next character from line
BEGIN
    Old_Pointer := Old_Pointer + 1;
    IF Old_Pointer > Old_Last THEN
        RETURN ' ';
    ELSE
        RETURN Old_Line( Old_Pointer );
    END IF;
END Read_Char;

```

PROCEDURE Put_Char(Char : Character) IS

```

-- Puts this Char to the output line
BEGIN
    New_Pointer := New_Pointer + 1;
    IF New_Pointer > New_Line'LAST THEN
        Process_Error( "Size of generated line too large." );
    END IF;
    New_Line( New_Pointer ) := Char;
END Put_Char;

```

PROCEDURE Read_Variable(Item : OUT Integer; Offset : OUT Integer) IS

```

-- The previous character read was '['.
-- Character are read until ']' is found.

```

```

-- [1]:      Item --> 1, Offset --> 0
-- [2-3]:    Item --> 2, Offset --> -3
-- [5+40]:   Item --> 5, Offset --> 40
--
-- [10]      ERROR: Must be 1 .. 9

Char : Character := ' ';
Head : Natural   := 0;
Size : Integer   := 1;
BEGIN
  Char := Read_Char;
  Size := Loop_Level( Char, Old_Pointer );
  Item := Size;
  IF Size > Max_Level THEN
    Process_Error( "No loop for this variable.", Old_Pointer );
  END IF;
  Char := Read_Char;
  Offset := 0;
  IF Twine.Sign( Char ) THEN
    Head := Old_Pointer;
    Char := Read_Char;
    IF NOT Twine.Digit( Char ) THEN
      Process_Error( "Number expected.", Old_Pointer );
    END IF;
    LOOP
      Char := Read_Char;
      EXIT WHEN Char = Right_Character;
      IF NOT Twine.Digit( Char ) THEN
        Process_Error( "Number expected.", Old_Pointer );
      END IF;
    END LOOP;
    Offset := Integer'VALUE( Old_Line( Head .. Old_Pointer - 1 ));
  END IF;
  IF Char /= Right_Character THEN
    Process_Error( Right_Character & " expected.", Old_Pointer );
  END IF;
END Read_Variable;

PROCEDURE Save_State( Item : IN OUT Action_Type; Offset : Integer ) IS
  -- Saves information about the current counter access
BEGIN
  IF Item.Var_Count = Max_Variables THEN
    Process_Error( "Too many variables in loop.", Old_Pointer );
  END IF;
  Lines( Current_Line ).Template_At( New_Pointer ) := True;
  Item.Var_Count := Item.Var_Count + 1;
  Item.Var_Position( Item.Var_Count )
    := ( Current_Line, New_Pointer, Offset );
  New_Pointer := New_Pointer + Twine.Length( Item.Start_Image ) - 1;
  IF New_Pointer > New_Line'LAST THEN
    Process_Error( "Size of generated line too large." );
  END IF;
END Save_State;

BEGIN
  Old_Last := Twine.Length( Lines( Current_Line ).Line );
  Old_Line( 1 .. Old_Last ) := Twine.Image( Lines( Current_Line ).Line );
  Error_Line_Number := Lines( Current_Line ).Error_Lines;
  Error_Last := Old_Last;
  Error_Line := Old_Line;
  WHILE Old_Pointer < Old_Last LOOP
    Char := Read_Char;
    Put_Char( Char );
    IF Char = Left_Character THEN
      Read_Variable( Level, Offset );
      Save_State( Action( Save_Item( Level ) ), Offset );
    END IF;
  END LOOP;
  IF Old_Last /= New_Pointer OR ELSE Old_Line /= New_Line THEN
    Lines( Current_Line ).Line :=
      Twine.Create( New_Line( 1 .. New_Pointer ) );
  END IF;

```

Source File: EXPAND.ADA

```
END Prepare;

BEGIN
  Item := 1;
  Current_Level := 0;
  FOR Index IN 1 .. Number_Of_Lines LOOP
    WHILE Item <= Action'LAST AND THEN Action( Item ).First = Index LOOP
      Current_Level := Current_Level + 1;
      Save_Item( Current_Level ) := Item;
      Item := Item + 1;
    END LOOP;
    Prepare( Index, Current_Level );
    WHILE Current_Level > 0 AND THEN
      Action( Save_Item( Current_Level ) ).Last = Index
    LOOP
      Current_Level := Current_Level - 1;
    END LOOP;
  END LOOP;
END Initialize_Templates;
```

PROCEDURE Write_Out_Program(File_Name : String) IS

```
-- The saved text file is written out. The positions of the
-- first and last line of each loop are passed to a procedure
-- which recursively calls itself to perform nested looping.
--
-- Image such as PROCEDURE_0003 are written out as PROCEDURE_3
```

File : PQAC_IO.File_Type;

PROCEDURE Update

(Line : IN OUT Twine.Series;
Pair : Twine.Bounds;
Step : Integer) IS

Size : Natural := Pair.Tail - Pair.Head + 1;

FUNCTION Next_Value RETURN Integer IS

BEGIN
RETURN Integer'VALUE(Twine.Substring(Line, Pair)) + Step;
END Next_Value;

BEGIN

Twine.Copy(Line, Pair, Twine.Zeroed_Image(Next_Value, Size));
END Update;

PROCEDURE Replace

(Item : Coordinate;
Image : Twine.Series) IS

```
-- Copies the Image to the position of Item.
-- If Item contains an offset, it is added to the image.
```

Pair : Twine.Bounds :=
(Head => Item.Position,
Tail => Item.Position + Twine.Length(Image) - 1);

BEGIN

Twine.Copy(Lines(Item.Line).Line, Pair, Image);
IF Item.Offset /= 0 THEN
Update(Lines(Item.Line).Line, Pair, Item.Offset);
END IF;
END Replace;

PROCEDURE Put_A_Line(Item : Positive) IS

```
-- Outputs the Item number line of the saved text.
```

```

-- Leading zeroes are removed from counter images.

Buffer      : Twine.Input_Buffer;
Pair        : Twine.Bounds := Twine.Area( Lines( Item ).Line );
Char        : Character;
Size        : Natural := 0;
Skip        : Boolean := False;
Last_Skip   : Boolean := False;
BEGIN
  FOR Index IN Pair.Head .. Pair.Tail LOOP
    Char := Twine.Element( Lines( Item ).Line, Index );
    Last_Skip := Skip;
    Skip := Skip OR ELSE Lines( Item ).Template_At( Index );
    Skip := Skip AND THEN Char = '0';
    IF Last_Skip
      AND THEN ( NOT Skip )
      AND THEN ( NOT Twine.Digit( Char ) ) THEN
      Size := Size + 1;
      Buffer( Size ) := '0';
    END IF;
    IF NOT Skip THEN
      Size := Size + 1;
      Buffer( Size ) := Char;
    END IF;
  END LOOP;
  IF Skip THEN
    Size := Size + 1;
    Buffer( Size ) := '0';
  END IF;
  PQAC_IO.Put_Line( File, Buffer( 1 .. Size ) );
END Put_A_Line;

PROCEDURE Print_A_Loop( Item : Natural ) IS
  -- Item refers to the current loop.
  -- This loop is iterated over its range, with the text template
  -- between the LOOP and END being written out each time.
  -- If another loop is found embedded in this text, this procedure
  -- calls itself to process that loop before continuing.

  Index : Integer := 0;
  Count : Integer := 0;
  Next  : Integer := 0;

  FUNCTION Find_Next
    ( N      : Integer;
      Level  : Integer;
      Last   : Integer ) RETURN Integer IS
    Next : Integer := 0;
    Found : Boolean := False;
  BEGIN
    Next := N;
    WHILE ( NOT Found ) AND ( Next < Number_Of_Actions ) LOOP
      Next := Next + 1;
      Found := ( Action( Next ).Level = Level );
    END LOOP;
    IF NOT Found OR ELSE Action( Next ).First > Last THEN
      RETURN 0;
    ELSE
      RETURN Next;
    END IF;
  END Find_Next;

  BEGIN
    IF Item > 0 THEN
      Twine.Copy
        ( Action( Item ).Index_Image, Action( Item ).Start_Image );
    END IF;
    FOR Index IN 1 .. Action( Item ).Copies LOOP

```

Source File: EXPAND.ADA

```

    FOR Count IN 1 .. Action( Item ).Var_Count LOOP
        Replace( Action( Item ).Var_Position( Count ),
            Action( Item ).Index_Image );
    END LOOP;
    IF Item > 0 THEN
        Update( Action( Item ).Index_Image,
            Twine.Area( Action( Item ).Index_Image ),
            Action( Item ).Step );
    END IF;
    Count := Action( Item ).First;
    Next := Item;
    WHILE Count <= Action( Item ).Last LOOP
        Next := Find_Next
            ( Next, Action( Item ).Level + 1, Action( Item ).Last );
        IF Next = 0 THEN
            WHILE Count <= Action( Item ).Last LOOP
                Put_A_Line( Count );
                Count := Count + 1;
            END LOOP;
        ELSE
            WHILE Count < Action( Next ).First LOOP
                Put_A_Line( Count );
                Count := Count + 1;
            END LOOP;
            Print_A_Loop( Next );
            Count := Action( Next ).Last + 1;
        END IF;
    END LOOP;
END Print_A_Loop;

BEGIN
    PQAC_IO.Open_Output( File, File_Name );
    Print_A_Loop( 0 );      -- Start the recursion by called Zero level loop
                           -- which is the entire text area.
    PQAC_IO.Close( File );
END Write_Out_Program;

PROCEDURE Expand_File( Input_File : String; Output_File : String ) IS
BEGIN
    Read_In_Templates( Input_File );
    Initialize_Templates;
    Write_Out_Program( Output_File );
END Expand_File;

END Expand;
```

Source File: NAMES_.ADA

```
--
--
--               The Aerospace Corporation
--
-- Production Quality Ada Compiler Test Suite Support Software
--
--
-- Author:  BAP
-- Date:    10/01/88
-- File:    Names_.Ada
-- Component: Package Specification Names
-- Description: Enumeration types used by support software.
--
```

PACKAGE Names IS

TYPE File_Category IS

```
( Test,           -- Test Description
  List,           -- Compiler Listing
  Machine,        -- Compiler Machine Code Listing
  Ada,            -- Ada Code
  FORTRAN,        -- FORTRAN Code
  Expand,         -- Templates to be Expanded with Expand
  Execute,        -- Executable Code
  Object,         -- Linker Object Code
  Data,           -- Program Data
  Script );       -- Operating System Script
```

TYPE Compiler_Options IS

```
( Syntax_Only,    -- Perform Syntax Checking Only
  Optimize_Time,  -- Optimize for Time
  Optimize_Space, -- Optimize for Space
  Assembly_Listing, -- Create and Assembly Machine Code Listing
  Compiler_Listing, -- Create a Source Code Compiler Listing
  Statistics,      -- Put Maximum Amount of Compiler Statistics in Listing
  No_Optimize,     -- Perform no Optimization
  Time_Compile );  -- Special: Informs test procedures to time the compile
```

TYPE Transfer_Files IS

```
( Save_Time_1,    -- Start Time
  Save_Time_2,    -- Stop Time
  Save_Count,     -- Count of Ada Source Lines
  Comparison,     -- Comparison Data from Ada VS. FORTRAN
  Test_Result,    -- Contains Results of all tests so far.
  Parameters,     -- Passes parameters between programs
  PQAC_State );   -- State of PQAC test suite and working Ada library
```

TYPE OS_Primitives IS

```
( Create_Library, -- Create a new and empty Library
  Remove_Library, -- Delete the current Library
  Compile,        -- Ada Compile a file
  Link,          -- Ada Link a file
  Execute,       -- Execute a file
  Delete,        -- Delete a file
  List,          -- List out a file
  Expand,        -- Call the Expand Procedure
  Store_Time,    -- Save the current time in a file
  Compute_Rate,  -- Compute elapsed time and speed from file data
  Code_Size,     -- Save the size of the executable code in a file
  Count,        -- Save the number of Ada source lines in a file
  Print,        -- Print a string to the output stream
  Fortran,      -- FORTRAN compile a file
  Link_Fortran ); -- FORTRAN link a file
```

END Names;

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Parse_.Ada
--       Component: Package Specification Parse
--       Description:
--           Generates a script from a supplied input file containing test
--           information with embedded meta symbols. The first part of
--           the file should contain Ada comments with the test number,
--           and a description of the test. These comments get written
--           to the script output. The rest of the file may be one or
--           more Ada or FORTRAN code segments.
--
--           This package relies heavily on the Syntax package. Tests
--           are first Parsed, then Expanded if needed.
--
--           If any syntax errors are found in the Input_File, a message
--           informing the user of the problem will be written to the
--           output stream and Parse_Error will be raised.
--
--           Valid meta symbols for Parse are the first non-blank
--           characters on a line that begin with --*. There are seven
--           meta commands recognized by Parse: BEGIN, END, COMPILE,
--           FORTRAN, EXECUTE, COMPARE, and NEW_LIBRARY
--
--           The syntax for these commands are as follows:
--
--           BEGIN and END:
--
--               --* BEGIN  Compiler_1 Compiler_2 ...
--               Some Text: May be text, code, meta symbols, or whatever
--               --* END
--
--           Examples:
--
--               --* BEGIN
--               This text will not be used by any compiler
--               --* END
--
--               --* BEGIN  Dec_Vax_V1_4
--               This text will be used only by the DEC VAX compiler
--               --* END
--
--               --* BEGIN  Dec_Vax_V1_4  TeleGen2_V3_15
--               This text will be used by the DEC VAX and TeleGen2 compilers
--               --* END
--
--           COMPILE and FORTRAN:
--
--               --* COMPILE File_Name Option_1 Option_2 ...
--               --* FORTRAN File_Name
--
--           Zero or more options may be used. The valid options
--           are given in the enumeration type Compiler_Options in
--           the Names package. Code between this and the next
--           COMPILE or FORTRAN statement or End of File is written to
--           the File_Name with the appropriate Ada or FORTRAN suffix.
--
--           Examples:
--
--               --* COMPILE T000000
--               --* COMPILE T030204 TIME_COMPILE COMPILER_LISTING
--               --* FORTRAN COMPFOR
--
--           EXECUTE:
--
--               --* EXECUTE Procedure_Name
```


--
-- Tells the parser to issue a script command to execute
-- the given procedure.
--

--
-- COMPARE:
--

--* COMPARE Compiler_Option_1 Result_File_1
--* COMPARE Compiler_Option_2 Result_File_2
--* COMPARE ...
--

-- This command is used to compare the performance of the
-- same Ada code but using different compiler options.
--

-- The COMPILE command must precede this command. The
-- COMPARE command is then used to compile the code created
-- by the COMPILE command using the specified compiler option.
-- Compilation speed, execution speed, and execution times
-- are then saved in the named Result_File.
--

-- This command is currently used only by test T000000 to
-- compare various compilation times. Identical Ada and
-- FORTRAN code segments are compared against each other.
-- The Ada code is compiled using four different options
-- with the four Result_File_?'s plus the FORTRAN results
-- being read in and saved in one file.
--

--
-- NEW_LIBRARY:
--

--* NEW_LIBRARY
--

-- If a library exists it is removed. A new library without
-- any files is then created.
--

--
-- Special Case: Multiple Compile Statement
--

-- If the size of an expanded file is too large,
-- it may exceed the file capacities of some systems.
-- For this reason, the COMPILE command may be embedded
-- just after a first level loop statement of the
-- EXPAND tool. For example:
--

--! LOOP 10 [1]
--* COMPILE TEMP
--! LOOP 1000 [2]
Large code fragment
--! END [2]
--! END [1]
--

-- In this case, 10 files would be created from this code
-- fragment. The file would contain:
--

-- File TEMP1:
--! LOOP 1 START 1 STEP 1 [1]
--! LOOP 1000 [2]
Large code fragment
--! END [2]
--! END [1]
--

-- File TEMP2:
--! LOOP 1 START 2 STEP 1 [1]
--! LOOP 1000 [2]
Large code fragment
--! END [2]
--! END [1]
--

-- etc ...
--

-- File TEMP10:
--! LOOP 1 START 10 STEP 1 [1]
--! LOOP 1000 [2]
Large code fragment
--

Source File: PARSE_.ADA

```
--          --! END [2]
--          --! END [1]
--          In addition, the script file will contain commands to
--          compile each of these 10 files in order.
--
PACKAGE Parse IS
  Parse_Error : EXCEPTION;

  PROCEDURE Parse_Tool( Input_File : String; Output_File : String );
    -- The test data is read from the Input_File test file.
    -- A script file is created and written to Output_File.
    -- In addition, the Ada/FORTRAN code or Ada/FORTRAN code templates are
    -- are written to separate files. Each test may cause more than one
    -- code file to be created.

    -- The first line of the Input_File must look like:
    --      -- Test_Number      e.g.      -- T010100

    -- Each code segment must begin with a COMPILE statement, FORTRAN
    -- statement, or special multiple COMPILE statement as defined above.

    -- NEW_LIBRARY may be placed at the beginning, between code blocks,
    -- or at the end of a test. More than one NEW_LIBRARY may be used
    -- per test.
END Parse;
```

Source File: PARSE.ADA

```
--
--
--               The Aerospace Corporation
--
-- Production Quality Ada Compiler Test Suite Support Software
--
--
-- Author:   BAP
-- Date:    10/01/88
-- File:    Parse.Ada
-- Component: Package Body Parse
-- Description: ( See package specification description )
--
```

```
WITH Names;      -- Enumeration Declarations
WITH Twine;      -- String Manipulation Package
WITH Script;     -- Controls Output to the Script File
WITH Syntax;     -- Meta Symbol Parsing Package
WITH Common;     -- Interface to Compiler Specific Information and Status
WITH PQAC_IO;    -- Centralized Input and Output Package
```

PACKAGE BODY Parse IS

```
Limit : CONSTANT Natural := 1000; -- Limit on input file lines

TYPE Line_Number_List IS ARRAY( Positive RANGE <> ) OF Natural;

TYPE Text_Type( Maximum : Natural := 0 ) IS RECORD
  Size : Natural := 0;
  Text : Twine.Series_List( 1 .. Maximum );
  Save : Line_Number_List( 1 .. Maximum ) := ( OTHERS => 0 );
END RECORD;

TYPE Save_List( Maximum : Natural := 0 ) IS RECORD
  Size : Natural := 0;
  List : Script.Option_List( 1 .. Maximum );
  Save : Line_Number_List( 1 .. Maximum ) := ( OTHERS => 0 );
END RECORD;

Big_Line   : CONSTANT Twine.Output_Buffer := ( OTHERS => '_' );
Meta_Lines : ARRAY( 1 .. Limit ) OF Syntax.Process_Value;

Original : Text_Type( Limit ); -- Stores the original text.
Capital  : Text_Type( Limit ); -- Stores the original text capitalized.
```

```
FUNCTION "="( A, B : Syntax.Process_Value ) RETURN Boolean RENAMES Syntax."=";
FUNCTION "="( A, B : Names.File_Category ) RETURN Boolean RENAMES Names."=";
FUNCTION "="( A, B : Common.Library_Status ) RETURN Boolean RENAMES Common."=";
```

```
FUNCTION "&"( A : Twine.Series; B : Twine.Series ) RETURN String IS
BEGIN
  RETURN Twine.Image( A ) & Twine.Image( B );
END "&";
```

```
FUNCTION "&"( A : String; B : Twine.Series ) RETURN String IS
BEGIN
  RETURN A & Twine.Image( B );
END "&";
```

```
FUNCTION "&"( A : Twine.Series; B : String ) RETURN String IS
BEGIN
  RETURN Twine.Image( A ) & B;
END "&";
```

```

FUNCTION "&"( A : String; B : Common.System_Attributes ) RETURN String IS
BEGIN
    RETURN A & Common.Image( B );
END "&";

```

```

PROCEDURE Process_Error
( Message : String;
  Line    : String := "";
  Number   : Natural := 0;
  Position : Natural := 0 ) IS
    Blanks : CONSTANT Twine.Input_Buffer := ( OTHERS => ' ' );
BEGIN
    PQAC_IO.Record_Error( "" );
    PQAC_IO.Record_Error( "Error in Test Parser:" );
    PQAC_IO.Record_Error( "" );
    PQAC_IO.Record_Error( Message );
    IF Number /= 0 THEN
        PQAC_IO.Record_Error( "Line Number: " & Twine.Image( Number, 5 ) );
    END IF;
    IF Line /= "" THEN
        PQAC_IO.Record_Error( Line );
    END IF;
    IF Position /= 0 THEN
        PQAC_IO.Record_Error
        ( Blanks( 1 .. Position - 1 ) & "-> Offending Item" );
    END IF;
    PQAC_IO.Record_Error( "" );
    RAISE Parse_Error;
END Process_Error;

```

```

PROCEDURE Process_Error
( Message : String;
  Index    : Positive;
  Position : Natural := 0 ) IS
BEGIN
    Process_Error
    ( Message, Original.Text( Index ) & "",
      Original.Save( Index ), Position );
END Process_Error;

```

```

PROCEDURE Store
( Buffer : IN OUT Text_Type;
  Line   : String;
  Save   : Natural := 0 ) IS
    -- Saves the Line with original line number Save in the Buffer.
BEGIN
    IF Buffer.Size = Buffer.Maximum THEN
        Process_Error( "Storage space exceeded.", Line, Save );
    END IF;
    Buffer.Size := Buffer.Size + 1;
    Buffer.Text( Buffer.Size ) := Twine.Create( Line );
    Buffer.Save( Buffer.Size ) := Save;
END Store;

```

```

PROCEDURE Store
( Buffer : IN OUT Text_Type;
  Line   : Twine.Series;
  Save   : Natural := 0 ) IS
    -- Saves the line with original line number Save in the Buffer.
BEGIN
    Store( Buffer, Line & "", Save );

```

END Store;

PROCEDURE Store

(Buffer : IN OUT Save_List;
Item : String;
Save : Natural := 0) IS

-- Saves the option of the Item in the Buffer.

BEGIN

IF Buffer.Size = Buffer.Maximum THEN
Process_Error("Storage space exceeded.", Save);
END IF;
Buffer.Size := Buffer.Size + 1;
Buffer.List(Buffer.Size) := Common.Option_Of(Item);
Buffer.Save(Buffer.Size) := Save;
EXCEPTION
WHEN Common.Undefined_Error =>
Process_Error("Unknown Compiler Option " & Item, Save);

END Store;

FUNCTION List_Of(Group : Save_List) RETURN Script.Option_List IS

BEGIN

RETURN Group.List(1 .. Group.Size);

END List_Of;

FUNCTION List_Of(Group : Text_Type) RETURN Twine.Series_List IS

BEGIN

RETURN Group.Text(1 .. Group.Size);

END List_Of;

FUNCTION Word(Number : Natural; Line : Twine.Series) RETURN String IS

-- Returns the Nth word in the Line, separated by non letters and digits.

Pairs : Twine.Bounds_List(1 .. Number);

BEGIN

Twine.Next_Words(Line, Pairs);
RETURN Twine.Substring(Line, Pairs(Number));

END Word;

FUNCTION Word(Number : Natural; Index : Positive) RETURN String IS

BEGIN

IF Index > Capital.Size THEN
Process_Error("Index out of bounds.", Index);

END IF;

RETURN Word(Number, Capital.Text(Index));

END Word;

PROCEDURE Read_In_Test(File_Name : String) IS

-- The input file is read into the buffer. If a select statement is
-- found that does not contain the name of the current compiler, then
-- the text is ignored until the select end statement is found.
-- For example, if the current compiler is Compiler_1, then Hello There
-- will be included in the input buffer. If the current compiler is
-- Compiler_2 then the Hello There will be ignored. The begin select
-- end end Select statements are not included in the input buffer.

--
-- --* BEGIN Compiler_1
-- Hello There
-- --* END

--

```

Input      : PQAC_IO.File_Type;
Buffer_1   : Twine.Input_Buffer;
Buffer_2   : Twine.Input_Buffer;
Last       : Natural := 0;
Head       : Natural := 0;
Tail       : Natural := 0;
Error_Line : Natural := 0;
Do_Print   : Boolean := True;
Entered    : Boolean := False;
Kind       : Syntax.Process_Value;

```

```

FUNCTION Current_Compiler_In( Text : String ) RETURN Boolean IS
  Pair : Twine.Bounds := ( 1, Text'FIRST - 1 );
BEGIN
  Twine.Next_Word( Text, Pair.Tail + 1, Pair );
  Twine.Next_Word( Text, Pair.Tail + 1, Pair );
  LOOP
    Twine.Next_Word( Text, Pair.Tail + 1, Pair );
    EXIT WHEN Pair.Head > Pair.Tail;
    IF Common.Is_Current_Compiler( Twine.Substring( Text, Pair )) THEN
      RETURN True;
    END IF;
  END LOOP;
  RETURN False;
EXCEPTION
  WHEN OTHERS =>
    Process_Error
      ( "Undefined Compiler: " & Twine.Substring( Text, Pair ),
        Buffer_1( 1 .. Last ), Error_Line, Pair.Head );
END Current_Compiler_In;

```

```

BEGIN
  PQAC_IO.Open_Input( Input, File_Name );
  WHILE NOT PQAC_IO.End_Of_File( Input ) LOOP
    Error_Line := Error_Line + 1;
    PQAC_IO.Get_Line( Input, Buffer_1, Last );
    Buffer_2 := Buffer_1;
    Twine.Upper_Case( Buffer_2( 1 .. Last ) );
    Kind := Syntax.Process_Value_Of( Buffer_2( 1 .. Last ) );
    CASE Kind IS
      WHEN Syntax.Begin_Select =>
        IF Entered THEN
          Process_Error
            ( "Missing End Select Statement.",
              Buffer_1( 1 .. Last ), Error_Line );
        END IF;
        Entered := True;
        Do_Print := Current_Compiler_In( Buffer_2( 1 .. Last ) );
      WHEN Syntax.End_Select =>
        IF NOT Entered THEN
          Process_Error
            ( "Missing Begin Select Statement.",
              Buffer_1( 1 .. Last ), Error_Line );
        END IF;
        Entered := False;
        Do_Print := True;
      WHEN Syntax.In_Error =>
        Process_Error
          ( "Unknown Command.",
            Buffer_1( 1 .. Last ), Error_Line );
      WHEN OTHERS =>
        IF Do_Print THEN
          Store( Original, Buffer_1( 1 .. Last ), Error_Line );
          Store( Capital, Buffer_2( 1 .. Last ), Error_Line );
          Meta_Lines( Original.Size ) := Kind;
        END IF;
    END CASE;
  END LOOP;
  PQAC_IO.Close( Input );

```

```

    IF Entered THEN
        Process_Error
            ( "Missing End Select Statement.",
              Buffer_1( 1 .. Last ), Error_Line );
    END IF;
    IF Original.Size = 0 THEN
        Process_Error( "No text read from file " & File_Name & "." );
    END IF;
END Read_In_Test;

PROCEDURE Process_Comments( Last : OUT Natural ) IS
    -- Copies Leading comments to the output buffer.

    Next : Natural := 1;

    FUNCTION Comment( Line : Natural ) RETURN Boolean IS
    BEGIN
        RETURN Meta_Lines( Line ) = Syntax.Comment_Line;
    END Comment;

    FUNCTION Right_End( Line : Twine.Series ) RETURN String IS
        Pair : Twine.Bounds;
    BEGIN
        Twine.Next_Word( Line, 1, Pair );
        RETURN Twine.Clip( Twine.Substring
            ( Line, ( Pair.Tail + 1, Twine.Length( Line ) ) ) );
    END Right_End;

    PROCEDURE Show( Test, Compiler, Host, Target : String ) IS
    BEGIN
        Script.Print
            ( "Compiler: " & Compiler & " " &
              "Host: " & Host & " " &
              "Target: " & Target );
        Script.Print( "" );
        Script.Print( "Test Number " & Test );
    END Show;

    PROCEDURE Find_Test_Number( Current_Test : String ) IS
        Pairs : Twine.Bounds_List( 1 .. 2 );
    BEGIN
        IF NOT Comment( 1 ) THEN
            Process_Error( "Test Number Expected.", 1 );
        END IF;
        Twine.Next_Words( Capital.Text( 1 ), Pairs );
        IF Twine.Equal( Capital.Text( 1 ), Pairs( 2 ), Current_Test ) THEN
            Show( Current_Test,
                  Common.Image( Common.Current_Compiler ),
                  Common.Image( Common.Host_Machine ),
                  Common.Image( Common.Target_Machine ) );
        ELSE
            Process_Error
                ( "Test number missing or incorrect.", 1, Pairs( 1 ).Head );
        END IF;
    END Find_Test_Number;

BEGIN
    Find_Test_Number( Common.Image( Common.Current_Test ) );
    LOOP
        EXIT WHEN Next = Original.Size;
        EXIT WHEN NOT Comment( Next + 1 );
        Next := Next + 1;
        Script.Print( Right_End( Original.Text( Next ) ) );
    END LOOP;
    Last := Next;

```

Source File: PARSE.ADA

END Process_Comments;

PROCEDURE Create_Code_File

(File_Name : String;
 Head : Natural;
 Tail : Natural) IS

-- Creates a file with the given name and copies the output
-- text between the lines Head and Tail into it.

-- Every Expand tool Equivalence statement found is also copied to
-- the file, even if it isn't in one of the lines from Head to Tail.
-- This is because Expand Meta Symbols may require an equivalence
-- that was declared at the top of the file. Since the test files
-- may contain more than one code fragment, more than one code file
-- can be produced. The equivalences must be added to each separate
-- code file.

File : PQAC_IO.File_Type;

BEGIN

PQAC_IO.Open_Output(File, File_Name);

FOR Index IN 1 .. Head - 1 LOOP

 CASE Meta_Lines(Index) IS

 WHEN Syntax.Equivalence =>

 PQAC_IO.Put_Line(File, Original.Text(Index) & "");

 WHEN OTHERS => NULL;

 END CASE;

END LOOP;

FOR Index IN Head .. Tail LOOP

 CASE Meta_Lines(Index) IS

 WHEN Syntax.Normal_Text

 ! Syntax.Equivalence

 ! Syntax.Start_Loop

 ! Syntax.End_Loop

 ! Syntax.Comment_Line =>

 PQAC_IO.Put_Line(File, Original.Text(Index) & "");

 WHEN OTHERS => NULL;

 END CASE;

END LOOP;

PQAC_IO.Close(File);

END Create_Code_File;

PROCEDURE Parse_Meta_Symbols

(Head_Bound : Positive; -- First line in buffer to be parsed
 Tail_Bound : Natural; -- Last line in buffer to be parsed
 Code_Found : IN OUT Boolean; -- Set True if Ada or FORTRAN code found
 Support : IN OUT Boolean; -- Set True if Ada code is support package
 File_Name : IN OUT Twine.Series; -- Prefix of name to save code segment
 Suffix_1 : IN OUT Names.File_Category; -- Pre-Expand, EXP, ADA, or FOR
 -- Will only be EXP if Expand meta symbols found
 Suffix_2 : IN OUT Names.File_Category; -- Post-Expand, ADA or FOR
 Execute : IN OUT Text_Type; -- List of procedure names to execute
 Prefix : IN OUT Text_Type; -- List of file name prefixes from COMPARE statements
 Compare : IN OUT Save_List; -- List of compiler options from COMPARE statements
 -- Number of elements in Prefix and Compare will be the same
 Options : IN OUT Save_List) IS
 -- List of compiler options from COMPILE statement

Head : Natural := Head_Bound - 1;

Tail : Natural := Head_Bound - 1;

Found : Boolean := False;

FUNCTION Text_Code(Line : Positive) RETURN Boolean IS


```

-- Returns true if normal code not to be parsed
BEGIN
  CASE Meta_Lines( Line ) IS
    WHEN Syntax.Normal_Text
      ! Syntax.Comment_Line
      ! Syntax.Equivalence
      ! Syntax.Start_Loop
      ! Syntax.End_Loop => RETURN True;
    WHEN OTHERS => RETURN False;
  END CASE;
END Text_Code;

FUNCTION Short( Name : String ) RETURN String IS
  -- If the last character of the Name is ';' then remove it.
BEGIN
  IF Name'LENGTH > 0 AND THEN Name( Name'LAST ) = ';' THEN
    RETURN Name( Name'FIRST .. Name'LAST - 1 );
  ELSE
    RETURN Name;
  END IF;
END Short;

PROCEDURE Check_Support( Word_1, Word_2 : String ) IS
  -- Checks whether Word_2 is part of support software
BEGIN
  IF Word_1 = "WITH" AND THEN Common.Is_Support_Package( Word_2 ) THEN
    Support := True;
  END IF;
END Check_Support;

PROCEDURE Parse_Compile( Text : String; Line : Positive ) IS
  -- Call Syntax package procedure to parse line.
  -- Compile name bounds are returned in Name.
  -- Parameter name bounds are returned in Extra.
  Name : Twine.Bounds;
  Extra : Twine.Bounds_List( 1 .. Options.Maximum );
  Error : Natural := 0;
BEGIN
  Syntax.Parse_Compile_Name( Text, Name, Extra );
  File_Name := Twine.Create( Twine.Substring( Text, Name ) );
  FOR Index IN Extra'RANGE LOOP
    EXIT WHEN Extra( Index ).Head > Extra( Index ).Tail;
    Error := Extra( Index ).Head;
    Store( Options, Twine.Substring( Text, Extra( Index ) ), Error );
  END LOOP;
EXCEPTION
  WHEN Syntax.Name_Error =>
    Process_Error( "COMPILE Unit_Name expected.", Line );
  WHEN Common.Undefined_Error =>
    Process_Error( "Unknown Option.", Line, Error );
END Parse_Compile;

PROCEDURE Set_Found( Line : Natural; Suffix : Names.File_Category ) IS
BEGIN
  IF Found THEN -- Compile command already found
    Process_Error( "Compile Command Duplicated.", Line );
  END IF;
  Parse_Compile( Original.Text( Line ) & "", Line );
  Found := True;
  Suffix_1 := Suffix;
  Suffix_2 := Suffix;
END Set_Found;

BEGIN
  Code_Found := False;
  Support := False;
  WHILE Tail < Tail_Bound AND THEN NOT Text_Code( Tail + 1 ) LOOP
    -- Process all beginning lines containing a Parse meta symbol
    Tail := Tail + 1;

```

```

CASE Meta_Lines( Tail ) IS
  WHEN Syntax.Compile =>
    Set_Found( Tail, Names.Ada );
  WHEN Syntax.Fortran =>
    Set_Found( Tail, Names.Fortran );
  WHEN Syntax.Execute =>
    Store( Execute, Word( 3, Tail ), Tail );
  WHEN Syntax.Compare =>
    Store( Compare, Word( 3, Tail ), Tail );
    Store( Prefix, Word( 4, Tail ), Tail );
  WHEN OTHERS =>
    Process_Error( "Unexpected Command", Tail );
END CASE;
END LOOP;
Head := Tail;
WHILE Head < Tail_Bound AND THEN Text_Code( Head + 1 ) LOOP
  -- Process lines not containing a Parse meta symbol
  Head := Head + 1;
  CASE Meta_Lines( Head ) IS
    WHEN Syntax.Normal_Text => Code_Found := True;
    WHEN Syntax.Equivalence
      ! Syntax.Start_Loop      -- Expand must be called later
      ! Syntax.End_Loop        => Suffix_1 := Names.Expand;
    WHEN OTHERS                => NULL;
  END CASE;
  Check_Support( Word( 1, Head ), Short( Word( 2, Head ) ) );
END LOOP;
IF Head < Tail_Bound THEN
  Process_Error( "Unexpected Command", Tail_Bound );
END IF;
IF NOT Code_Found THEN
  IF Found OR ELSE Support OR ELSE Execute.Size > 0 THEN
    Process_Error( "No text to compile.", Head );
  END IF;
ELSIF NOT Found THEN
  Process_Error( "COMPILE command not found.", Head_Bound );
END IF;
END Parse_Meta_Symbols;

PROCEDURE Process_Single
( Head : Positive;
  Tail : Natural;
  List : Boolean := True ) IS

  -- A single block of Ada code or Expand Templates was found.
  -- This block is to be written to its own file.
  -- In addition, the block is to be examined for Parse meta symbols.
  -- These meta symbols are used to create the script file for the
  -- test.

  Maximum : CONSTANT Natural := 10;
  Code_Found : Boolean;
  Support : Boolean;
  File_Name : Twine.Series;
  Suffix_1 : Names.File_Category;
  Suffix_2 : Names.File_Category;
  Execute : Text_Type( Maximum );
  Prefix : Text_Type( Maximum );
  Compare : Save_List( Maximum );
  Options : Save_List( Maximum );

  FUNCTION ID( Name : String; Kind : Names.File_Category ) RETURN String
    RENAMES Common.Build_Name;

BEGIN
  Parse_Meta_Symbols
    ( Head_Bound => Head,
      Tail_Bound => Tail,
      Code_Found => Code_Found,
      Support => Support,
      File_Name => File_Name,
      Suffix_1 => Suffix_1,

```

```

        Suffix_2    => Suffix_2,
        Execute     => Execute,
        Prefix      => Prefix,
        Compare     => Compare,
        Options     => Options );
IF NOT Code_Found THEN
    RETURN;
END IF;
IF Suffix_2 = Names.Ada THEN -- Check library state
    IF Common.Library_State = Common.UnInitialized THEN
        Script.Keep( Names.Create_Library );
        Common.Set_Library_State( Common.Initialized );
    END IF;
END IF;
Create_Code_File( ID( File_Name & "", Suffix_1 ), Head, Tail );
IF Support AND THEN Common.Library_State = Common.Initialized THEN
    FOR Index IN 1 .. Common.Support_Size LOOP -- Compile support
        Script.Keep_Compile
            ( Common.Support_Package( Index ),
              Names.Ada, Script.No_Options, Delete_After => False );
    END LOOP;
    Common.Set_Library_State( Common.Support_Compiled ); -- Set state
END IF;
Script.Keep_Code_List( File_Name & "", Suffix_1, List ); -- Make File
IF Suffix_1 = Names.Expand THEN -- Send command to Expand
    Script.Keep_Expand( File_Name & "", Suffix_1, Suffix_2 );
END IF;
IF Compare.Size = 0 THEN
    -- No compares, do a simple compile
    Script.Keep_Compile( File_Name & "", Suffix_2, List_Of(Options),True);
    Script.Keep_Listings( File_Name & "", Suffix_2, List_Of( Options ) );
ELSE
    -- Are some compares, compile each type
    Script.Keep_Compare
        ( File_Name & "", Suffix_2, List_Of( Compare ), List_Of( Prefix ) );
END IF;
FOR Index IN 1 .. Execute.Size LOOP -- Send commands to execute files
    Script.Keep_Execute( Execute.Text( Index ) & "", Suffix_2 );
END LOOP;
END Process_Single;

```

PROCEDURE Process_Multiple

```

( Head : Natural;
  Last : Natural;
  Tail : OUT Natural ) IS

    -- See description of Special Case in package specification.
    -- This procedure is used when a COMPILE or FORTRAN statement
    -- is found directly after an Expand procedure LOOP statement.
    -- The LOOP must not be nested, i.e. level 1 ([1]). Example:
    --
    -- --! LOOP 4 STEP 2 START 5 [1]
    -- --* COMPILE TESTFILE ( options ... )
    -- PROCEDURE X[1] IS
    -- BEGIN
    --     NULL;
    -- END X[1];
    -- --! END [1]
    --
    -- This procedure will then manipulate the buffer and call Process_Single
    -- 4 times ( LOOP 4 ). Therefore, four separate code files will be
    -- created. This procedure is needed to work with file size limitations.
    --
    -- The files created for the above example will be:
    --
    -- --* COMPILE TESTFILE1 ( options ... )
    -- --! LOOP 1 START 5 [1]
    -- PROCEDURE X[1] IS
    -- BEGIN
    --     NULL;
    -- END X[1];
    -- --! END [1]

```

```
--
-- --* COMPILE TESTFILE2 ( options ... )
-- --! LOOP 1 START 7 [1]
-- PROCEDURE X[1] IS
-- BEGIN
--     NULL;
-- END X[1];
-- --! END [1]
--
-- --* COMPILE TESTFILE3 ( options ... )
-- --! LOOP 1 START 9 [1]
-- PROCEDURE X[1] IS
-- BEGIN
--     NULL;
-- END X[1];
-- --! END [1]
--
-- --* COMPILE TESTFILE4 ( options ... )
-- --! LOOP 1 START 11 [1]
-- PROCEDURE X[1] IS
-- BEGIN
--     NULL;
-- END X[1];
-- --! END [1]
--
```

```
Next      : Natural := Head + 1;
Level     : Natural := 1;
Save_Kind_1 : Syntax.Process_Value;
Save_Kind_2 : Syntax.Process_Value;
Save_line_1 : Twine.Series;
Save_line_2 : Twine.Series;
Value      : Integer;
Copies     : Positive;
Start      : Integer;
Step       : Integer;
Width      : Natural;
Name       : Twine.Series;
Options    : Twine.Series;
Position   : Natural;
```

```
PROCEDURE Parse_Compile( Line : String ) IS
    Pair : Twine.Bounds;
    Next : Twine.Bounds_List( 1 .. 1 );
BEGIN
    Syntax.Parse_Compile_Name( Line, Pair, Next );
    Name := Twine.Create( Twine.Substring( Line, Pair ) );
    IF Next( 1 ).Head > Next( 1 ).Tail THEN
        Options := Twine.Create( "" );
    ELSE
        Options := Twine.Create( Line( Next( 1 ).Head .. Line'LAST ) );
    END IF;
    Position := Pair.Tail + 1;
EXCEPTION
    WHEN Syntax.Name_Error =>
        Process_Error( "COMPILE Unit_Name expected.", Head + 1 );
END Parse_Compile;
```

```
PROCEDURE Parse_A_Loop( Line : String ) IS
BEGIN
    Syntax.Parse_Loop( Line, Copies, Start, Step, Width );
EXCEPTION
    WHEN Syntax.Count_Error =>
        Process_Error( "Iteration step must be non-zero.", Head );
    WHEN Syntax.Step_Error =>
        Process_Error( "Iteration step must be non-zero.", Head );
    WHEN Syntax.Range_Error =>
        Process_Error( "Range of loop must be non-negative.", Head );
    WHEN Syntax.Name_Error =>
        Process_Error( "Identifier not defined.", Head );
    WHEN Syntax.Value_Error =>
```

```

        Process_Error( "Integer value expected here.", Head );
    END Parse_A_Loop;

BEGIN
    LOOP
        Next := Next + 1;
        IF Next >= Last THEN
            Process_Error( "Compile Loop command not closed.", Last );
        END IF;
        CASE Meta_Lines( Next ) IS
            WHEN Syntax.Start_Loop =>
                Level := Level + 1;
            WHEN Syntax.End_Loop =>
                Level := Level - 1;
                Tail := Next;
                EXIT WHEN Level = 0;
            WHEN Syntax.Compile ! Syntax.Fortran =>
                Process_Error( "Previous Compile Loop not closed.", Next );
            WHEN OTHERS => NULL;
        END CASE;
    END LOOP;
    Parse_A_Loop( Original.Text( Head ) & "" );
    Parse_Compile( Original.Text( Head + 1 ) & "" );
    Value := Start;
    Save_Line_1 := Original.Text( Head );      -- Save information from
    Save_Line_2 := Original.Text( Head + 1 );  -- the two lines that are
    Save_Kind_1 := Meta_Lines( Head );          -- are changed
    Save_Kind_2 := Meta_Lines( Head + 1 );
    Meta_Lines( Head ) := Meta_Lines( Head + 1 );
    Meta_Lines( Head + 1 ) := Syntax.Start_Loop;
    IF Meta_Lines( Head ) = Syntax.Compile THEN
        Original.Text( Head ) := Twine.Create
            ( "---x COMPILE " & Name & Twine.Zeroed_Image( Value, Width )
              & " " & Options );
    ELSIF Meta_Lines( Head ) = Syntax.Fortran THEN
        Original.Text( Head ) := Twine.Create
            ( "---x FORTRAN " & Name & Twine.Zeroed_Image( Value, Width )
              & " " & Options );
    ELSE
        Process_Error( "COMPILE Unit_Name Expected.", Head );
    END IF;
    Original.Text( Head + 1 ) := Twine.Create
        ( "---! LOOP 1 START " & Twine.Image( Value, Width ) & " [1]" );
    FOR Index IN 1 .. Copies LOOP
        Twine.Copy
            ( Original.Text( Head ),
              ( Position, Position + Width - 1 ),
              Twine.Zeroed_Image( Value, Width ) );
        Twine.Copy
            ( Original.Text( Head + 1 ),
              ( 18, 18 + Width - 1 ),
              Twine.Image( Value, Width ) );
        Process_Single( Head, Next, Index = 1 );
        Value := Value + Step;
    END LOOP;
    Original.Text( Head ) := Save_Line_1;      -- Restore the two lines
    Original.Text( Head + 1 ) := Save_Line_2;  -- that were changed
    Meta_Lines( Head ) := Save_Kind_1;
    Meta_Lines( Head + 1 ) := Save_Kind_2;
END Process_Multiple;

```

```

PROCEDURE Process_Files( Next : Natural; Last : Natural ) IS

```

```

    Head : Natural := Next;
    Tail : Natural := Next - 1;

```

```

FUNCTION Blanks( Line : Natural ) RETURN Boolean IS
    Temp : CONSTANT Twine.Input_Buffer := ( OTHERS => ' ' );
    Size : CONSTANT Natural := Twine.Length( Original.Text( Line ) );
BEGIN

```

```

    RETURN Twine.Equal( Original.Text( Line ), Temp( 1 .. Size ) );
END Blanks;

```

```

PROCEDURE Remove_Library IS
BEGIN

```

```

    IF Common.Library_State /= Common.UnInitialized THEN
        Script.Keep( Names.Remove_Library );
        Common.Set_Library_State( Common.UnInitialized );
    END IF;
END Remove_Library;

```

```

PROCEDURE Load_Equivalences IS

```

```

    -- Every line containing EXPAND equivalences is found.
    -- Each of these lines must be output to every code file
    -- created, even for multiple files.

```

```

    Error : Natural;

```

```

BEGIN

```

```

    FOR Index IN Next .. Last LOOP

```

```

        Error := Index;

```

```

        IF Meta_Lines( Index ) = Syntax.Equivalence THEN

```

```

            Syntax.Parse_Equivalence( Original.Text( Index ) & " " );

```

```

        END IF;

```

```

    END LOOP;

```

```

EXCEPTION

```

```

    WHEN Syntax.Statement_Error =>

```

```

        Process_Error( "Reserved word IS not found.", Error );

```

```

    WHEN Syntax.Capacity_Error =>

```

```

        Process_Error( "Exceeded equivalence capacities.", Error );

```

```

    WHEN Syntax.Duplicate_Error =>

```

```

        Process_Error( "Equivalence name used twice.", Error );

```

```

    WHEN Syntax.Name_Error =>

```

```

        Process_Error( "Identifier not defined.", Error );

```

```

    WHEN Syntax.Value_Error =>

```

```

        Process_Error( "Integer value expected here.", Error );

```

```

END Load_Equivalences;

```

```

BEGIN

```

```

    Load_Equivalences;

```

```

    LOOP

```

```

        EXIT WHEN Tail = Last;

```

```

        Head := Tail + 1;

```

```

        CASE Meta_Lines( Head ) IS

```

```

            WHEN Syntax.Equivalence =>

```

```

                Tail := Head;

```

```

            WHEN Syntax.New_library =>

```

```

                Remove_Library;

```

```

                Tail := Head;

```

```

            WHEN Syntax.Start_Loop =>

```

```

                -- Next line must be COMPILE or FORTRAN

```

```

                IF Head >= Last OR ELSE

```

```

                    ( Meta_Lines( Head + 1 ) /= Syntax.Compile AND THEN

```

```

                      Meta_Lines( Head + 1 ) /= Syntax.Fortran ) THEN

```

```

                        Process_Error( "COMPILE Unit_Name Expected.", Head );

```

```

                END IF;

```

```

                Process_Multiple( Head, Last, Tail );

```

```

            WHEN Syntax.Compile ! Syntax.Fortran =>

```

```

                -- Find next COMPILE or FORTRAN or End of File and process.

```

```

                LOOP

```

```

                    Tail := Tail + 1;

```

```

                    EXIT WHEN Tail = Last;

```

```

                    EXIT WHEN Meta_Lines( Tail + 1 ) = Syntax.Compile;

```

```

                    EXIT WHEN Meta_Lines( Tail + 1 ) = Syntax.Fortran;

```

```

                    EXIT WHEN Meta_Lines( Tail + 1 ) = Syntax.New_Library;

```

```

                    EXIT WHEN Tail + 1 < Last AND THEN

```

```

                        Meta_Lines( Tail + 1 ) = Syntax.Start_Loop AND THEN

```

```

                        ( Meta_Lines( Tail + 2 ) = Syntax.Compile OR ELSE

```

```

                          Meta_Lines( Tail + 2 ) = Syntax.Fortran );

```

```

                        -- This is if next group is a multiple compile

```

```

                END LOOP;

```

```

                Process_Single( Head, Tail );

```

```

            WHEN OTHERS =>

```

Source File: PARSE.ADA

```

        IF Blanks( Head ) THEN
            Tail := Head;
        ELSE
            Process_Error( "COMPILE Unit_Name Expected.", Head );
        END IF;
    END CASE;
END LOOP;
Script.Print( "" );
Script.Print( "Test " & Common.Current_Test & " Completed" );
Script.Print( Big_Line );
Script.Print( "" );
END Process_Files;

PROCEDURE Parse_Tool( Input_File : String; Output_File : String ) IS
    Last : Natural := 0;
BEGIN
    -- Input_File : Name of test file to be parsed.
    -- Output_File : Name of script file to be created.

    Read_In_Test( Input_File );

    -- Read_In_Test : The Input_File is read in and saved. Text between
    -- Begin_Select and End_Select that is not selected for the current
    -- compiler is ignored.

    Process_Comments( Last );

    -- Process_Comments : The beginning comments of the test file are
    -- copied to the script file buffer. The test name expected in
    -- the first line of the test file is verified. The return value
    -- Last is the last line of the Original buffer that was processed.

    Process_Files( Last + 1, Original.Size );

    -- Process_File : The remaining text in the file is processed.
    -- This is composed of one or more segments of Ada test code
    -- with embedded code expander and parser meta symbols. If there
    -- are more than one segment, they are separated with the Compile
    -- meta command. This Compile command may possibly be the first
    -- command after an unnested code Expander meta loop construct.
    -- This allows multiple segments to be declared with the same code.
    -- For each segment, a temporary file is created. If the code needs
    -- to be sent through the code expander then commands to do this
    -- are issued.

    Script.Output_Script( Output_File );

    -- Create_Script_File : The Script buffer is written to the given file.
END Parse_Tool;

END Parse;
```

Source File: PQAC_IO_.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:  BAP
--       Date:    10/01/88
--       File:    PQAC_IO_.Ada
--       Component: Package Specification PQAC_IO
--       Description: Centralized Input and Output Package.
--                   Instead of using Text_IO, this package is used to
--                   allow the redirection of input and output if needed.
--                   There are also several utility function provided here.
--
```

PACKAGE PQAC_IO IS

TYPE File_Type IS LIMITED PRIVATE;

File_Error : EXCEPTION;

PROCEDURE Open_Input(File : IN OUT File_Type; Name : String);
PROCEDURE Open_Output(File : IN OUT File_Type; Name : String);
PROCEDURE Close(File : IN OUT File_Type);

PROCEDURE Delete_File(Name : String);

PROCEDURE Put(File : File_Type; Text : String);
PROCEDURE Put_Line(File : File_Type; Text : String);
PROCEDURE New_Line(File : File_Type);

PROCEDURE Get(File : File_Type; Text : OUT String);
PROCEDURE Get_Line(File : File_Type; Text : OUT String; Last : OUT Natural);

PROCEDURE Get(File : File_Type; Item : OUT Float);
PROCEDURE Get(File : File_Type; Item : OUT Integer);

PROCEDURE Put_Value(File : String; Item : Float); -- Puts single value
PROCEDURE Put_Value(File : String; Item : Integer); -- Puts single value

PROCEDURE Get_Value(File : String; Item : OUT Float); -- Gets single value
PROCEDURE Get_Value(File : String; Item : OUT Integer); -- Gets single value

PROCEDURE Get(From : String; Item : OUT Float; Last : OUT Positive);
PROCEDURE Get(From : String; Item : OUT Integer; Last : OUT Positive);

PROCEDURE Put(Text : String);
PROCEDURE Put_Line(Text : String);
PROCEDURE New_Line;

PROCEDURE Get_Line(Prompt : String; Text : OUT String; Last : OUT Natural);
-- Returns input from the keyboard.

PROCEDURE Append(File_Name : String; Text : String);
-- Appends one line of text to the screen.

PROCEDURE Record_Error(Message : String);
-- Displays error message.

FUNCTION End_Of_File(File : File_Type) RETURN Boolean;

GENERIC
TYPE Enum IS (<>);
PACKAGE Enumeration_IO IS

PROCEDURE Get(File : File_Type; Item : OUT Enum);
PROCEDURE Get(From : String; Item : OUT Enum; Last : OUT Positive);

END Enumeration_IO;

Source File: PQAC_IO_.ADA

PRIVATE

TYPE File_Descriptor;

TYPE File_Type IS ACCESS File_Descriptor;

END PQAC_IO;

Source File: PQAC_IO.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    PQAC_IO.Ada
--       Component: Package Body PQAC_IO
--       Description: Centralized Input and Output Package
--
```

```
WITH Twine;    -- String Manipulation Package
WITH Text_IO;
```

```
PACKAGE BODY PQAC_IO IS
```

```
TYPE File_Descriptor IS RECORD
  File : Text_IO.File_Type;
END RECORD;
```

```
PACKAGE Flt_IO IS NEW Text_IO.Float_IO( Float );
PACKAGE Int_IO IS NEW Text_IO.Integer_IO( Integer );
```

```
PROCEDURE Open_Input( File : IN OUT File_Type; Name : -String ) IS
BEGIN
  IF File = NULL THEN
    File := NEW File_Descriptor;
  END IF;
  Text_IO.Open( File.File, Text_IO.In_File, Name );
  Text_IO.Reset( File.File );
EXCEPTION
  WHEN OTHERS =>
    Record_Error( "Error opening " & Name & " as input." );
    RAISE File_Error;
END Open_Input;
```

```
PROCEDURE Open_Output( File : IN OUT File_Type; Name : String ) IS
BEGIN
  IF File = NULL THEN
    File := NEW File_Descriptor;
  END IF;
  Text_IO.Create( File.File, Text_IO.Out_File, Name );
  Text_IO.Reset( File.File );
EXCEPTION
  WHEN OTHERS =>
    Record_Error( "Error opening " & Name & " as output." );
    RAISE File_Error;
END Open_Output;
```

```
PROCEDURE Close( File : IN OUT File_Type ) IS
BEGIN
  Text_IO.Close( File.File );
EXCEPTION
  WHEN OTHERS =>
    Record_Error( "Error closing a file." );
    RAISE File_Error;
END Close;
```

```
PROCEDURE Delete_File( Name : String ) IS
  File : Text_IO.File_Type;
BEGIN
  Text_IO.Open( File, Text_IO.In_File, Name );
```

Source File: PQAC_IO.ADA

```
Text_IO.Delete( File );  
EXCEPTION  
  WHEN OTHERS => NULL;  
END Delete_File;
```

```
PROCEDURE Put( File : File_Type; Text : String ) IS  
BEGIN  
  Text_IO.Put( File.File, Text );  
EXCEPTION  
  WHEN OTHERS => RAISE File_Error;  
END Put;
```

```
PROCEDURE Put_Line( File : File_Type; Text : String ) IS  
BEGIN  
  Text_IO.Put_Line( File.File, Text );  
EXCEPTION  
  WHEN OTHERS => RAISE File_Error;  
END Put_Line;
```

```
PROCEDURE New_Line( File : File_Type ) IS  
BEGIN  
  Text_IO.New_Line( File.File );  
EXCEPTION  
  WHEN OTHERS =>  
    Record_Error( "Error in file." );  
    RAISE File_Error;  
END New_Line;
```

```
PROCEDURE Get( File : File_Type; Text : OUT String ) IS  
BEGIN  
  Text_IO.Get( File.File, Text );  
EXCEPTION  
  WHEN OTHERS => RAISE File_Error;  
END Get;
```

```
PROCEDURE Get_Line( File : File_Type; Text : OUT String; Last : OUT Natural ) IS  
BEGIN  
  Text_IO.Get_Line( File.File, Text, Last );  
EXCEPTION  
  WHEN OTHERS => RAISE File_Error;  
END Get_Line;
```

```
PROCEDURE Get( File : File_Type; Item : OUT Float ) IS  
BEGIN  
  Flt_IO.Get( File.File, Item );  
EXCEPTION  
  WHEN OTHERS => RAISE File_Error;  
END Get;
```

```
PROCEDURE Get( File : File_Type; Item : OUT Integer ) IS  
BEGIN  
  Int_IO.Get( File.File, Item );  
EXCEPTION  
  WHEN OTHERS => RAISE File_Error;  
END Get;
```

```
PROCEDURE Put_Value( File : String; Item : Float ) IS  
  Output : File_Type;
```

Source File: PQAC_IO.ADA

```
BEGIN
  Open_Output( Output, File );
  Flt_IO.Put( Output.File, Item, 8, 4 );
  Close( Output );
END Put_Value;
```

```
PROCEDURE Put_Value( File : String; Item : Integer ) IS
  Output : File_Type;
BEGIN
  Open_Output( Output, File );
  Int_IO.Put( Output.File, Item, 3 );
  Close( Output );
END Put_Value;
```

```
PROCEDURE Get_Value( File : String; Item : OUT Float ) IS
  Input : File_Type;
BEGIN
  Open_Input( Input, File );
  Get( Input, Item );
  Close( Input );
END Get_Value;
```

```
PROCEDURE Get_Value( File : String; Item : OUT Integer ) IS
  Input : File_Type;
BEGIN
  Open_Input( Input, File );
  Get( Input, Item );
  Close( Input );
END Get_Value;
```

```
PROCEDURE Get( From : String; Item : OUT Float; Last : OUT Positive ) IS
BEGIN
  Flt_IO.Get( From, Item, Last );
EXCEPTION
  WHEN OTHERS => RAISE File_Error;
END Get;
```

```
PROCEDURE Get( From : String; Item : OUT Integer; Last : OUT Positive ) IS
BEGIN
  Int_IO.Get( From, Item, Last );
EXCEPTION
  WHEN OTHERS => RAISE File_Error;
END Get;
```

```
PROCEDURE Put( Text : String ) IS
BEGIN
  Text_IO.Put( Text );
END Put;
```

```
PROCEDURE Put_Line( Text : String ) IS
BEGIN
  Text_IO.Put_Line( Text );
END Put_Line;
```

```
PROCEDURE New_Line IS
BEGIN
  Text_IO.New_Line;
END New_Line;
```

Source File: PQAC_IO.ADA

```
PROCEDURE Get_Line( Prompt : String; Text : OUT String; Last : OUT Natural) IS
BEGIN
    Text_IO.Put( Prompt );
    Text_IO.Get_Line( Text, Last );
END Get_Line;
```

```
FUNCTION End_Of_File( File : File_Type ) RETURN Boolean IS
BEGIN
    RETURN Text_IO.End_Of_File( File.File );
EXCEPTION
    WHEN OTHERS => RAISE File_Error;
END End_Of_File;
```

```
PROCEDURE Append( File_Name : String; Text : String ) IS
```

```
    File : Text_IO.File_Type;
    Save : Twine.Series_List( 1 .. 1000 );
    Last : Natural := 0;
```

```
    PROCEDURE Load_File IS
        Input : Text_IO.File_Type;
        Buffer : Twine.Input_Buffer;
        Size : Natural := 0;
    BEGIN
        Text_IO.Open( Input, Text_IO.In_File, File_Name );
        WHILE NOT Text_IO.End_Of_File( Input ) LOOP
            Text_IO.Get_Line( Input, Buffer, Size );
            Last := Last + 1;
            Save( Last ) := Twine.Create( Buffer( 1 .. Size ) );
        END LOOP;
        Text_IO.Close( Input );
    EXCEPTION
        WHEN OTHERS => NULL;
    END Load_File;
```

```
    PROCEDURE Open_File IS
    BEGIN
        Text_IO.Open( File, Text_IO.Out_File, File_Name );
        FOR Index IN 1 .. Last LOOP
            Text_IO.Put_Line( File, Twine.Image( Save( Index ) ) );
        END LOOP;
    EXCEPTION
        WHEN OTHERS =>
            Text_IO.Create( File, Text_IO.Out_File, File_Name );
    END Open_File;
```

```
BEGIN
    Load_File;
    Open_File;
    Text_IO.Put_Line( File, Text );
    Text_IO.Close( File );
END Append;
```

```
PROCEDURE Record_Error( Message : String ) IS
BEGIN
    Text_IO.Put_Line( Message );
END Record_Error;
```

```
PACKAGE BODY Enumeration_IO IS
```

Source File: PQAC_IO.ADA

```
PACKAGE Enum_IO IS NEW Text_IO Enumeration_IO( Enum );

PROCEDURE Get( File : File_Type; Item : OUT Enum ) IS
BEGIN
    Enum_IO.Get( File.File, Item );
EXCEPTION
    WHEN OTHERS => RAISE File_Error;
END Get;

PROCEDURE Get( From : String; Item : OUT Enum; Last : OUT Positive ) IS
BEGIN
    Enum_IO.Get( From, Item, Last );
EXCEPTION
    WHEN OTHERS => RAISE File_Error;
END Get;

END Enumeration_IO;

END PQAC_IO;
```

Source File: RATING_.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Rating_.Ada
--       Component: Package Specification Rating
--       Description: Contains a procedure for producing a compiler rating
--                   based on observed results.
--
```

PACKAGE Rating IS

Rating_Error : EXCEPTION;

PROCEDURE Rating_Tool

(Weight_Table : String;
 Results_File : String;
 Rating_Output : String);

-- The Weight_Table file contains a list of all of the test names,
-- with a weight and method of assigning points to each test.

-- The Results_File contains a list of all of the results of the
-- execution of the PQAC test suite.

-- Results from the evaluation are written to the Rating_Output file.

-- Weight_Table:

-- Field 1: Test Number; 7 Characters, first character 'T'
-- Field 2: Minimal Test; 1 Character, either 'M' or ' '
-- Field 3: Test Weight; Integer range 0 to 100
-- Field 4: Point Cutoff Percent; Integer range 0 to 100

-- The same Weight_Table file should be used for different compilers
-- that are to be compared against each other. Tests that are designated
-- as minimal by the report should have an 'M' in field 2.

-- The test weight in field 3 may be 0 for those tests that are simply
-- definitions or for tests such as T000000 whose results are used by
-- other tests.

-- Test point cutoff percent in field 4 is used for assigning points
-- after a test has completed. The value represents the base percent
-- of success for awarding points to a test. If the point cutoff is
-- 100, then a test must pass 100% to get the full weight, otherwise
-- it will be awarded 0 points. If the point cutoff is 0, then the
-- straight pass percentage of the weights will be awarded. If the
-- point cutoff is somewhere in between, such as 50, then the test
-- must pass by MORE than 50% to get any points. So if the point cutoff
-- is 50% and the test passes by 75%, then it is awarded half of the
-- tests weight. Values are not rounded up. If the point cutoff is
-- 75%, total point 10, and the test passes by 77%, then 0 points would
-- be awarded. If the test passed by 78%, then 1 point would be awarded.

-- Selected Points Awarded From Total of 10

	Point Cutoff %				
	0%	25%	50%	75%	100%
Test Pass %					
0%	0	0	0	0	0
25%	2	0	0	0	0
50%	5	3	0	0	0
75%	7	6	5	0	0
100%	10	10	10	10	10

```

--
--              Fields
--          111111123334444
-- Example -->T000000 0 100
-- File    -->T010100 0 100
-- Contents -->T020401M 10 50
--          -->T020402M 10 50
--          -->T030103 2 0
--
-- Results_File:
--
-- The first line of the file should contain the name of the compiler.
-- Each line after that will contain the field described here.
-- Any lines after the first that do not contain a test number in the
-- first columns will be ignored. If multiple lines for the same
-- test number are encountered, a message to that effect will be printed
-- and the latest value for the test results will be used.
-- If a test is omitted from this file, this will be indicated in
-- the Rating_Output file.
--
-- Field 1: Test Number; 7 Characters, first character 'T'
-- Field 2: Test Pass Percent; Integer from 0 .. 100 or Special Code
-- Field 3: Test Comment; Up to 60 characters
--
-- Special Codes:
-- "xxx" Test was not run, definition or not applicable
-- "???" Problem with the test, must be examined
-- "... " Test results must be manually interpreted
-- "==" duplicated, comment contains name of duplicate test
--
--              Fields
--          1111111 222 333...
-- Example -->DEC VAX V1.4 Ada Compiler
-- File    -->T010100 xxx Definition.
-- Contents -->T020402 85
--          -->T030310 == T030309
--          -->T040101 ...
--          -->T060503 ??? Times not repeatable.
--
-- Rating_Output:
--
-- The Weight_Table and Results_File files are read in and the
-- Rating_Output file is created. This file contains a header,
-- list of individual statistics, and summary information.
--
-- Examples of the individual lines are:
--
-- Num      Test      Weight Score      %      Comments:
--
-- 1. T000000      0      0      100
-- 2. T010100      0      N/A      N/A      Definition.
-- 3. T020401*     10      N/A      N/A      ** Test Results Not Found
-- 4. T020402*     10      7      85
-- 5. T030103      2      2      100
-- 6. T030309      1      1      100
-- 7. T030310      1      N/A      N/A      ** Same as T030309
-- 8. T040101     10      N/A      N/A      ** Manual Action Required to Finish
-- 9. T040102     10      N/A      N/A      ** Manual Action Required to Finish
-- 10. T060503     1      N/A      N/A      ** Times not repeatable.
--
-- * Denotes a minimal requirement.
-- ** Denotes tests that need to be examined.
--
-- If the comment for the test begins with "xx" then the test should
-- be examined. After each such test has been examined and evaluated,
-- the Results_File should be manually edited with the correct success
-- percentage for each of the tests put in. If it is determined that
-- the test should be ignored for the compiler, then "xxx" should be
-- placed in the pass percentage column of the Results_File with an
-- explanation in the comment field.

```



```
--  
-- In the case of line 3, a result for the test has not been found  
-- in the Results_File. This usually indicates that the test has  
-- not ran successfully. Such a test is usually given a pass percentage  
-- of 0, but the test itself must be examined to make sure.  
--  
-- In the case of line 7, test test for T030310 is the same as T030309.  
-- The results of test T030309 can then be simply inserted into the  
-- results field of test T030310.  
--  
-- In the case of lines 8 and 9, the test must be manually interpreted.  
-- The manual procedure outlined in the test description must be followed,  
-- and a pass percentage for the test must be determined and placed in  
-- the Results_File.  
--  
-- In the case of line 10, the times for the test were not repeatable.  
-- The test may be reran, the test rewritten, or the test may be  
-- determined to be untestable and should be ignored.  
--  
-- The rating procedure should be applied to the Results_File repeatedly  
-- until the Rating_Output file does not contain any comments that  
-- begin with "xx".
```

END Rating;

Source File: RATING.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Rating.Ada
--       Component: Package Body Rating
--       Description: ( See the package specification description )
--
WITH Twine;      -- String Manipulation Package
WITH PQAC_IO;   -- Centralized Input and Output Package

PACKAGE BODY Rating IS

    Name_Size : CONSTANT Natural := 7;

    TYPE Test_Type IS ( Normal, Minimal );
    TYPE Test_State IS
        ( Empty, Weighted, Finished, Unfinished, Unused, Unknown, Duplicated );

    SUBTYPE Weight_Range IS Integer RANGE 0 .. 1000;
    SUBTYPE Percent_Range IS Integer RANGE 0 .. 100;
    SUBTYPE Test_Index IS Integer RANGE 0 .. 200;
    SUBTYPE Test_Range IS Integer RANGE 1 .. Test_Index'LAST;

    TYPE Test_Record IS RECORD
        Test      : Twine.Series;
        Comment   : Twine.Series;
        Status    : Test_State      := Empty;
        Version   : Test_Type       := Normal;
        Weight    : Weight_Range    := 0;
        Percent   : Percent_Range   := 0;
        Passed    : Percent_Range   := 0;
        Score     : Weight_Range    := 0;
    END RECORD;

    Blanks      : CONSTANT Twine.Output_Buffer := ( OTHERS => ' ' );
    Table       : ARRAY( Test_Range ) OF Test_Record;
    Tests       : Test_Index := 0;
    Compiler    : Twine.Series;

    FUNCTION Cut_Off( Line : String; Size : Natural ) RETURN String IS
    BEGIN
        IF Line'LENGTH <= Size THEN
            RETURN Line;
        ELSE
            RETURN Line( Line'FIRST .. Line'FIRST + Size - 1 );
        END IF;
    END Cut_Off;

    PROCEDURE Print( Line : String ) IS
    BEGIN
        PQAC_IO.Put_Line( Cut_Off( Line, 80 ) );
    END Print;

    PROCEDURE Print( File : PQAC_IO.File_Type; Line : String ) IS
    BEGIN
        PQAC_IO.Put_Line( File, Cut_Off( Line, 80 ) );
    END Print;
```

Source File: RATING.ADA

```
PROCEDURE Center
( File : PQAC_IO.File_Type;
  Line : String;
  Tail : Natural := 80 ) IS

  PROCEDURE Work( Text : String ) IS
  BEGIN
    Print( File, Blanks( 1 .. ( Tail - Text'LENGTH ) / 2 ) & Text );
  END Work;

BEGIN
  Work( Cut_Off( Line, Tail ) );
END Center;

FUNCTION "&"( Text : String; Value : Integer ) RETURN String IS
BEGIN
  RETURN Text & Twine.Image( Value, 4 );
END "&";

PROCEDURE Record_Error( Message : String ) IS
BEGIN
  Print( "" );
  Print( "" );
  Print( "An Error has occurred while processing results." );
  Print( Message );
  Print( "" );
  RAISE Rating_Error;
END Record_Error;

FUNCTION Test_Name( Line : String ) RETURN String IS
BEGIN
  RETURN Line( Line'FIRST .. Line'FIRST + Name_Size - 1 );
END Test_Name;

FUNCTION Contains_Test_Name( Line : String ) RETURN Boolean IS
  FUNCTION All_Digits( Text : String ) RETURN Boolean IS
  BEGIN
    FOR Index IN Text'RANGE LOOP
      IF NOT ( Text( Index ) IN '0' .. '9' ) THEN
        RETURN False;
      END IF;
    END LOOP;
    RETURN True;
  END All_Digits;

BEGIN
  RETURN Line'LENGTH >= 7 AND THEN Line( Line'FIRST ) = 'T' AND THEN
    All_Digits( Line( Line'FIRST + 1 .. Line'FIRST + Name_Size - 1 ) );
END Contains_Test_Name;

FUNCTION "<"( A, B : Twine.Series ) RETURN Boolean IS
BEGIN
  RETURN Twine.Image( A ) < Twine.Image( B );
END "<";

FUNCTION "-"( A, B : Twine.Series ) RETURN Boolean IS
BEGIN
  RETURN Twine.Image( A ) = Twine.Image( B );
END "-";
```

```

PROCEDURE Store_Weight( Line : String ) IS
  Head : Natural := Line'FIRST + Name_Size;
  Last : Natural := 0;
  Temp : Test_Record;
BEGIN
  Tests := Tests + 1;
  Table( Tests ).Test := Twine.Create( Test_Name( Line ) );
  CASE Line( Head ) IS
    WHEN 'M' => Table( Tests ).Version := Minimal;
    WHEN ' ' => Table( Tests ).Version := Normal;
    WHEN OTHERS => Record_Error( "Unexpected letter in column 8: " & Line );
  END CASE;
  PQAC_IO.Get( Line( Head+1 .. Line'LAST ), Table( Tests ).Weight, Last );
  PQAC_IO.Get( Line( Last+1 .. Line'LAST ), Table( Tests ).Percent, Last );
  Table( Tests ).Status := Weighted;
  Temp := Table( Tests );
  FOR Index IN REVERSE 1 .. Tests - 1 LOOP
    EXIT WHEN Table( Index ).Test < Table( Index + 1 ).Test;
    IF Table( Index ).Test = Table( Index + 1 ).Test THEN
      Record_Error( "Duplicate Test Number " & Test_Name( Line ) );
    END IF;
    Temp := Table( Index );
    Table( Index ) := Table( Index + 1 );
    Table( Index + 1 ) := Temp;
  END LOOP;
EXCEPTION
  WHEN Rating_Error => RAISE;
  WHEN OTHERS => Record_Error( "Two numeric values expected: " & Line );
END Store_Weight;

```

```

PROCEDURE Load_Table( From_File : String ) IS
  Input : PQAC_IO.File_Type;
  Size : Natural := 0;
  Buffer : Twine.Output_Buffer;
BEGIN
  PQAC_IO.Open_Input( Input, From_File );
  WHILE NOT PQAC_IO.End_Of_File( Input ) LOOP
    PQAC_IO.Get_Line( Input, Buffer, Size );
    IF Contains_Test_Name( Buffer( 1 .. Size ) ) THEN
      Store_Weight( Buffer( 1 .. Size ) );
    END IF;
  END LOOP;
  PQAC_IO.Close( Input );
EXCEPTION
  WHEN OTHERS => Record_Error( "Error reading WEIGHTS file: " & From_File );
END Load_Table;

```

```

FUNCTION Find_Test_Index( Name : String ) RETURN Test_Index IS
  A : Test_Index := 1;
  B : Test_Index := Tests;
  M : Test_Index := 0;
BEGIN
  LOOP
    EXIT WHEN A > B;
    M := ( A + B ) / 2;
    IF Twine.Image( Table( M ).Test ) = Name THEN
      RETURN M;
    ELSIF Twine.Image( Table( M ).Test ) > Name THEN
      B := M - 1;
    ELSE
      A := M + 1;
    END IF;
  END LOOP;
  Record_Error( "Test " & Name & " not given a weight." );
END Find_Test_Index;

```

Source File: RATING.ADA

```
FUNCTION Code_Of( Code : String ) RETURN Test_State IS
  Value : Percent_Range;
BEGIN
  IF Code = "xxx" THEN
    RETURN Unused;
  ELSIF Code = "???" THEN
    RETURN Unknown;
  ELSIF Code = "... " THEN
    RETURN Unfinished;
  ELSIF Code = "===" THEN
    RETURN Duplicated;
  ELSE
    Value := Integer'VALUE( Code );
    RETURN Finished;
  END IF;
EXCEPTION
  WHEN OTHERS => RETURN Empty;
END Code_Of;
```

```
PROCEDURE Store_Score( Line : String ) IS
  Code : String( 1 .. 3 ) := Line( Name_Size + 2 .. Name_Size + 4 );
  Place : Test_Index := 1;
```

```
FUNCTION Evaluate( T : Test_Record ) RETURN Natural IS
```

```
  FUNCTION X( Weight, Cutoff, Pass : Float ) RETURN Float IS
  BEGIN
    IF Pass >= 100.0 THEN
      RETURN Weight;
    ELSIF Pass <= Cutoff THEN
      RETURN 0.0;
    ELSE
      RETURN Weight * ( Pass - Cutoff ) / ( 100.0 - Cutoff );
    END IF;
  END X;
```

```
  BEGIN
    RETURN Natural( X( Float(T.Weight), Float(T.Percent), Float(T.Passed)) );
  END Evaluate;
```

```
FUNCTION Comment_Of( Line : String ) RETURN String IS
```

```
  BEGIN
    IF Line'LAST >= Name_Size + 6 THEN
      RETURN Line( Name_Size + 6 .. Line'LAST );
    ELSE
      RETURN "";
    END IF;
  END Comment_Of;
```

```
BEGIN
```

```
  Place := Find_Test_Index( Test_Name( Line ) );
  IF Table( Place ).Status /= Weighted THEN
    Print( "Test " & Test_Name( Line ) & " results superseded." );
  END IF;
  Table( Place ).Status := Code_Of( Code );
  IF Table( Place ).Status = Finished THEN
    Table( Place ).Passed := Integer'VALUE( Code );
  ELSE
    Table( Place ).Passed := 0;
  END IF;
  Table( Place ).Score := Evaluate( Table( Place ) );
  Table( Place ).Comment := Twine.Create( Comment_Of( Line ) );
  IF Table( Place ).Status = Empty THEN
    Record_Error( "Percentage Value Error: " & Test_Name( Line ) & "." );
  END IF;
END Store_Score;
```

Source File: RATING.ADA

```
PROCEDURE Read_Scores( Input_File : String ) IS
  Input   : PQAC_IO.File_Type;
  Buffer   : Twine.Output_Buffer;
  Size    : Natural := 0;
BEGIN
  PQAC_IO.Open_Input( Input, Input_File );
  PQAC_IO.Get_Line( Input, Buffer, Size );
  Compiler := Twine.Create( Buffer( 1 .. Size ) );
  WHILE NOT PQAC_IO.End_Of_File( Input ) LOOP
    PQAC_IO.Get_Line( Input, Buffer, Size );
    IF Contains_Test_Name( Buffer( 1 .. Size ) ) THEN
      Store_Score( Buffer( 1 .. Size ) );
    END IF;
  END LOOP;
  PQAC_IO.Close( Input );
  FOR Index IN 1 .. Tests LOOP
    IF Table( Index ).Status = Empty THEN
      Record_Error( "Found Empty Status at " & Index );
    ELSIF Table( Index ).Status = Weighted THEN
      Table( Index ).Comment := Twine.Create( "" );
    END IF;
  END LOOP;
EXCEPTION
  WHEN OTHERS => Record_Error( "Error reading result file: " & Input_File );
END Read_Scores;
```

```
PROCEDURE Process_Results( Output_File : String ) IS
  Banner : CONSTANT String( 1 .. 31 ) := "PQAC Test Suite Statistics for ";
  Output  : PQAC_IO.File_Type;
  Buffer   : Twine.Output_Buffer;
```

```
FUNCTION Test_Of( T : Test_Record ) RETURN String IS
BEGIN
  IF T.Version = Minimal THEN
    RETURN Twine.Image( T.Test ) & "x";
  ELSE
    RETURN Twine.Image( T.Test ) & " ";
  END IF;
END Test_Of;
```

```
FUNCTION Score( T : Test_Record ) RETURN String IS
BEGIN
  IF T.Status = Finished THEN
    RETURN "" & T.Score;
  ELSE
    RETURN " N/A";
  END IF;
END Score;
```

```
FUNCTION Percent( T : Test_Record ) RETURN String IS
BEGIN
  IF T.Status = Finished THEN
    RETURN "" & T.Passed;
  ELSE
    RETURN " N/A";
  END IF;
END Percent;
```

```
FUNCTION Comment( T : Test_Record ) RETURN String IS
```

```
  FUNCTION Explanation( Code : Test_State ) RETURN String IS
  BEGIN
    CASE Code IS
      WHEN Empty      => RETURN "xxx INTERNAL ERROR xxx";
      WHEN Weighted   => RETURN "xx Test Results Not Found";
      WHEN Finished   => RETURN "";
      WHEN Unfinished => RETURN "xx Manual Action Required to Finish";
      WHEN Unused     => RETURN "";
    END CASE;
  END Explanation;
END Comment;
```

```

        WHEN Unknown      => RETURN "*** ";
        WHEN Duplicated   => RETURN "*** Same as ";
    END CASE;
END Explanation;

BEGIN
    RETURN Explanation( T.Status ) & Twine.Image( T.Comment );
END Comment;

PROCEDURE Print( Item : Natural; T : Test_Record ) IS
BEGIN
    Print( Output,
           "" & Item & ".  " & Test_Of( T ) & "  " & T.Weight & "  " &
           Score( T ) & "  " & Percent( T ) & "  " & Comment( T ) );
END Print;

GENERIC
    WITH PROCEDURE Parse
        ( Test      : Test_Record;
          Valid      : OUT Boolean;
          Applied    : OUT Boolean;
          Weight     : OUT Weight_Range;
          Score      : OUT Weight_Range );
    PROCEDURE Stat_Control( Title : String );

PROCEDURE Stat_Control( Title : String ) IS
    Total      : Natural := 0;
    Partial    : Natural := 0;
    Passed     : Natural := 0;
    Valid      : Boolean;
    Applied    : Boolean;
    Weight     : Weight_Range;
    Score      : Weight_Range;

    FUNCTION Ratio( A, B : Natural ) RETURN String IS
        Rate : Natural;
    BEGIN
        IF B = 0 THEN
            RETURN "N/A";
        ELSE
            Rate := Natural( Float( 100 * A ) / Float( B ) );
            IF Rate >= 100 AND THEN A < B THEN
                Rate := 99;
            END IF;
            RETURN "" & Rate;
        END IF;
    END Ratio;

    PROCEDURE Print( A : String; B : String; C : String := "" ) IS
        Size : Natural := 40 - A'LENGTH;
    BEGIN
        Print( Output, A & Blanks( 1 .. Size ) & B & C );
    END Print;

BEGIN
    FOR Index IN 1 .. Tests LOOP
        Parse( Table( Index ), Valid, Applied, Weight, Score );
        IF Valid THEN
            Total := Total + Weight;
            IF Applied THEN
                Partial := Partial + Weight;
                Passed := Passed + Score;
            END IF;
        END IF;
    END LOOP;
    Print( Output, "" );
    Center( Output, "Statistics Using " & Title, 50 );
    Print( Output, "" );

```

Source File: RATING.ADA

```
Print( "Total " & Title & ":", "" & Total );
Print( "Applicable " & Title & ":", "" & Partial );
Print( "Passed:", "" & Passed );
Print( "Failed:", "" & ( Partial - Passed ) );
Print( "Pass Percentage:", Ratio( Passed, Partial ), "%" );
Print( Output, "" );
END Stat_Control;

PROCEDURE Parse_All_Tests
( Test : Test_Record;
  Valid : OUT Boolean;
  Applied : OUT Boolean;
  Weight : OUT Weight_Range;
  Score : OUT Weight_Range ) IS
BEGIN
  Valid := True;
  Applied := Test.Status = Finished;
  Weight := 1;
  IF Test.Passed = 100 THEN
    Score := 1;
  ELSE
    Score := 0;
  END IF;
END Parse_All_Tests;

PROCEDURE Parse_All_Weights
( Test : Test_Record;
  Valid : OUT Boolean;
  Applied : OUT Boolean;
  Weight : OUT Weight_Range;
  Score : OUT Weight_Range ) IS
BEGIN
  Valid := True;
  Applied := Test.Status = Finished;
  Weight := Test.Weight;
  Score := Test.Score;
END Parse_All_Weights;

PROCEDURE Parse_Min_Tests
( Test : Test_Record;
  Valid : OUT Boolean;
  Applied : OUT Boolean;
  Weight : OUT Weight_Range;
  Score : OUT Weight_Range ) IS
BEGIN
  Valid := Test.Version = Minimal;
  Applied := Test.Status = Finished;
  Weight := 1;
  IF Test.Passed = 100 THEN
    Score := 1;
  ELSE
    Score := 0;
  END IF;
END Parse_Min_Tests;

PROCEDURE Parse_Min_Weights
( Test : Test_Record;
  Valid : OUT Boolean;
  Applied : OUT Boolean;
  Weight : OUT Weight_Range;
  Score : OUT Weight_Range ) IS
BEGIN
  Valid := Test.Version = Minimal;
  Applied := Test.Status = Finished;
  Weight := Test.Weight;
  Score := Test.Score;
END Parse_Min_Weights;

PROCEDURE Print_All_Tests IS NEW Stat_Control( Parse_All_Tests );
```


Source File: RATING.ADA

```
PROCEDURE Print_All_Weights IS NEW Stat_Control( Parse_All_Weights );
PROCEDURE Print_Min_Tests   IS NEW Stat_Control( Parse_Min_Tests );
PROCEDURE Print_Min_Weights IS NEW Stat_Control( Parse_Min_Weights );
```

```
BEGIN
  PQAC_IO.Open_Output( Output, Output_File );
  Print( Output, "" );
  Print( Output, "" );
  Center( Output, Banner & Twine.Image( Compiler ) );
  Print( Output, "" );
  Print( Output, "" );
  Print( Output, "" );
  Print( Output, " Num      Test   Weight Score    %   Comments:" );
  Print( Output, "" );
  FOR Index IN 1 .. Tests LOOP
    Print( Index, Table( Index ) );
  END LOOP;
  Print( Output, "" );
  Print( Output, "      * Denotes a minimal requirement." );
  Print( Output, "      ** Denotes tests that need to be examined." );
  Print( Output, "" );
  Print_All_Tests( "Tests" );
  Print_All_Weights( "Weights" );
  Print_Min_Tests( "Minimal Tests" );
  Print_Min_Weights( "Minimal Weights" );
  PQAC_IO.Close( Output );
END Process_Results;
```

```
PROCEDURE Rating_Tool
( Weight_Table : String;
  Results_File : String;
  Rating_Output : String ) IS
BEGIN
  Load_Table( Weight_Table );
  Read_Scores( Results_File );
  Process_Results( Rating_Output );
END Rating_Tool;
```

END Rating;

Source File: RESULT_.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Result_.Ada
--       Component: Package Specification Result
--       Description: This package is used by the tests for recording their results.
--                   Several utility functions used by some tests are included here.
--
```

PACKAGE Result IS

```
SUBTYPE Percentage IS Integer RANGE 0 .. 100;      -- Percent
SUBTYPE File_Length IS Integer RANGE 0 .. 10_000_000; -- Machine Words
```

```
Result_Error : EXCEPTION;
```

```
PROCEDURE Print( Message : String );
-- Sends the Message to the test output stream.
```

```
PROCEDURE Passed( Test : String; Percent : Percentage; Comment : String := "" );
-- Records the pass percentage for the given test. A comment may
-- be included which will be printed in the results report.
```

```
PROCEDURE Passed( Test : String; Success : Boolean; Comment : String := "" );
-- Same as previous function, with Success values of False and True
-- interpreted as 0% and 100% respectively.
```

```
PROCEDURE Manual_Test ( Test : String; Comment : String := "" );
-- Record the fact that the test needs manual interpretation.
```

```
PROCEDURE Not_Applicable( Test : String; Comment : String := "" );
-- Record the fact that the test is not applicable.
```

```
PROCEDURE Inconclusive ( Test : String; Comment : String := "" );
-- Record the fact the the test encountered an error or needs adjustment.
```

```
PROCEDURE Equivalent ( Test : String; Old_Test : String );
-- Record the fact that a test is the same as another.
```

```
PROCEDURE Print_Code_Size( File : String; Size : OUT File_Length );
-- Prints the size in machine words of the specified file to the output
-- stream. The size is also returned.
```

```
FUNCTION Image
( Value : Integer;
  Field : Positive := 8 ) RETURN String;
-- Returns the image of the specified integer in a string of the specified
-- field length.
```

```
FUNCTION Image
( Value : Float;
  Field : Positive := 8;
  Aft  : Positive := 2;
  Exp  : Natural  := 0 ) RETURN String;
-- Returns the image of the specified float in a string of the specified
-- field length.
```

```
FUNCTION Min( Value_1, Value_2 : Integer ) RETURN Integer;
```

```
FUNCTION Max( Value_1, Value_2 : Integer ) RETURN Integer;
```

```
FUNCTION Temp_Name RETURN String;
-- Returns the name of a temporary file name that may be used by the tests
-- for file input and output tests.
```

END Result;

Source File: RESULT.ADA

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:  BAP
--       Date:    10/01/88
--       File:    Result.Ada
--       Component: Package Body Result
--       Description: ( See Package Specification Description )
--
--
WITH Names;      -- Enumeration Types
WITH Twine;      -- String Manipulation Package
WITH Count;      -- Package for counting Ada source lines and File sizes
WITH Common;     -- Interface to compiler tables and test suite state
WITH PQAC_IO;    -- Centralized Input and Output package

PACKAGE BODY Result IS

  TYPE Result_Type IS
    ( Finished,      -- test result is complete
      Unfinished,    -- test needs manual interpretation
      Unused,        -- test not applicable
      Unknown,       -- test error or test needs adjustment
      Duplicated );  -- same test as another

  FUNCTION ID( Prefix : String; Suffix : Names.File_Category ) RETURN String
    RENAMES Common.Build_Name;

  FUNCTION "&"( Text : String; Item : Integer ) RETURN String IS
  BEGIN
    RETURN Text & Image( Item, 3 );
  END "&";

  PROCEDURE Print( Message : String ) IS
  BEGIN
    PQAC_IO.Put_Line( Message );
  END Print;

  PROCEDURE Save_Test
    ( Test      : String;
      Value     : Percentage;
      Kind      : Result_Type;
      Text      : String := "";
      Comment   : String := "" ) IS

    FUNCTION Command RETURN String IS
    BEGIN
      CASE Kind IS
        WHEN Finished  => RETURN Image( Value, 3 );
        WHEN Unfinished => RETURN "...";
        WHEN Unused    => RETURN "xxx";
        WHEN Unknown   => RETURN "???";
        WHEN Duplicated => RETURN "===";
      END CASE;
    END Command;

    FUNCTION Result_Line RETURN String IS
    BEGIN
      RETURN Test & " " & Command & " " & Comment;
    END Result_Line;

  BEGIN
    Print( "" );
```

Source File: RESULT.ADA

```
    Print( Test & " " & Text & " " & Comment );
    PQAC_IO.Append( Common.Image( Names.Test_Result ), Result_Line );
END Save_Test;
```

```
PROCEDURE Passed( Test : String; Percent : Percentage; Comment :String:="" ) IS
BEGIN
    Save_Test( Test, Percent, Finished, "Passed " & Percent & "%", Comment );
END Passed;
```

```
PROCEDURE Passed( Test : String; Success : Boolean; Comment : String := "" ) IS
BEGIN
    CASE Success IS
        WHEN True => Passed( Test, 100 );
        WHEN False => Passed( Test, 0 );
    END CASE;
END Passed;
```

```
PROCEDURE Inconclusive( Test : String; Comment : String := "" ) IS
BEGIN
    Save_Test( Test, 0, Unknown, "Inconclusive Results.", Comment );
END Inconclusive;
```

```
PROCEDURE Not_Applicable( Test : String; Comment : String := "" ) IS
BEGIN
    Save_Test( Test, 0, Unused, "Not Applicable.", Comment );
END Not_Applicable;
```

```
PROCEDURE Manual_Test( Test : String; Comment : String := "" ) IS
BEGIN
    Save_Test( Test, 0, Unfinished, "Requires Manual Action.", Comment );
END Manual_Test;
```

```
PROCEDURE Equivalent( Test : String; Old_Test : String ) IS
BEGIN
    Save_Test( Test, 0, Duplicated, "Results Same As", Old_Test );
END Equivalent;
```

```
PROCEDURE Print_Code_Size( File : String; Size : OUT File_Length ) IS
    Total : Natural;
```

```
    PROCEDURE Process( Input : String; Output : String ) IS
    BEGIN
        Count.Code_Size( Input, Output );
        PQAC_IO.Get_Value( Output, Total );
        PQAC_IO.Delete_File( Output );
        Size := Total;
        Print( "Size Of " & Input & ": " &
              Image( Total, 10 ) & " Words." );
    END Process;
```

```
    BEGIN
        Process( ID( File, Names.Execute ), ID( File, Names.Data ) );
    END Print_Code_Size;
```

```
FUNCTION Image
( Value : Integer;
  Field : Positive := 8 ) RETURN String IS
BEGIN
```

Source File: RESULT.ADA

```
        RETURN Twine.Image( Value, Field );  
    END Image;
```

```
FUNCTION Image  
    ( Value : Float;  
      Field : Positive := 8;  
      Aft   : Positive := 2;  
      Exp   : Natural  := 0 ) RETURN String IS  
BEGIN  
    RETURN Twine.Image( Value, Field, Aft, Exp );  
END Image;
```

```
FUNCTION Min( Value_1, Value_2 : Integer ) RETURN Integer IS  
BEGIN  
    IF Value_1 < Value_2 THEN  
        RETURN Value_1;  
    ELSE  
        RETURN Value_2;  
    END IF;  
END Min;
```

```
FUNCTION Max( Value_1, Value_2 : Integer ) RETURN Integer IS  
BEGIN  
    IF Value_1 > Value_2 THEN  
        RETURN Value_1;  
    ELSE  
        RETURN Value_2;  
    END IF;  
END Max;
```

```
FUNCTION Temp_Name RETURN String IS  
BEGIN  
    RETURN ID( "TEMP", Names.Data );  
END Temp_Name;
```

```
END Result;
```

Source File: SCRIPT_.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--       Author:  BAP
--       Date:   10/01/88
--       File:   Script_.Ada
--       Component: Package Specification Script
-- Description:  This package controls output to the Script file for each test.
--
```

```
WITH Names;  -- Enumeration Types
WITH Twine;  -- String Manipulation Package
```

PACKAGE Script IS

```
TYPE Option_List IS ARRAY( Positive RANGE <> ) OF Names.Compiler_Options;
No_Options : CONSTANT Option_List( 1..0 ) := ( OTHERS => Names.Syntax_Only );
Script_Error : EXCEPTION;
```

PROCEDURE Print(Text : String);

```
-- Sends a command to print the Text to the script file.
```

PROCEDURE Keep

```
( Command : Names.OS_Primitives;
  Line    : String := "" );
```

```
-- Sends the command with the Line arguments to the script file.
```

PROCEDURE Keep_Execute

```
( File_Name   : String;
  File_Type   : Names.File_Category;
  Time_Name_1 : String := "";
  Time_Name_2 : String := "";
  Code_Size   : String := "" );
```

```
-- Sends commands to link and execute the given File_Name. File_Type
-- may be Ada or FORTRAN. If Time_Name_1 and Time_Name_2 are not ""
-- then these file names will be used to hold the current time before
-- and after the File_Name is executed. If Code_Size is not "" then
-- the size of the executable file will be saved in that file name.
-- Commands to delete the executable file and object file after the
-- test is finished will also be sent to the script file.
```

PROCEDURE Keep_Compile

```
( File_Name       : String;
  File_Type       : Names.File_Category;
  Compile_Options : Option_List := No_Options;
  Delete_After    : Boolean     := True );
```

```
-- Sends a command to compile the given File_Name. The File_Type may
-- be Ada for FORTRAN. The complete file name including appropriate
-- suffix is created by this procedure. The compile command uses
-- the specified Compile_Options. If Time_Compile is one of the
-- options, then the number of lines of Ada source compiled per
-- minute, or speed of FORTRAN compilation, is computed. If Delete_After
-- is true, then the Ada or FORTRAN source file will be deleted after
-- the test is completed.
```

PROCEDURE Keep_Listings

```
( File_Name       : String;
  File_Type       : Names.File_Category;
  Compile_Options : Option_List := No_Options );
```

Source File: SCRIPT_.ADA

```
-- If Compiler_Listing or Assembly_Listing are one of the specified
-- Compile_Options then commands will be sent to the script file
-- to print out the specified listing.
```

PROCEDURE Keep_Compare

```
( File_Name      : String;
  File_Type      : Names.File_Category;
  Compile_Options : Option_List;
  Save_Names     : Twine.Series_List );

-- For each compiler option in Compile_Options, commands are sent
-- out to compile the given File_Name and File_Type with the
-- specified compiler option. The code will then be linked and
-- executed with the execution speeds and execution code sizes also
-- recorded. The test statistics are stored in 6 files. The
-- base name for these files is the corresponding name in Save_Names.
--
-- File Name - Contents
-- baseA.DAT - Compile Start Time
-- baseB.DAT - Compile Stop Time
-- baseC.DAT - Ada Source lines
-- baseD.DAT - Execute Start Time
-- baseE.DAT - Execute Stop Time
-- baseF.DAT - Executable File Size
--
-- Example: Compile_Options == ( Syntax_Only, Space_Optimized )
--          Save_Names      == ( "TEMP1", "TEMP2" )
--
-- Files Produced: ( A, B, C, D, E, F as defined above )
-- Syntax_Only Statistics:
--   TEMP1A.DAT, TEMP1B.DAT, TEMP1C.DAT,
--   TEMP1D.DAT, TEMP1E.DAT, TEMP1F.DAT
-- Space_Optimized Statistics:
--   TEMP2A.DAT, TEMP2B.DAT, TEMP2C.DAT,
--   TEMP2D.DAT, TEMP2E.DAT, TEMP2F.DAT
```

PROCEDURE Keep_Code_List

```
( File_Name      : String;
  File_Type      : Names.File_Category;
  Is_Duplicated   : Boolean := False );

-- Sends commands to the script file causing the specified File_Name
-- to be listed in the test output stream. If Is_Duplicated is True,
-- then the labels on the listed code are produced, but the file is
-- not listed. A comment that the code has been previously listed
-- is printed instead.
```

PROCEDURE Keep_Expand

```
( File_Name      : String;
  Old_Suffix     : Names.File_Category;
  New_Suffix     : Names.File_Category );

-- Sends a command to invoke the code expander tool. The input
-- is taken from the File_Name with the Old_Suffix and written to
-- a File_Name with the New_Suffix.
```

PROCEDURE Output_Script(File_Name : String);

```
-- All of the commands accumulated so far from the previous procedures
-- are written to the specified File_Name. This procedure should
-- only be called once per test. After this procedure has been called,
-- none of the other procedures in this package should be called.
```

END Script;

Source File: SCRIPT.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:  BAP
--       Date:    10/01/88
--       File:    Script.Ada
--       Component: Package Body Script
--       Description: ( See Package Specification Description )
--
WITH Common;  -- Compiler dependent tables and test suite state
WITH PQAC_IO; -- Centralized input and output package

PACKAGE BODY Script IS

    Limit : CONSTANT Natural := 1000;

    TYPE Text_Type( Maximum : Natural := 0 ) IS RECORD
        Size : Natural := 0;
        Text : Twine.Series_List( 1 .. Maximum );
    END RECORD;

    Output  : Text_Type( Limit );
    Deletes : Text_Type( Limit );

    FUNCTION ID( Name : String; Kind : Names.File_Category ) RETURN String
        RENAMES Common.Build_Name;

    FUNCTION "="( A, B : Names.Compiler_Options ) RETURN Boolean
        RENAMES Names."=";

    FUNCTION "="( A, B : Names.File_Category ) RETURN Boolean
        RENAMES Names."=";

    FUNCTION "="( A, B : Names.OS_Primitives ) RETURN Boolean
        RENAMES Names."=";

    FUNCTION "&"( A : Common.System_Attributes; Text : String ) RETURN String IS
    BEGIN
        RETURN Common.Image( A ) & Text;
    END "&";

    FUNCTION "&"( Option : Names.OS_Primitives; Text : String ) RETURN String IS
    BEGIN
        RETURN Common.Image( Option ) & Text;
    END "&";

    FUNCTION "&"( Option : Names.Transfer_Files; Text : String ) RETURN String IS
    BEGIN
        RETURN Common.Image( Option ) & Text;
    END "&";

    FUNCTION "&"( Text : String; Option : Names.Transfer_Files ) RETURN String IS
    BEGIN
        RETURN Text & Common.Image( Option );
    END "&";

    FUNCTION "&"( A : Twine.Series; B : String ) RETURN String IS
    BEGIN
```


Source File: SCRIPT.ADA

```
    RETURN Twine.Image( A ) & B;
END "&";
```

```
FUNCTION Image( Option : Names.Compiler_Options ) RETURN String IS
BEGIN
    IF Option = Names.Time_Compile THEN
        RETURN "";
    ELSE
        RETURN Common.Image( Option );
    END IF;
END Image;
```

```
FUNCTION Image( List : Option_List ) RETURN String IS
BEGIN
    IF List'LENGTH = 0 THEN
        RETURN "";
    ELSIF List'LENGTH = 1 THEN
        RETURN Image( List( List'FIRST ) ) & " ";
    ELSE
        RETURN Image( List( List'FIRST ) ) &
            Image( List( List'FIRST + 1 .. List'LAST ) );
    END IF;
END Image;
```

```
PROCEDURE Process_Error( Line : String ) IS
BEGIN
    PQAC_IO.Record_Error( Line );
    RAISE Script_Error;
END Process_Error;
```

```
FUNCTION Message_Of( Message : IN String ) RETURN String IS
    Size : CONSTANT Natural := Twine.Output_Buffer'LENGTH;
    Dash : CONSTANT String( 1 .. Size / 2 ) := ( OTHERS => '-' );
    Half_1 : Natural := ( Size - Message'LENGTH ) / 2;
    Half_2 : Natural := Size - Message'LENGTH - Half_1;
BEGIN
    RETURN Dash( 1 .. Half_1 ) & Message & Dash( 1 .. Half_2 );
END Message_Of;
```

```
FUNCTION Member( Option : Names.Compiler_Options; List : Option_List )
    RETURN Boolean IS
BEGIN
    FOR Index IN List'RANGE LOOP
        IF List( Index ) = Option THEN
            RETURN True;
        END IF;
    END LOOP;
    RETURN False;
END Member;
```

```
PROCEDURE Keep
( Command : Names.OS_Primitives;
  Line : String := "" ) IS
BEGIN
    IF Output.Size = Output.Maximum THEN
        Process_Error( "Storage space exceeded." );
    END IF;
    Output.Size := Output.Size + 1;
    Output.Text( Output.Size ) := Twine.Create( Command & " " & Line );
END Keep;
```

```

PROCEDURE Save_Delete( Name : String ) IS
  PROCEDURE Try_Delete( Text : String ) IS
    BEGIN
      FOR Index IN 1 .. Deletes.Size LOOP
        IF Twine.Equal( Text, Deletes.Text( Index ) ) THEN
          RETURN;
        END IF;
      END LOOP;
      IF Deletes.Size = Deletes.Maximum THEN
        Process_Error( "Storage space exceeded." );
      END IF;
      Deletes.Size := Deletes.Size + 1;
      Deletes.Text( Deletes.Size ) := Twine.Create( Text );
    END Try_Delete;

  BEGIN
    Try_Delete( Names.Delete & " " & Name );
  END Save_Delete;

PROCEDURE Print( Text : String ) IS
  BEGIN
    Keep( Names.Print, Text );
  END Print;

PROCEDURE Keep_Save_Time
  ( File_Name : String ) IS
  BEGIN
    IF File_Name /= "" THEN
      Keep( Names.Store_Time, File_Name );
      Save_Delete( File_Name );
    END IF;
  END Keep_Save_Time;

PROCEDURE Keep_Count_Lines
  ( File_Name : String;
    Save_Name : String ) IS
  BEGIN
    IF Save_Name /= "" THEN
      Keep( Names.Count, File_Name & " " & Save_Name );
      Save_Delete( Save_Name );
    END IF;
  END Keep_Count_Lines;

PROCEDURE Keep_Code_Size
  ( File_Name : String;
    Save_Name : String ) IS
  BEGIN
    IF Save_Name /= "" THEN
      Keep( Names.Code_Size, File_Name & " " & Save_Name );
      Save_Delete( Save_Name );
    END IF;
  END Keep_Code_Size;

PROCEDURE Keep_Execute
  ( File_Name : String;
    File_Type : Names.File_Category;
    Time_Name_1 : String := "";
    Time_Name_2 : String := "";
    Code_Size : String := "" ) IS
  BEGIN
    Print( "LINKING " & File_Name & " ..." );
    IF File_Type = Names.Ada THEN

```

```

        Keep( Names.Link, File_Name );
    ELSE
        Keep( Names.Link_Fortran, File_Name );
    END IF;
    Print( "EXECUTING " & File_Name & " ..." );
    Keep_Save_Time( Time_Name_1 );
    Keep( Names.Execute, File_Name );
    Keep_Save_Time( Time_Name_2 );
    Keep_Code_Size( ID( File_Name, Names.Execute ), Code_Size );
    Save_Delete( ID( File_Name, Names.Object ) );
    Save_Delete( ID( File_Name, Names.Execute ) );
END Keep_Execute;

```

```

PROCEDURE Set_Compile
( Name      : String;
  Option    : String;
  File_Type : Names.File_Category;
  Delete_After : Boolean := False;
  Time_Name_1 : String := "";
  Time_Name_2 : String := "";
  Count_Name  : String := "" ) IS
BEGIN
    Keep( Names.Print, "COMPILING " & Option & Name & " ..." );
    Keep_Save_Time( Time_Name_1 );
    CASE File_Type IS
        WHEN Names.Ada =>
            Keep( Names.Compile,
                  Common.Base_Compiler_Option & Option & " " & Name );
        WHEN Names.FORTRAN =>
            Keep( Names.Fortran, Name );
        WHEN OTHERS =>
            Process_Error( "Ada or FORTRAN expected." );
    END CASE;
    Keep_Save_Time( Time_Name_2 );
    Keep_Count_Lines( Name, Count_Name );
    IF Time_Name_1 /= "" THEN
        Keep( Names.Compute_Rate,
              Time_Name_1 & " " &
              Time_Name_2 & " " &
              Count_Name );
    END IF;
    IF Delete_After THEN
        Save_Delete( Name );
    END IF;
END Set_Compile;

```

```

PROCEDURE Keep_Compile
( File_Name : String;
  File_Type : Names.File_Category;
  Compile_Options : Option_List := No_Options;
  Delete_After : Boolean := True ) IS
BEGIN
    IF Member( Names.Time_Compile, Compile_Options ) THEN
        Set_Compile
        ( ID( File_Name, File_Type ),
          Image( Compile_Options ),
          File_Type,
          Delete_After,
          Names.Save_Time_1 & "",
          Names.Save_Time_2 & "",
          Names.Save_Count & "" );
    ELSE
        Set_Compile
        ( ID( File_Name, File_Type ),
          Image( Compile_Options ), File_Type, Delete_After );
    END IF;
END Keep_Compile;

```

```

PROCEDURE Keep_Listings
( File_Name      : String;
  File_Type      : Names.File_Category;
  Compile_Options : Option_List := No_Options ) IS
BEGIN
  IF Member( Names.Compiler_Listing, Compile_Options ) THEN
    Keep( Names.Print, Message_Of( " START OF COMPILER LISTING " ) );
    Keep( Names.List, ID( File_Name, Names.List ) );
    Keep( Names.Print, Message_Of( " END OF COMPILER LISTING " ) );
    Save_Delete( ID( File_Name, Names.List ) );
  END IF;
  IF Member( Names.Assembly_Listing, Compile_Options ) THEN
    Keep( Names.Print, Message_Of( " START OF ASSEMBLY LISTING " ) );
    Keep( Names.List, ID( File_Name, Names.Machine ) );
    Keep( Names.Print, Message_Of( " END OF ASSEMBLY LISTING " ) );
    Save_Delete( ID( File_Name, Names.Machine ) );
  END IF;
END Keep_Listings;

PROCEDURE Keep_Compare
( File_Name      : String;
  File_Type      : Names.File_Category;
  Compile_Option : Names.Compiler_Options;
  Save_Name      : String ) IS
  New_List : Option_List( 1 .. 1 ) := ( OTHERS => Compile_Option );
BEGIN
  Set_Compile
    ( ID( File_Name, File_Type ), Image( New_List ), File_Type, True,
      ID( Save_Name & "A", Names.Data ),
      ID( Save_Name & "B", Names.Data ),
      ID( Save_Name & "C", Names.Data ) );
  Keep_Execute
    ( File_Name, File_Type,
      ID( Save_Name & "D", Names.Data ),
      ID( Save_Name & "E", Names.Data ),
      ID( Save_Name & "F", Names.Data ) );
  IF Member( Names.Compiler_Listing, New_List ) THEN
    Save_Delete( ID( File_Name, Names.List ) );
  END IF;
  IF Member( Names.Assembly_Listing, New_List ) THEN
    Save_Delete( ID( File_Name, Names.Machine ) );
  END IF;
END Keep_Compare;

PROCEDURE Keep_Compare
( File_Name      : String;
  File_Type      : Names.File_Category;
  Compile_Options : Option_List;
  Save_Names     : Twine.Series_List ) IS
BEGIN
  FOR Index IN Compile_Options'RANGE LOOP
    Keep_Compare
      ( File_Name, File_Type,
        Compile_Options( Index ),
        Twine.Image( Save_Names( Index ) ) );
  END LOOP;
END Keep_Compare;

PROCEDURE Keep_Code_List
( File_Name      : String;
  File_Type      : Names.File_Category;
  Is_Duplicated  : Boolean := False ) IS
BEGIN
  Print( "" );
  Print( Message_Of( " TEST CODE " & ID( File_Name, File_Type ) & " " ) );
  IF Is_Duplicated THEN
    Keep( Names.List, ID( File_Name, File_Type ) );
  ELSE

```

Source File: SCRIPT.ADA

```
        Print( "( See Previous Code Segment )" );
    END IF;
    Print( Message_Of( "  END OF TEST CODE  " ) );
    Print( "" );
END Keep_Code_List;

PROCEDURE Keep_Expand
( File_Name      : String;
  Old_Suffix     : Names.File_Category;
  New_Suffix     : Names.File_Category ) IS

    PROCEDURE Do_Expand( File_1, File_2 : String ) IS
    BEGIN
        Print( "EXPANDING " & File_1 & " --> " & File_2 );
        Keep( Names.Expand, File_1 & " " & File_2 );
        Save_Delete( File_1 );
    END Do_Expand;

BEGIN
    Do_Expand( ID( File_Name, Old_Suffix ), ID( File_Name, New_Suffix ) );
END Keep_Expand;

PROCEDURE Output_Script( File_Name : String ) IS
    File : PQAC_IO.File_Type;
BEGIN
    PQAC_IO.Open_Output( File, File_Name );
    FOR Index IN 1 .. Output.Size LOOP
        PQAC_IO.Put_Line( File, Output.Text( Index ) & "" );
    END LOOP;
    FOR Index IN 1 .. Deletes.Size LOOP
        PQAC_IO.Put_Line( File, Deletes.Text( Index ) & "" );
    END LOOP;
    PQAC_IO.Close( File );
END Output_Script;

END Script;
```

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Support.Ada
--       Component: Procedure Support
--       Description: Main procedure that drives the various parts of the test suite
--                   support software. Every function of the test suite is accessed
--                   through this procedure.
--
--       When this procedure is executed, a line from the parameter file is read in
--       and this line is parsed. The line should contain a command name followed
--       by a list of arguments. The parameter file name is found in the Tables
--       package.
--
--       Allowed Commands:
--
--       Set_Up
--       Parse      Test_Name
--       Expand     In_File Out_File
--       Count      In_File Out_File
--       Code_Size  In_File Out_File
--       Store_Time Out_File
--       Compute_Rate Start_Time Stop_Time Optional_Size
--       Rating     Weight_File Result_File
--
--       Action Descriptions:
--
--       Set_Up
--       This should be called once before executing any of the tests in
--       the test suite or before calling any of the command listed below.
--       It initializes the test suite state by creating the test suite
--       state file, and prints out the first line of the results file
--       with the current compilers name. When this command is executed,
--       a list of possible compilers will be displayed, and the user will
--       be prompted for the name of the current compiler.
--
--       Parse Test_Name - Example "PARSE T010100"
--       Test_Name must be of the form T?????? where ? are all digits, e.g.
--       Parse T010100. In this case, file T010100.TST will be parsed and a
--       script file T010100.SCR will be created, along with any files created
--       during the parsing of the test.
--
--       Expand In_File Out_File - Example "EXPAND TEST_FILE.GEN TEST_FILE.ADA"
--       The file Test_File.Gen will be expanded with the results placed
--       in Test_File.Ada.
--
--       Count In_File Out_File - Example "COUNT TEST_FILE.ADA SCOUNT.DAT"
--       The number of Ada source lines in Test_File.Ada will be saved
--       in Scount.Dat.
--
--       Code_Size In_File Out_File - Example "CODE_SIZE TEST_FILE.EXE SSIZE.DAT"
--       The number of machine words in Test_File.Exe will be saved
--       in Ssize.Dat.
--
--       Store_Time Out_File - Example "STORE_TIME STIME1.DAT"
--       The current time will be saved in Stime1.Dat.
--
--       Compute_Rate Start_Time Stop_Time Optional_Size
--       - Example "COMPUTE_RATE STIME1.DAT STIME2.DAT" or
--       "COMPUTE_RATE STIME1.DAT STIME2.DAT SCOUNT.DAT"
--       The elapsed time of Stime2 - Stime1 will be printed out. If
--       the optional size parameter is present, then the computed
--       compilation speed in Lines/Minute/MIP will also be printed.
--
--       Rating Weight_File Result_File - Example "RATING WEIGHT DEC_VAX_V1_4"
--       The weights for the tests should be contained in WEIGHT.DAT
--       The raw results of the tests should be contained in DEC_VAX_V1_4.DAT
--       These results will be processed as explained in the Rating package.
```

Source File: SUPPORT.ADA

```
--      The results will be output to the DEC_VAX_V1_4.LIS file.
--      Notice that these argument file names do not contain a suffix.
--
```

```
WITH Common;      -- Compiler dependent information and test suite status
WITH Names;       -- Enumeration types
WITH Twine;       -- String manipulation package
WITH Parse;       -- Test file parse package
WITH Expand;      -- Code fragment expand package
WITH Count;       -- Counts Ada source lines and file sizes
WITH Times;       -- Package for timing actions
WITH Rating;      -- Result recording package
WITH PQAC_IO;     -- Centralized input and output package
```

PROCEDURE Support IS

```
TYPE Action_Type IS
  ( Parse_File, Expand_File, Count_File, Code_Size,
    Store_Time, Compute_Rate, Make_Rating, Set_Up );
```

```
SUBTYPE Initialization_Needed IS Action_Type RANGE Parse_File..Compute_Rate;
```

```
Processing_Error : EXCEPTION;
```

```
Command_In : Twine.Series;
Parameters : Twine.Series_List( 1 .. 10 );
Total      : Natural := 0;
```

```
FUNCTION "&"( Text : String; Line : Twine.Series ) RETURN String IS
BEGIN
  RETURN Text & Twine.Image( Line );
END "&";
```

```
PROCEDURE Read_Parameters( File_Name : String ) IS
  File   : PQAC_IO.File_Type;
  Pair   : Twine.Bounds;
  Buffer  : Twine.Input_Buffer;
  Size   : Natural := 0;
BEGIN
  PQAC_IO.Open_Input( File, File_Name );
  PQAC_IO.Get_Line( File, Buffer, Size );
  PQAC_IO.Close( File );
  Twine.Next_Word( Buffer( 1 .. Size ), Pair.Tail + 1, Pair );
  IF Pair.Head > Pair.Tail THEN
    PQAC_IO.Record_Error( "Parameter File " & File_Name & " Empty." );
    RAISE Processing_Error;
  END IF;
  Command_In := Twine.Create( Twine.Substring( Buffer, Pair ) );
  LOOP
    Twine.Next_Word( Buffer( 1 .. Size ), Pair.Tail + 1, Pair );
    EXIT WHEN Pair.Head > Pair.Tail;
    Total := Total + 1;
    Parameters( Total ) := Twine.Create( Twine.Substring( Buffer, Pair ) );
  END LOOP;
EXCEPTION
  WHEN OTHERS => RAISE Processing_Error;
END Read_Parameters;
```

```
FUNCTION Parameter( Item : Positive ) RETURN String IS
BEGIN
  RETURN Twine.Image( Parameters( Item ) );
END Parameter;
```

Source File: SUPPORT.ADA

```
FUNCTION Parameter
  ( Item : Positive;
    File : Names.File_Category ) RETURN String IS
BEGIN
  RETURN Common.Build_Name( Parameter( Item ), File );
END Parameter;

PROCEDURE Check_Arguments( Low : Natural; High : Natural ) IS
BEGIN
  IF Total < Low THEN
    PQAC_IO.Record_Error( "Missing Arguments to " & Command_In );
    RAISE Processing_Error;
  END IF;
  IF Total > High THEN
    PQAC_IO.Record_Error( "Extra Arguments for " & Command_In );
    RAISE Processing_Error;
  END IF;
END Check_Arguments;

PROCEDURE Run_Parse_File IS
BEGIN
  Check_Arguments( 1, 1 );
  Common.Set_Current_Test( Parameter( 1 ) );
  Parse.Parse_Tool
    ( Input_File => Parameter( 1, Names.Test ),
      Output_File => Parameter( 1, Names.Script ) );
END Run_Parse_File;

PROCEDURE Run_Expand_File IS
BEGIN
  Check_Arguments( 2, 2 );
  Expand.Expand_File
    ( Input_File => Parameter( 1 ),
      Output_File => Parameter( 2 ) );
END Run_Expand_File;

PROCEDURE Run_Count_File IS
BEGIN
  Check_Arguments( 2, 2 );
  Count.Count_File
    ( Input_File => Parameter( 1 ),
      Output_File => Parameter( 2 ) );
END Run_Count_File;

PROCEDURE Run_Code_Size IS
BEGIN
  Check_Arguments( 2, 2 );
  Count.Code_Size
    ( Input_File => Parameter( 1 ),
      Output_File => Parameter( 2 ) );
END Run_Code_Size;

PROCEDURE Run_Make_Rating IS
BEGIN
  Check_Arguments( 2, 2 );
  Rating.Rating_Tool
    ( Weight_Table => Parameter( 1, Names.Data ),
      Results_File => Parameter( 2, Names.Data ),
      Rating_Output => Parameter( 2, Names.List ) );
END Run_Make_Rating;
```



```
PROCEDURE Run_Compute_Rate IS
BEGIN
```

```
    Check_Arguments( 2, 3 );
    IF Total = 2 THEN
        Times.Compute_Rate
            ( Time_1_File => Parameter( 1 ),
              Time_2_File => Parameter( 2 ) );
    ELSE
        Times.Compute_Rate
            ( Time_1_File => Parameter( 1 ),
              Time_2_File => Parameter( 2 ),
              Count_File  => Parameter( 3 ) );
    END IF;
```

```
END Run_Compute_Rate;
```

```
PROCEDURE Run_Store_Time IS
```

```
BEGIN
    Check_Arguments( 1, 1 );
    Times.Put_Time
        ( File_Name => Parameter( 1 ),
          Time       => Times.Current_Time );
END Run_Store_Time;
```

```
FUNCTION Command RETURN Action_Type IS
```

```
    TYPE Action_Image IS
        ( Parse, Expand, Count, Store_Time, Code_Size,
          Compute_Rate, Rating, Set_Up );
```

```
    Convert : CONSTANT ARRAY( Action_Image ) OF Action_Type :=
        ( Parse      => Parse_File,
          Expand     => Expand_File,
          Count      => Count_File,
          Code_Size  => Code_Size,
          Store_Time => Store_Time,
          Compute_Rate => Compute_Rate,
          Rating     => Make_Rating,
          Set_Up     => Set_Up );
```

```
BEGIN
    RETURN Convert( Action_Image'VALUE( "" & Command_In ) );
EXCEPTION
    WHEN OTHERS =>
        PQAC_ID.Record_Error( "Unknown Command: " & Command_In );
        RAISE Processing_Error;
END Command;
```

```
BEGIN
```

```
    Read_Parameters( Common.Image( Names.Parameters ) );
    IF Command IN Initialization_Needed THEN
        Common.Initialize;
    END IF;
    CASE Command IS
        WHEN Parse_File  => Run_Parse_File;
        WHEN Expand_File => Run_Expand_File;
        WHEN Compute_Rate => Run_Compute_Rate;
        WHEN Count_File  => Run_Count_File;
        WHEN Code_Size   => Run_Code_Size;
        WHEN Make_Rating  => Run_Make_Rating;
        WHEN Store_Time   => Run_Store_Time;
        WHEN Set_Up       => Common.Create_Status_File;
    END CASE;
    IF Command IN Initialization_Needed THEN
        Common.Shut_Down;
    END IF;
```

Source File: SUPPORT.ADA

```
EXCEPTION
  WHEN Processing_Error =>
    PQAC_IO.Record_Error( "" );
    PQAC_IO.Record_Error( "Support Terminated." );
  WHEN OTHERS =>
    PQAC_IO.Record_Error( "" );
    PQAC_IO.Record_Error( "Support Abnormally Terminated." );
    RAISE;
END Support;
```

Source File: SYNTAX_.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Syntax_.Ada
--       Component: Package Specification Syntax
--       Description: This package contains subprograms for parsing the meta symbols
--                   contained in the test files, and in the files that are to be
--                   manipulated using the Expand program.
--
```

WITH Twine; -- String manipulation package

PACKAGE Syntax IS

TYPE Process_Value IS

```
( In_Error,
  Normal_Text,
  Equivalence,
  Start_Loop,
  End_Loop,
  Comment_Line,
  Begin_Select,
  End_Select,
  Compare,
  Execute,
  Compile,
  Fortran,
  New_Library );
```

```
Capacity_Error : EXCEPTION;
Statement_Error : EXCEPTION;
Name_Error : EXCEPTION;
Duplicate_Error : EXCEPTION;
Value_Error : EXCEPTION;
Count_Error : EXCEPTION;
Step_Error : EXCEPTION;
Range_Error : EXCEPTION;
```

FUNCTION Process_Value_Of(Text : String) RETURN Process_Value;

```
-- In_Error      : Line begins with "--x" or "--!" but is not one of below
-- Normal_Text   : Line does not begin with "--"
-- Equivalence    : Line begins with "--! EQUATE"
-- Start_Loop     : Line begins with "--! LOOP", "--! STEP", or "--! START"
-- End_Loop       : Line begins with "--! END"
-- Comment_Line   : Line begins with "--" but not "--x" or "--!"
-- Begin_Select   : Line begins with "--x BEGIN"
-- End_Select     : Line begins with "--x END"
-- Compare        : Line begins with "--x COMPARE"
-- Execute        : Line begins with "--x EXECUTE"
-- Compile        : Line begins with "--x COMPILE"
-- Fortran        : Line begins with "--x FORTRAN"
-- New_Library    : Line begins with "--x NEW_LIBRARY"
```

PROCEDURE Parse_Equivalence(Text : String);

```
-- The Equivalence statement in the Text line is parsed and the value
-- of the equate symbol is saved. Line must look like:
--
-- --! EQUATE Symbol IS Expression
-- Expression ::= Number ! Symbol ! Expression (x/!-!+) Expression
--
-- Statement_Error will be Raised if 'IS' is not found.
-- Duplicate_Error will be Raised if Symbol has already been equated.
-- Capacity_Error will be Raised if table is full.
```

Source File: SYNTAX_.ADA

-- Name_Error will be Raised if Expression contains undefined symbol
-- Value_Error will be Raised if Expression is not symbol or integer

PROCEDURE Parse_Compile_Name

(Text : String;
 Name : OUT Twine.Bounds;
 Options : OUT Twine.Bounds_List);

-- The Text line is parsed and the compile options are returned.
-- Name will contain the name of the file to compile.
--
-- Statement_Error will be raised if not a Compile or Fortran statement
-- Name_Error will be raised if no file name is given

PROCEDURE Parse_Loop

(Text : String;
 Loop_Copies : OUT Positive;
 Loop_Start : OUT Integer;
 Loop_Step : OUT Integer;
 Loop_Width : OUT Natural);

-- The loop statement line is parsed.
-- Statement must look like:
--
-- --! LOOP Expression START Expression STEP Expression [X]
--
-- The LOOP, START, and STEP field may be in any order. All but
-- one of them may be omitted. If omitted, default value of 1 assumed.
--
-- Count_Error will be raised if LOOP X, X < 1
-- Step_Error will be raised if STEP X, X = 0
-- Range_Error will be raised if any value of loop range negative
--
-- Loop_Copies returns LOOP x value
-- Loop_Start returns START x value
-- Loop_Step returns STEP x value
-- Loop_Width returns the maximum string image width of the loop counter.

END Syntax;

Source File: SYNTAX.ADA

```
--
--
--               The Aerospace Corporation
--
--   Production Quality Ada Compiler Test Suite Support Software
--
--
--   Author:  BAP
--   Date:   10/01/88
--   File:   Syntax.Ada
--   Component: Package Body Syntax
--   Description: ( See Package Specification Description )
--
```

WITH PQAC_IO; -- Centralized input and output package

PACKAGE BODY Syntax IS

TYPE Reserved_Word IS

```
( R_Loop,
  R_Step,
  R_Start,
  R_Begin,
  R_End,
  R_Equate,
  R_Is,
  R_Compare,
  R_Execute,
  R_Compile,
  R_Fortran,
  R_New_Library,
  Comment,
  Meta_Expand,
  Meta_Parse );
```

TYPE Element IS RECORD

 Name : Twine.Series;

 Value : Integer := 0;

END RECORD;

Symbol_Table : ARRAY(1 .. 100) OF Element;

Table_Pointer : Natural := 0;

Reserved_Words : CONSTANT ARRAY(Reserved_Word) OF Twine.Series :=

```
( R_Loop      => Twine.Create( "LOOP" ),
  R_Step      => Twine.Create( "STEP" ),
  R_Start     => Twine.Create( "START" ),
  R_Begin     => Twine.Create( "BEGIN" ),
  R_End       => Twine.Create( "END" ),
  R_Equate    => Twine.Create( "EQUATE" ),
  R_Is        => Twine.Create( "IS" ),
  R_Compare   => Twine.Create( "COMPARE" ),
  R_Execute   => Twine.Create( "EXECUTE" ),
  R_Compile   => Twine.Create( "COMPILE" ),
  R_Fortran   => Twine.Create( "FORTRAN" ),
  R_New_Library => Twine.Create( "NEW_LIBRARY" ),
  Comment     => Twine.Create( "--" ),
  Meta_Expand => Twine.Create( "--!" ),
  Meta_Parse  => Twine.Create( "--x" ) );
```

FUNCTION Equal(Name : String; Word : Reserved_Word) RETURN Boolean IS
BEGIN

 RETURN Twine.Equal(Name, Reserved_Words(Word));
END Equal;

PROCEDURE Locate_Name

Source File: SYNTAX.ADA

```
( Name      : String;
  Position  : OUT Positive;
  Found     : OUT Boolean ) IS
  Count     : Positive := 1;
BEGIN
  LOOP
    Position := Count;
    IF Count > Table_Pointer THEN
      Found := False;
      EXIT;
    END IF;
    IF Twine.Equal( Symbol_Table( Count ).Name, Name ) THEN
      Found := True;
      EXIT;
    END IF;
    Count := Count + 1;
  END LOOP;
END Locate_Name;

PROCEDURE Retrieve_Value
( Word  : String;
  Value : OUT Integer;
  Found : OUT Boolean ) IS
  Position : Positive;
  Is_Found : Boolean;
BEGIN
  Locate_Name( Word, Position, Is_Found );
  Found := Is_Found;
  IF Is_Found THEN
    Value := Symbol_Table( Position ).Value;
  END IF;
END Retrieve_Value;

PROCEDURE Add_To_Table
( Word  : String;
  Value : Integer ) IS
  Position : Positive;
  Found    : Boolean;
BEGIN
  Locate_Name( Word, Position, Found );
  IF Found THEN
    PQAC_IO.Record_Error( "Duplicate Item: " & Word );
    RAISE Duplicate_Error;
  END IF;
  Table_Pointer := Table_Pointer + 1;
  IF Table_Pointer > Symbol_Table'LAST THEN
    RAISE Capacity_Error;
  END IF;
  Symbol_Table( Table_Pointer ).Name := Twine.Create( Word );
  Symbol_Table( Table_Pointer ).Value := Value;
END Add_To_Table;

FUNCTION Parse_Value( Text : String ) RETURN Integer IS
  Found : Boolean := False;
  Pair  : Twine.Bounds;
  Sign  : Integer := 1;
  Value : Integer := 0;
  Next  : Integer := 0;

  FUNCTION Negative( Line : String; Bound : Twine.Bounds )
    RETURN Boolean IS
  BEGIN
    RETURN Bound.Head = Bound.Tail AND THEN ( Line( Bound.Head ) = '-' );
  END Negative;

  FUNCTION Operation( Line : String; Bound : Twine.Bounds )
```

```

    RETURN Boolean IS
    BEGIN
        RETURN Bound.Head = Bound.Tail AND THEN
            ( Line( Bound.Head ) = '-' OR ELSE
              Line( Bound.Head ) = '+' OR ELSE
              Line( Bound.Head ) = 'x' OR ELSE
              Line( Bound.Head ) = '/' );
    END Operation;

    FUNCTION Find_Value( Text : String; Bound : Twine.Bounds )
    RETURN Integer IS
    BEGIN
        RETURN Integer'VALUE( Twine.Substring( Text, Bound ) );
    EXCEPTION
        WHEN OTHERS =>
            PQAC_IO.Record_Error
                ( "Integer Expected: " & Twine.Substring( Text, Bound ) );
            RAISE Value_Error;
    END Find_Value;

    BEGIN
        Twine.Next_Word( Text, Text'FIRST, Pair );
        IF Negative( Text, Pair ) THEN
            Twine.Next_Word( Text, Pair.Tail + 1, Pair );
            Sign := -1;
        END IF;
        IF Pair.Head > Pair.Tail THEN
            RAISE Value_Error;
        END IF;
        IF Twine.Letter( Text( Pair.Head ) ) THEN
            Retrieve_Value( Text( Pair.Head .. Pair.Tail ), Value, Found );
            IF NOT Found THEN
                PQAC_IO.Record_Error
                    ( "Undefined Name: " & Twine.Substring( Text, Pair ) );
                RAISE Name_Error;
            END IF;
        ELSE
            Value := Find_Value( Text, Pair );
        END IF;
        Value := Value * Sign;
        Twine.Next_Word( Text, Pair.Tail + 1, Pair );
        IF Operation( Text, Pair ) THEN
            Next := Parse_Value( Text( Pair.Head + 1 .. Text'LAST ) );
            CASE Text( Pair.Head ) IS
                WHEN '+' => RETURN Value + Next;
                WHEN '-' => RETURN Value - Next;
                WHEN 'x' => RETURN Value * Next;
                WHEN '/' => RETURN Value / Next;
                WHEN OTHERS => RETURN Value;
            END CASE;
        ELSE
            RETURN Value;
        END IF;
    END Parse_Value;

    FUNCTION Process_Value_Of( Text : String ) RETURN Process_Value IS
    TYPE Reserved_Process IS ARRAY( Reserved_Word ) OF Process_Value;

    Expanding : CONSTANT Reserved_Process := -- Valid meta symbols for Expand
    ( R_Loop      => Start_Loop,
      R_Step      => Start_Loop,
      R_Start     => Start_Loop,
      R_Begin     => In_Error,
      R_End       => End_Loop,
      R_Equate    => Equivalence,
      R_Is        => In_Error,
      R_Compare   => In_Error,
      R_Execute   => In_Error,
      R_Compile   => In_Error,
      R_Fortran   => In_Error,

```

```

    R_New_Library => In_Error,
    Comment       => In_Error,
    Meta_Expand   => In_Error,
    Meta_Parse    => In_Error );

Parsing : CONSTANT Reserved_Process := -- Valid meta symbols for Parse
( R_Loop        => In_Error,
  R_Step         => In_Error,
  R_Start        => In_Error,
  R_Begin        => Begin_Select,
  R_End          => End_Select,
  R_Equate       => In_Error,
  R_Is           => In_Error,
  R_Compare      => Compare,
  R_Execute      => Execute,
  R_Compile      => Compile,
  R_Fortran      => Fortran,
  R_New_Library  => New_Library,
  Comment        => In_Error,
  Meta_Expand    => In_Error,
  Meta_Parse     => In_Error );

Pairs : Twine.Bounds_List( 1 .. 2 );
Caps  : String( Text'RANGE ) := Text;

FUNCTION Convert
( Word : String;
  Table : Reserved_Process ) RETURN Process_Value IS
BEGIN
  FOR Index IN Reserved_Word LOOP
    IF Equal( Word, Index ) THEN
      RETURN Table( Index );
    END IF;
  END LOOP;
  RETURN In_Error;
END Convert;

FUNCTION Find_Process
( Word_1 : String;
  Word_2 : String ) RETURN Process_Value IS
BEGIN
  IF Equal( Word_1, Meta_Expand ) THEN
    RETURN Convert( Word_2, Expanding );
  ELSIF Equal( Word_1, Meta_Parse ) THEN
    RETURN Convert( Word_2, Parsing );
  ELSIF Equal( Word_1, Comment ) THEN
    RETURN Comment_Line;
  ELSE
    RETURN Normal_Text;
  END IF;
END Find_Process;

BEGIN
  Twine.Upper_Case( Caps );
  Twine.Next_Words( Caps, Pairs );
  RETURN Find_Process
    ( Twine.Substring( Caps, Pairs( 1 ) ),
      Twine.Substring( Caps, Pairs( 2 ) ) );
END Process_Value_Of;

PROCEDURE Parse_Equivalence( Text : String ) IS
  Caps : String( Text'RANGE ) := Text;
  Pair_1 : Twine.Bounds;
  Pair_2 : Twine.Bounds;
BEGIN
  Twine.Upper_Case( Caps );
  Twine.Next_Word( Caps, Caps'FIRST, Pair_1 );
  WHILE Pair_1.Head <= Pair_1.Tail LOOP
    Pair_2 := Pair_1;
    Twine.Next_Word( Caps, Pair_1.Tail + 1, Pair_1 );
  END LOOP;

```



```

        EXIT WHEN Equal( Caps( Pair_1.Head .. Pair_1.Tail ), R_Is );
    END LOOP;
    IF Pair_1.Head > Pair_1.Tail THEN
        PQAC_IO.Record_Error( "Reserved Word IS not found" );
        RAISE Statement_Error; -- Reserved word IS not found.
    END IF;
    Add_To_Table
        ( Caps( Pair_2.Head .. Pair_2.Tail ),
          Parse_Value( Caps( Pair_1.Tail + 1 .. Caps'LAST ) ) );
END Parse_Equivalence;

```

```

PROCEDURE Parse_Compile_Name
( Text      : String;
  Name      : OUT Twine.Bounds;
  Options   : OUT Twine.Bounds_List ) IS
  Pairs     : Twine.Bounds_List( 1 .. Options'LAST + 3 );
  Caps      : String( Text'RANGE ) := Text;
  Value     : Process_Value;
BEGIN
    Twine.Upper_Case( Caps );
    Value := Process_Value_Of( Caps );
    IF Value /= Compile AND THEN Value /= Fortran THEN
        RAISE Statement_Error;
    END IF;
    Twine.Next_Words( Caps, Pairs );
    IF Pairs( 3 ).Head > Pairs( 3 ).Tail THEN
        RAISE Name_Error;
    END IF;
    Name := Pairs( 3 );
    Options := Pairs( 4 .. Pairs'LAST );
END Parse_Compile_Name;

```

```

PROCEDURE Parse_Loop
( Text      : String;
  Loop_Copies : OUT Positive;
  Loop_Start : OUT Integer;
  Loop_Step  : OUT Integer;
  Loop_Width : OUT Natural ) IS

    Caps : String( Text'RANGE ) := Text;

    Copies : Integer := 0;
    Start  : Integer := 0;
    Step   : Integer := 0;
    Last   : Integer := 0;

    FUNCTION Locate( Text : String; Word : Reserved_Word ) RETURN Integer IS
        Size : CONSTANT Integer := Twine.Length( Reserved_Words( Word ) ) - 1;
    BEGIN
        FOR Index IN Text'FIRST + Size .. Text'LAST LOOP
            IF Equal( Text( Index - Size .. Index ), Word ) AND THEN
                ( Index >= Text'LAST OR ELSE Text( Index + 1 ) = ' ' ) THEN
                RETURN Parse_Value( Text( Index + 1 .. Text'LAST ) );
            END IF;
        END LOOP;
        RETURN 1;
    END Locate;

BEGIN
    Twine.Upper_Case( Caps );
    Copies := Locate( Caps, R_Loop );
    Start  := Locate( Caps, R_Start );
    Step   := Locate( Caps, R_Step );
    Last   := Start + ( Copies - 1 ) * Step;
    IF Copies < 1 THEN
        RAISE Count_Error;
    ELSIF Step = 0 THEN
        RAISE Step_Error;
    ELSIF Start < 0 OR ELSE Last < 0 THEN
        RAISE Range_Error;
    END IF;

```

Source File: SYNTAX.ADA

```
    END IF;  
    Loop_Copies := Copies;  
    Loop_Start  := Start;  
    Loop_Step   := Step;  
    Loop_Width  := Integer'IMAGE( Twine.Max( Start, Last ) )'LENGTH;  
END Parse_Loop;
```

END Syntax;

Source File: TABLES_.ADA

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Tables_.Ada
--       Component: Package Specification Tables
--       Description: Compiler and Host dependent information package.
--
-- To add a compiler to the test suite domain the following actions must be
-- performed:
--
--       1. A name for the new compiler must be added to the Compiler_Domain
--          enumeration type.
--
--       2. An entry in the Compiler_Table must be made for the new compiler.
--
--       3. Entries for any new host or target architectures must be included
--          in the Host_Architecture and Target_Architecture enumeration types.
--          Entries in the Host_Table and Target_Table must also be made.
--          These tables include information for building file names.
--
--       4. If needed, a new Compiler_Vendor name must be added to that type.
--
--       5. A base compiler option must be added to the new compiler entry.
--
--       6. Compiler options must be added for each of the standardized
--          compiler options created in the Names package.
--
-- Current Table Entry Examples:
--
--       Causes "COMPILE File" to become
--
--       Dec_Vax_V1_4  --> ADA/NOCOPY_SOURCE/NONOTE_SOURCE File
--       TeleGen2_V3_15 --> TSADA/VMS/PROCEED File
--
--       Causes "COMPILE File OPTIMIZE_TIME ASSEMBLY_LISTING" to become
--
--       Dec_Vax_V1_4  --> ADA/NOCOPY_SOURCE/NONOTE_SOURCE/OPTIMIZE=TIME/LIST File
--       TeleGen2_V3_15 --> TSADA/VMS/PROCEED/OPTIMIZE=ALL/MON/LIST File
--
WITH Twine; -- String manipulation package
WITH Names; -- Enumeration types

PACKAGE Tables IS

    TYPE Compiler_Domain IS ( Dec_Vax_V1_4, TeleGen2_V3_15 );
    -- List of every possible compiler and host/target implementation
    -- of the test suite. Each item in this list has an associated
    -- Vendor, Host, and Target specified in the tables below.

    TYPE Host_Architecture IS ( Vax_8600 );
    -- List of possible host architectures.

    TYPE Target_Architecture IS ( Vax_8600, Mil_Std_1750A );
    -- List of possible target architectures.

    TYPE Compiler_Vendor IS ( Dec_Vax, Telesoft );
    -- List of possible compiler vendors

    TYPE Suffix_List IS ARRAY( Names.File_Category ) OF Twine.Series;

    TYPE Option_List IS ARRAY( Names.Compiler_Options ) OF Twine.Series;

    TYPE Host_Descriptor IS RECORD
```

Source File: TABLES_.ADA

```
Rated_MIPS : Float := 0.0; -- speed of host machine
Name       : Twine.Series; -- name of host machine
Suffix     : Suffix_list;  -- file name building information
END RECORD;
```

```
TYPE Target_Descriptor IS RECORD
  Rated_MIPS : Float := 0.0; -- speed of host machine
  Name       : Twine.Series; -- name of host machine
END RECORD;
```

```
TYPE Compiler_Descriptor IS RECORD
  Name       : Twine.Series; -- Name used in reporting results
  Vendor     : Compiler_Vendor := Compiler_Vendor'FIRST;
  Host       : Host_Architecture := Host_Architecture'FIRST;
  Target     : Target_Architecture := Target_Architecture'FIRST;
  Basic_Command : Twine.Series; -- Base command for invoking compiler
  Options     : Option_list; -- Literal parameters to be used for the
                                -- compiler options.
END RECORD;
```

```
TYPE Special_Name_Record IS RECORD -- Stores file name information
  Name : Twine.Series;
  Kind : Names.File_Category;
END RECORD;
```

Host_Table : CONSTANT ARRAY(Host_Architecture) OF Host_Descriptor :=

```
( Vax_8600 =>
  ( Rated_MIPS => 4.2,
    Name       => Twine.Create( "DEC VAX 8600" ),
    Suffix     =>
      -- These values are used to create file names
      ( Names.Test      => Twine.Create( ".TST" ),
        Names.List     => Twine.Create( ".LIS" ),
        Names.Machine  => Twine.Create( ".LIS" ),
        Names.Ada      => Twine.Create( ".ADA" ),
        Names.FORTRAN  => Twine.Create( ".FOR" ),
        Names.Expand   => Twine.Create( ".EXP" ),
        Names.Execute  => Twine.Create( ".EXE" ),
        Names.Object   => Twine.Create( ".OBJ" ),
        Names.Data     => Twine.Create( ".DAT" ),
        Names.Script   => Twine.Create( ".SCR" ) ) ) );
```

Target_Table : CONSTANT ARRAY(Target_Architecture) OF Target_Descriptor :=

```
( Vax_8600 =>
  ( Rated_MIPS => 4.2,
    Name       => Twine.Create( "DEC VAX 8600" ) ),
  Mil_Std_1750A =>
  ( Rated_MIPS => 0.0,
    Name       => Twine.Create( "MIL-STD-1750A" ) ) );
```

Compiler_Table : CONSTANT ARRAY(Compiler_Domain) OF Compiler_Descriptor :=

```
( Dec_Vax_V1_4 =>
  ( Name       => Twine.Create( "DEC VAX V1.4" ),
    Vendor     => Dec_Vax,
```

```

Host    => Vax_8600,
Target  => Vax_8600,
Basic_Command =>
    -- /NOCOPY and /NONOTE used to minimize disk usage
    Twine.Create( "ADA/NOCOPY_SOURCE/NONOTE_SOURCE" ),
Options =>
    ( Names.Syntax_Only    =>
      Twine.Create( "/SYNTAX_ONLY" ),
      Names.Optimize_Time  =>
      Twine.Create( "/OPTIMIZE=TIME" ),
      Names.Optimize_Space =>
      Twine.Create( "/OPTIMIZE=SPACE" ),
      Names.Assembly_Listing =>
      Twine.Create( "/MACHINE_CODE/LIST" ),
      Names.Compiler_Listing =>
      Twine.Create( "/LIST" ),
      Names.Statistics     =>
      -- No special command for printing all statistics
      Twine.Create( "" ),
      Names.No_Optimize    =>
      Twine.Create( "/NOOPTIMIZE" ),
      Names.Time_Compile   =>
      -- Special option, string should be "" for all compilers
      Twine.Create( "" ) ) ),

```

TeleGen2_V3_15 =>

```

( Name    => Twine.Create( "Telesoft TeleGen2 V3.15" ),
  Vendor  => Telesoft,
  Host    => Vax_8600,
  Target  => Vax_8600,
  Basic_Command =>
    Twine.Create( "TSADA/VMS/PROCEED" ),
  Options =>
    ( Names.Syntax_Only    =>
      Twine.Create( "/NOOBJECT" ),
      Names.Optimize_Time  =>
      Twine.Create( "/OPTIMIZE=ALL" ),
      Names.Optimize_Space =>
      Twine.Create( "/OPTIMIZE=NOINLINE" ),
      Names.Assembly_Listing =>
      Twine.Create( "/MACHINE_CODE=TEMPL.LIS" ),
      Names.Compiler_Listing =>
      Twine.Create( "/MON/LIST" ),
      Names.Statistics     =>
      -- No special command for printing all statistics
      Twine.Create( "" ),
      Names.No_Optimize    =>
      Twine.Create( "/NOOPTIMIZE" ),
      Names.Time_Compile   =>
      -- Special option, string should be "" for all compilers
      Twine.Create( "" ) ) ) );

```

Support_Packages : CONSTANT Twine.Series_List(1 .. 16) :=

-- List of support packages that need to be compiled for use by each test.
 -- This list must be recompiled each time the library is deleted by a test.

```

( 1 => Twine.Create( "NAMES_" ),
  2 => Twine.Create( "TWINE_" ),
  3 => Twine.Create( "TABLES_" ),
  4 => Twine.Create( "PQAC_IO_" ),
  5 => Twine.Create( "COMMON_" ),
  6 => Twine.Create( "COUNT_" ),
  7 => Twine.Create( "RESULT_" ),
  8 => Twine.Create( "COMPARE_" ),
  9 => Twine.Create( "TIMES_" ),
  10 => Twine.Create( "TWINE" ),
  11 => Twine.Create( "PQAC_IO" ),
  12 => Twine.Create( "COMMON" ),
  13 => Twine.Create( "COUNT" ),
  14 => Twine.Create( "RESULT" ),
  15 => Twine.Create( "TIMES" ),

```

Source File: TABLES_.ADA

```
16 => Twine.Create( "COMPARE" ) );
```

```
Special_Names : CONSTANT ARRAY( Names.Transfer_Files )
  OF Special_Name_Record :=
( Names.Save_Time_1      => -- Saves Start Time
  ( Name => Twine.Create( "STIME1" ),
    Kind => Names.Data ),
  Names.Save_Time_2      => -- Saves Stop Time
  ( Name => Twine.Create( "STIME2" ),
    Kind => Names.Data ),
  Names.Save_Count       => -- Saves Ada Source Line Count
  ( Name => Twine.Create( "SCOUNT" ),
    Kind => Names.Data ),
  Names.Comparison       => -- Saves test T000000 results of compilations.
  ( Name => Twine.Create( "COMPARE" ),
    Kind => Names.Data ),
  Names.Test_Result      => -- Saves raw results of each test.
                           -- RESULT is replaced with the name of the
                           -- current compiler, e.g. DEC_VAX_V1_4
  ( Name => Twine.Create( "RESULT" ),
    Kind => Names.Data ),
  Names.Parameters       => -- Used to pass parameters between programs.
  ( Name => Twine.Create( "PARAM" ),
    Kind => Names.Data ),
  Names.PQAC_State       => -- Contains the current state of the test suite.
  ( Name => Twine.Create( "STATE" ),
    Kind => Names.Data ) );
```

```
END Tables;
```

Source File: TIMES_.ADA

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Times_.Ada
--       Component: Package Specification Times
--       Description: This package contains procedures for accessing the clock
--                   for timing purposes. These procedures are used for timing
--                   inside tests, as well as for compilations. Procedures
--                   for retrieving times and source lines counts from files
--                   are also provided. These values are used in computing
--                   compilation speed in Lines/Minute/MIP
--
```

PACKAGE Times IS

TYPE Time_Type IS PRIVATE;

TYPE Time_List IS ARRAY(Positive RANGE <>) OF Time_Type;

Time_Type_First : CONSTANT Time_Type; -- Smallest value of time

Time_Type_Last : CONSTANT Time_Type; -- Largest value of time

Data_File_Error : EXCEPTION;

PROCEDURE Reset_Time;

-- Sets the clock value to 0

FUNCTION Current_Time RETURN Time_Type;

-- Returns the elapsed time since the last Reset_Time

PROCEDURE Put_Time(File_Name : String; Time : Time_Type);

-- Saves the specified Time in the given File_Name

PROCEDURE Get_Time

(File_Name : String;

Time : OUT Time_Type;

Delete_File : Boolean := False);

-- Returns the Time saved in the given File_Name. If the file does

-- not exist then Data_File_Error will be raised. If Delete_File is

-- True, then the file will be deleted after the Time is read.

PROCEDURE Get_Size

(File_Name : String;

Size : OUT Natural;

Delete_File : Boolean := False);

-- Returns the number of Ada source lines value saved in the given

-- File_Name. The number of lines is returned in Size. If the file

-- does not exist then Data_File_Error will be raised. If Delete_File

-- is True, then the file will be deleted after the Size is read.

PROCEDURE Compute_Rate

(Time_1_File : String;

Time_2_File : String;

Count_File : String := "");

-- The times from Time_1_File and Time_2_File are retrieved. The

-- elapsed time will be Time_2 - Time_1. If Count_File /= "" then

-- the number of Ada source lines saved in this file is retrieved.

-- The elapsed time is printed to the test output stream. If Count_File

-- /= "" then the number of Lines/Minute/MIP is also computed and

-- printed to the output stream.

FUNCTION Compute_Rate

(Time_1_File : String;

Source File: TIMES_.ADA

```
Time_2_File : String;
Count_File  : String := "" ) RETURN Natural;
-- Same as the previous procedure except the number of Lines/Minute/MIP
-- is also returned. If Count_File = "" then 0 is returned.

FUNCTION Image( Time : Time_Type ) RETURN String;
-- Returns a string Image of the specifiec Time value.

FUNCTION Seconds( Time : Time_Type ) RETURN Float;
-- Converts the private type Time to a Float value of seconds.

FUNCTION Elapsed( Time_1, Time_2 : Time_Type ) RETURN Time_Type;
-- Returns a Time_Type value of the elapsed time from Time_1 to Time_2.

FUNCTION Difference( Time_1, Time_2 : Time_Type ) RETURN Float;
-- Returns a float value of the elapsed time from Time_1 to Time_2.

FUNCTION Max( Time_1, Time_2 : Time_Type ) RETURN Time_Type;
FUNCTION Min( Time_1, Time_2 : Time_Type ) RETURN Time_Type;

FUNCTION Max( List : Time_List ) RETURN Time_Type;
FUNCTION Min( List : Time_List ) RETURN Time_Type;
-- Returns the Max or Min time value in the list.

FUNCTION Repeatabl( List : Time_List ) RETURN Boolean;
-- Returns true if the Repeatabl_Percent of the List is >= 95%.

FUNCTION Repeatabl_Percent( List : Time_List ) RETURN Natural;
-- Returns the percentage of the minimum value in Time_List over the
-- maximum value in the Time_List.
-- I.E. 100 * ( Min( List ) / Max( List ) )

PRIVATE

TYPE Time_Type IS RANGE 0 .. 24 * 60 * 60 * 100;

Time_Type_First : CONSTANT Time_Type := Time_Type'FIRST;
Time_Type_Last  : CONSTANT Time_Type := Time_Type'LAST;

END Times;
```


Source File: TIMES.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Times.Ada
--       Component: Package Body Times
--       Description: ( See Package Specification Description )
--
--
WITH Twine;      -- String manipulation package
WITH Names;     -- Enumeration types
WITH Common;    -- Interface to compiler information and test suite status
WITH PQAC_IO;   -- Centralized input and output package
WITH Calendar;

PACKAGE BODY Times IS

    Base_Time : Time_Type := 0;

    FUNCTION "&"( Text : String; Value : Integer ) RETURN String IS
    BEGIN
        RETURN Text & Twine.Image( Value, 8 );
    END "&";

    FUNCTION Absolute_Time RETURN Time_Type IS
        Hundred : CONSTANT Calendar.Day_Duration := 100.0;
        Seconds : Duration := Calendar.Seconds( Calendar.Clock );
    BEGIN
        RETURN Time_Type( Seconds * Hundred );
    END Absolute_Time;

    FUNCTION Name_Of( Name : String ) RETURN String IS
    BEGIN
        RETURN Common.Build_Name( Name, Names.Data );
    END Name_Of;

    PROCEDURE Reset_Time IS
    BEGIN
        Base_Time := Absolute_Time;
    END Reset_Time;

    FUNCTION Current_Time RETURN Time_Type IS
    BEGIN
        RETURN Elapsed( Base_Time, Absolute_Time );
    END Current_Time;

    PROCEDURE Put_Time( File_Name : String; Time : Time_Type ) IS
    BEGIN
        PQAC_IO.Put_Value( Name_Of( File_Name ), Integer( Time ) );
    END Put_Time;

    PROCEDURE Get_Time
        ( File_Name : String;
          Time       : OUT Time_Type;
          Delete_File : Boolean := False ) IS
```

```

    Result : Integer;
BEGIN
    PQAC_IO.Get_Value( Name_Of( File_Name ), Result );
    Time := Time_Type( Result );
    IF Delete_File THEN
        PQAC_IO.Delete_File( Name_Of( File_Name ) );
    END IF;
EXCEPTION
    WHEN OTHERS => RAISE Data_File_Error;
END Get_Time;

PROCEDURE Get_Size
( File_Name : String;
  Size      : OUT Natural;
  Delete_File : Boolean := False ) IS
BEGIN
    PQAC_IO.Get_Value( Name_Of( File_Name ), Size );
    IF Delete_File THEN
        PQAC_IO.Delete_File( Name_Of( File_Name ) );
    END IF;
EXCEPTION
    WHEN OTHERS => RAISE Data_File_Error;
END Get_Size;

PROCEDURE Compute_Rate
( Time_1_File : String;
  Time_2_File : String;
  Count_File : String := "" ) IS
    Result : Natural;
BEGIN
    Result := Compute_Rate( Time_1_File, Time_2_File, Count_File );
END Compute_Rate;

FUNCTION Compute_Rate
( Time_1_File : String;
  Time_2_File : String;
  Count_File : String := "" ) RETURN Natural IS
    Time_1 : Time_Type;
    Time_2 : Time_Type;
    Time_3 : Time_Type;

    FUNCTION Print_Ratios( Time : Time_Type ) RETURN Natural IS
        Source : Natural := 0;
        Ratio_1 : Natural := 0;
        Ratio_2 : Natural := 0;
        Minutes : Float := 0.0;
    BEGIN
        PQAC_IO.New_Line;
        PQAC_IO.Put_Line( Common.Image( Common.Host_Banner ) );
        PQAC_IO.New_Line;
        Get_Size( Count_File, Source );
        PQAC_IO.Put_Line( "Size: " & Source & " Ada Source Lines" );
        Minutes := Float( Time ) / 6000.00;
        Ratio_1 := Natural( Float( Source ) / Minutes );
        PQAC_IO.Put_Line( "Speed: " & Ratio_1 & " Lines/Minute" );
        Ratio_2 := Natural( Float( Ratio_1 ) / Common.Host_Rated_MIPS );
        PQAC_IO.Put_Line( " " & Ratio_2 & " Lines/Minute/MIPS" );
        PQAC_IO.New_Line;
        RETURN Ratio_2;
    EXCEPTION
        WHEN OTHERS =>
            PQAC_IO.Put_Line( "Error: " & 0 & " Lines/Minute" );
            RETURN 0;
    END Print_Ratios;
END Compute_Rate;

```

Source File: TIMES.ADA

```
BEGIN
  Get_Time( Time_1_File, Time_1 );
  Get_Time( Time_2_File, Time_2 );
  Time_3 := Elapsed( Time_1, Time_2 );
  PQAC_IO.Put_Line( Image( Time_3 ) & " Elapsed Time" );
  IF Count_File = "" THEN
    RETURN 0;
  ELSE
    RETURN Print_Ratios( Time_3 );
  END IF;
END Compute_Rate;

FUNCTION Image( Time : Time_Type ) RETURN String IS
BEGIN
  RETURN Twine.Image( Float( Time ) / 100.00, 8, 2 ) & " Seconds";
END Image;

FUNCTION Seconds( Time : Time_Type ) RETURN Float IS
BEGIN
  RETURN Float( Time ) / 100.0;
END Seconds;

FUNCTION Elapsed( Time_1, Time_2 : Time_Type ) RETURN Time_Type IS
BEGIN
  IF Time_1 <= Time_2 THEN
    RETURN Time_2 - Time_1;
  ELSE -- Clock has wrapped around so must adjust
    RETURN Time_2 + ( Time_Type'LAST - Time_1 );
  END IF;
END Elapsed;

FUNCTION Difference( Time_1, Time_2 : Time_Type ) RETURN Float IS
BEGIN
  RETURN Float( Time_1 - Time_2 ) / 100.0;
END Difference;

FUNCTION Max( Time_1, Time_2 : Time_Type ) RETURN Time_Type IS
BEGIN
  IF Time_1 > Time_2 THEN
    RETURN Time_1;
  ELSE
    RETURN Time_2;
  END IF;
END Max;

FUNCTION Min( Time_1, Time_2 : Time_Type ) RETURN Time_Type IS
BEGIN
  IF Time_1 < Time_2 THEN
    RETURN Time_1;
  ELSE
    RETURN Time_2;
  END IF;
END Min;

FUNCTION Max( List : Time_List ) RETURN Time_Type IS
  Result : Time_Type := Time_Type_First;
BEGIN
  FOR Index IN List'RANGE LOOP
    Result := Max( Result, List( Index ) );
  END LOOP;
END Max;
```

Source File: TIMES.ADA

```
END LOOP;  
RETURN Result;  
END Max;
```

```
FUNCTION Min( List : Time_List ) RETURN Time_Type IS  
  Result : Time_Type := Time_Type_Last;  
BEGIN  
  FOR Index IN List'RANGE LOOP  
    Result := Min( Result, List( Index ) );  
  END LOOP;  
  RETURN Result;  
END Min;
```

```
FUNCTION Repeatable( List : Time_List ) RETURN Boolean IS  
BEGIN  
  RETURN Repeatable_Percent( List ) >= 95;  
END Repeatable;
```

```
FUNCTION Repeatable_Percent( List : Time_List ) RETURN Natural IS  
  Low   : Integer := Integer( Min( List ) );  
  High  : Integer := Integer( Max( List ) );  
  
  FUNCTION Min( A, B : Integer ) RETURN Integer IS  
  BEGIN  
    IF A < B THEN RETURN A;  
    ELSE RETURN B;  
    END IF;  
  END Min;  
  
  BEGIN  
    RETURN Min( 100, 100 * ( Low + 1 ) / High );  
  EXCEPTION  
    WHEN OTHERS => RETURN 0;  
  END Repeatable_Percent;
```

```
END Times;
```

Source File: TWINE_.ADA

```
--
--
--               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:   BAP
--       Date:    10/01/88
--       File:    Twine_.Ada
--       Component: Package Specification Twine
-- Description:   This package has been created for the manipulation of strings.
--               This package was necessitated by the need for tables of
--               containing strings of varying lengths, and for passing
--               arrays containing strings of different sizes as arguments.
--               The names Twine and Series were chosen because they are
--               short words and are synonyms for String.
```

PACKAGE Twine IS

```
Input_Size  : CONSTANT Natural := 132;
Output_Size : CONSTANT Natural := 80;
```

```
SUBTYPE Input_Buffer  IS String( 1 .. Input_Size );
SUBTYPE Output_Buffer IS String( 1 .. Output_Size );
```

```
TYPE Series IS PRIVATE;      -- dynamic string entity
```

```
TYPE Bounds IS RECORD      -- used for designating substrings
  Head : Positive := 1;
  Tail : Natural  := 0;
END RECORD;
```

```
TYPE Series_List IS ARRAY( Positive RANGE <> ) OF Series;
TYPE Bounds_List IS ARRAY( Positive RANGE <> ) OF Bounds;
```

```
Illegal_Bounds  : EXCEPTION;
Undefined_Series : EXCEPTION;
```

```
FUNCTION Create( Text : String ) RETURN Series;
-- A Series value for the given string is returned.
```

```
FUNCTION Length( Line : Series ) RETURN Natural;
-- The length of the string is returned.
```

```
FUNCTION Area( Line : Series ) RETURN Bounds;
-- Returns Bounds'( 1, Length( Line ) )
```

```
FUNCTION Image( Line : Series ) RETURN String;
-- Returns the String value of the Series.
```

```
FUNCTION Element( Line : Series; Position : Positive ) RETURN Character;
-- Returns the character in the specified Position of Line.
```

```
PROCEDURE Delete( Line : IN OUT Series );
-- The Line is deallocated in memory.
```

```
PROCEDURE Next_Word( Text : String; Head : Positive; Pair : OUT Bounds );
PROCEDURE Next_Word( Line : Series; Head : Positive; Pair : OUT Bounds );
-- The given Text or Line is scanned starting in position Head. Blanks
-- are skipped until a non-blank character is found. Pair contains
-- the head and tail position of the next word on the line. Words are
-- single special characters, or alpha-numeric characters terminated
-- with a space, end-of-line, or special character. If no words are
-- found on the line after Head, then Pair is returned as ( X, X - 1 )
-- where X is the last position in the line.
```

```
PROCEDURE Next_Words( Text : String; Pairs : OUT Bounds_List );
PROCEDURE Next_Words( Line : Series; Pairs : OUT Bounds_List );
-- Words are scanned from the Text or Line and their boundry points are
-- placed into Pairs. If there are more elements in Pairs then words
-- on the line, then the excess elements of Pairs will be of the form
```

Source File: TWINE_.ADA

```
-- ( X, X - 1 ).

FUNCTION Substring( Text : String; Pair : Bounds ) RETURN String;
FUNCTION Substring( Line : Series; Pair : Bounds ) RETURN String;
-- Returns the substring of the line specified at the positions in Pair.

PROCEDURE Upper_Case( Text : IN OUT String );
PROCEDURE Upper_Case( Line : IN OUT Series );
-- Replaces all of the lower case letters in the line with upper case.

PROCEDURE Copy( Line : IN OUT Series; Pair : Bounds; Text : String );
PROCEDURE Copy( Line : IN OUT Series; Pair : Bounds; Text : Series );
PROCEDURE Copy( Line : IN OUT Series; Text : String );
PROCEDURE Copy( Line : IN OUT Series; Text : Series );
-- Copies the specified Text or Pair substring of Text into Line.

FUNCTION Equal( Text : String; Line : Series ) RETURN Boolean;
FUNCTION Equal( Line : Series; Text : String ) RETURN Boolean;
FUNCTION Equal( Line : Series; Text : Series ) RETURN Boolean;
-- Returns True if the string values are equal.

FUNCTION Equal( Line : Series; Pair : Bounds; Text : String )
RETURN Boolean;
FUNCTION Equal( Line : Series; Pair : Bounds; Text : Series )
RETURN Boolean;
FUNCTION Equal( Line : String; Pair : Bounds; Text : String )
RETURN Boolean;
FUNCTION Equal( Line : String; Pair : Bounds; Text : Series )
RETURN Boolean;
-- Returns True if the specified Pair substring of Line is equal to Text.

FUNCTION Clip( Text : String ) RETURN String;
-- Returns a string with Text stripped of leading and trailing spaces.

FUNCTION Image
( Value : Float;
  Field : Positive := 1;
  Aft   : Positive := 1;
  Exp   : Natural  := 0 ) RETURN String;
-- Returns the String Image of Value of size Field.

FUNCTION Image
( Value : Integer;
  Field : Positive := 1 ) RETURN String;
-- Returns the String Image of Value of size Field.

FUNCTION Zeroed_Image
( Value : Natural;
  Field : Positive := 1 ) RETURN String;
-- Returns the String Image of Value of size Field with leading spaces
-- filled with zeros.

FUNCTION Min( A : Integer; B : Integer ) RETURN Integer;
FUNCTION Max( A : Integer; B : Integer ) RETURN Integer;

FUNCTION Digit( Char : Character ) RETURN Boolean; -- '0'..'1'
FUNCTION Letter( Char : Character ) RETURN Boolean; -- 'a'..'z', 'A'..'Z'
FUNCTION Alpha( Char : Character ) RETURN Boolean; -- Digit or Letter
FUNCTION Sign( Char : Character ) RETURN Boolean;  -- '+' or '-'

PRIVATE

  TYPE Text_Record( Size : Natural := 0 );

  TYPE Series IS ACCESS Text_Record;

END Twine;
```

Source File: TWINE.ADA

```
--
--                               The Aerospace Corporation
--
--       Production Quality Ada Compiler Test Suite Support Software
--
--
--       Author:  BAP
--       Date:    10/01/88
--       File:    Twine.Ada
--       Component: Package Body Twine
--       Description: ( See Package Specification Description )
--
WITH Text_IO;
WITH Unchecked_Deallocation;

PACKAGE BODY Twine IS

    TYPE Text_Record( Size : Natural := 0 ) IS RECORD
        Text : String( 1 .. Size ) := ( OTHERS => ' ' );
    END RECORD;

    FUNCTION Create( Text : String ) RETURN Series IS
        Line : Series := NEW Text_Record( Text'LENGTH );
    BEGIN
        Line.Text := Text;
        RETURN Line;
    END Create;

    FUNCTION Length( Line : Series ) RETURN Natural IS
    BEGIN
        RETURN Line.Size;
    EXCEPTION
        WHEN Constraint_Error => RAISE Undefined_Series;
    END Length;

    FUNCTION Area( Line : Series ) RETURN Bounds IS
    BEGIN
        RETURN ( 1, Line.Size );
    EXCEPTION
        WHEN Constraint_Error => RAISE Undefined_Series;
    END Area;

    FUNCTION Image( Line : Series ) RETURN String IS
    BEGIN
        RETURN Line.Text;
    EXCEPTION
        WHEN Constraint_Error => RAISE Undefined_Series;
    END Image;

    FUNCTION Element( Line : Series; Position : Positive ) RETURN Character IS
    BEGIN
        IF Line = NULL THEN
            RAISE Undefined_Series;
        END IF;
        RETURN Line.Text( Position );
    EXCEPTION
        WHEN Constraint_Error => RAISE Illegal_Bounds;
    END Element;

    PROCEDURE Delete( Line : IN OUT Series ) IS
```

Source File: TWINE.ADA

```
PROCEDURE Deallocate IS NEW Unchecked_Deallocation( Text_Record, Series );
BEGIN
  Deallocate( Line );
END Delete;
```

```
PROCEDURE Next_Word( Text : String; Head : Positive; Pair : OUT Bounds ) IS
  TYPE Class_Type IS ( Alpha, Extra, Space );
```

```
  Next : Natural := Head;
  Class : Class_Type;
```

```
FUNCTION Class_Of( Char : Character ) RETURN Class_Type IS
BEGIN
```

```
  CASE Char IS
```

```
    WHEN '0' .. '9' => RETURN Alpha;
    WHEN 'a' .. 'z' => RETURN Alpha;
    WHEN 'A' .. 'Z' => RETURN Alpha;
    WHEN '-'       => RETURN Alpha;
    WHEN '.'       => RETURN Alpha;
    WHEN ','       => RETURN Space;
    WHEN OTHERS   => RETURN Extra;
```

```
  END CASE;
```

```
END Class_Of;
```

```
BEGIN
```

```
  WHILE Next IN Text'RANGE AND THEN Text( Next ) = ' ' LOOP
    Next := Next + 1;
```

```
  END LOOP;
```

```
  Pair.Head := Next;
```

```
  Pair.Tail := Next - 1;
```

```
  IF Next IN Text'RANGE AND THEN Text( Next ) /= ' ' THEN
```

```
    Class := Class_Of( Text( Next ) );
```

```
    WHILE ( Next + 1 ) IN Text'RANGE
```

```
      AND THEN Class_Of( Text( Next + 1 ) ) = Class LOOP
```

```
        Next := Next + 1;
```

```
    END LOOP;
```

```
    Pair.Tail := Next;
```

```
  END IF;
```

```
END Next_Word;
```

```
PROCEDURE Next_Word( Line : Series; Head : Positive; Pair : OUT Bounds ) IS
BEGIN
```

```
  Next_Word( Line.Text, Head, Pair );
```

```
EXCEPTION
```

```
  WHEN Constraint_Error => RAISE Undefined_Series;
```

```
END Next_Word;
```

```
PROCEDURE Next_Words( Text : String; Pairs : OUT Bounds_List ) IS
  Pair : Bounds := ( Text'FIRST, Text'FIRST - 1 );
```

```
BEGIN
```

```
  FOR Index IN Pairs'RANGE LOOP
```

```
    Next_Word( Text, Pair.Tail + 1, Pair );
```

```
    Pairs( Index ) := Pair;
```

```
  END LOOP;
```

```
END Next_Words;
```

```
PROCEDURE Next_Words( Line : Series; Pairs : OUT Bounds_List ) IS
BEGIN
```

```
  Next_Words( Line.Text, Pairs );
```

```
EXCEPTION
```

```
  WHEN Constraint_Error => RAISE Undefined_Series;
```

```
END Next_Words;
```



```

FUNCTION Substring( Text : String; Pair : Bounds ) RETURN String IS
BEGIN
    RETURN Text( Pair.Head .. Pair.Tail );
EXCEPTION
    WHEN Constraint_Error => RAISE Illegal_Bounds;
END Substring;

```

```

FUNCTION Substring( Line : Series; Pair : Bounds ) RETURN String IS
BEGIN
    RETURN Substring( Line.Text, Pair );
EXCEPTION
    WHEN Constraint_Error => RAISE Undefined_Series;
END Substring;

```

```

PROCEDURE Upper_Case( Text : IN OUT String ) IS
    FUNCTION Upper_Case( Char : Character ) RETURN Character IS
    BEGIN
        IF Char IN 'a' .. 'z' THEN
            RETURN Character'VAL( Character'POS( Char ) - 32 );
        ELSE
            RETURN Char;
        END IF;
    END Upper_Case;
BEGIN
    FOR Index IN Text'RANGE LOOP
        Text( Index ) := Upper_Case( Text( Index ) );
    END LOOP;
END Upper_Case;

```

```

PROCEDURE Upper_Case( Line : IN OUT Series ) IS
BEGIN
    Upper_Case( Line.Text );
EXCEPTION
    WHEN Constraint_Error => RAISE Undefined_Series;
END Upper_Case;

```

```

PROCEDURE Copy( Line : IN OUT Series; Pair : Bounds; Text : String ) IS
    Size : Natural := Min( Text'LENGTH, Pair.Tail - Pair.Head + 1 );
BEGIN
    IF Line = NULL THEN
        RAISE Undefined_Series;
    END IF;
    Line.Text( Pair.Head .. Pair.Tail ) := ( OTHERS => ' ' );
    Line.Text( Pair.Head .. Pair.Head + Size - 1 ) :=
        Text( Text'FIRST .. Text'FIRST + Size - 1 );
EXCEPTION
    WHEN Constraint_Error => RAISE Illegal_Bounds;
END Copy;

```

```

PROCEDURE Copy( Line : IN OUT Series; Pair : Bounds; Text : Series ) IS
BEGIN
    Copy( Line, Pair, Image( Text ) );
EXCEPTION
    WHEN Constraint_Error => RAISE Undefined_Series;
END Copy;

```

```

PROCEDURE Copy( Line : IN OUT Series; Text : Series ) IS
BEGIN
    Copy( Line, ( 1, Line.Size ), Text );

```

Source File: TWINE.ADA

```
EXCEPTION
  WHEN Constraint_Error => RAISE Undefined_Series;
END Copy;
```

```
PROCEDURE Copy( Line : IN OUT Series; Text : String ) IS
BEGIN
  Copy( Line, ( 1, Line.Size ), Text );
EXCEPTION
  WHEN Constraint_Error => RAISE Undefined_Series;
END Copy;
```

```
FUNCTION Equal( Text : String; Line : Series ) RETURN Boolean IS
BEGIN
  RETURN Text = Line.Text;
EXCEPTION
  WHEN Constraint_Error => RAISE Undefined_Series;
END Equal;
```

```
FUNCTION Equal( Line : Series; Text : String ) RETURN Boolean IS
BEGIN
  RETURN Text = Line.Text;
EXCEPTION
  WHEN Constraint_Error => RAISE Undefined_Series;
END Equal;
```

```
FUNCTION Equal( Line : Series; Text : Series ) RETURN Boolean IS
BEGIN
  RETURN Text.Text = Line.Text;
EXCEPTION
  WHEN Constraint_Error => RAISE Undefined_Series;
END Equal;
```

```
FUNCTION Equal( Line : Series; Pair : Bounds; Text : String )
  RETURN Boolean IS
BEGIN
  IF Line = NULL THEN
    RAISE Undefined_Series;
  END IF;
  RETURN Text = Line.Text( Pair.Head .. Pair.Tail );
EXCEPTION
  WHEN Constraint_Error => RAISE Illegal_Bounds;
END Equal;
```

```
FUNCTION Equal( Line : Series; Pair : Bounds; Text : Series )
  RETURN Boolean IS
BEGIN
  RETURN Equal( Line, Pair, Text.Text );
EXCEPTION
  WHEN Constraint_Error => RAISE Undefined_Series;
END Equal;
```

```
FUNCTION Equal( Line : String; Pair : Bounds; Text : String )
  RETURN Boolean IS
BEGIN
  RETURN Line( Pair.Head .. Pair.Tail ) = Text;
EXCEPTION
  WHEN Constraint_Error => RAISE Illegal_Bounds;
END Equal;
```

```

FUNCTION Equal( Line : String; Pair : Bounds; Text : Series )
  RETURN Boolean IS
BEGIN
  RETURN Equal( Line, Pair, Text.Text );
EXCEPTION
  WHEN Constraint_Error => RAISE Undefined_Series;
END Equal;

```

```

FUNCTION Clip( Text : String ) RETURN String IS
BEGIN
  FOR Head IN Text'RANGE LOOP
    IF Text( Head ) /= ' ' THEN
      FOR Tail IN REVERSE Text'RANGE LOOP
        IF Text( Tail ) /= ' ' THEN
          RETURN Text( Head .. Tail );
        END IF;
      END LOOP;
    END IF;
  END LOOP;
  RETURN "";
END Clip;

```

```

FUNCTION Image
  ( Value : Float;
    Field : Positive := 1;
    Aft   : Positive := 1;
    Exp   : Natural  := 0 ) RETURN String IS
  Text : Output_Buffer := ( OTHERS => ' ' );
  PACKAGE Flt_IO IS NEW Text_IO.Float_IO( Float );
BEGIN
  Flt_IO.Put( Text, Value, Aft, Exp );
  FOR Index IN REVERSE 1 .. Text'LAST - Field LOOP
    IF Text( Index ) = ' ' THEN
      RETURN Text( Index + 1 .. Text'LAST );
    END IF;
  END LOOP;
  RETURN Text;
EXCEPTION
  WHEN OTHERS => RAISE Illegal_Bounds;
END Image;

```

```

FUNCTION Image
  ( Value : Integer;
    Field : Positive := 1 ) RETURN String IS
  Text : Output_Buffer := ( OTHERS => ' ' );
  PACKAGE Int_IO IS NEW Text_IO.Integer_IO( Integer );
BEGIN
  Int_IO.Put( Text, Value );
  FOR Index IN REVERSE 1 .. Text'LAST - Field LOOP
    IF Text( Index ) = ' ' THEN
      RETURN Text( Index + 1 .. Text'LAST );
    END IF;
  END LOOP;
  RETURN Text;
EXCEPTION
  WHEN OTHERS => RAISE Illegal_Bounds;
END Image;

```

```

FUNCTION Zeroed_Image
  ( Value : Natural;
    Field : Positive := 1 ) RETURN String IS

```

Source File: TWINE.ADA

```
Text : String( 1 .. Field ) := Image( Value, Field );
BEGIN
  FOR Index IN Text'RANGE LOOP
    IF Text( Index ) = ' ' THEN
      Text( Index ) := '0';
    END IF;
  END LOOP;
  RETURN Text;
EXCEPTION
  WHEN OTHERS => RAISE Illegal_Bounds;
END Zeroed_Image;

FUNCTION Min( A : Integer; B : Integer ) RETURN Integer IS
BEGIN
  IF A < B THEN
    RETURN A;
  ELSE
    RETURN B;
  END IF;
END Min;

FUNCTION Max( A : Integer; B : Integer ) RETURN Integer IS
BEGIN
  IF A > B THEN
    RETURN A;
  ELSE
    RETURN B;
  END IF;
END Max;

FUNCTION Digit( Char : Character ) RETURN Boolean IS
BEGIN
  RETURN Char IN '0' .. '9';
END Digit;

FUNCTION Letter( Char : Character ) RETURN Boolean IS
BEGIN
  RETURN Char IN 'A' .. 'Z' OR ELSE Char IN 'a' .. 'z';
END Letter;

FUNCTION Alpha( Char : Character ) RETURN Boolean IS
BEGIN
  RETURN Digit( Char ) OR ELSE Letter( Char );
END Alpha;

FUNCTION Sign( Char : Character ) RETURN Boolean IS
BEGIN
  RETURN Char = '-' OR ELSE Char = '+';
END Sign;

END Twine;
```

11. PQAC TEST FILES (T000000 through T080800)

The test files listed here are contained in the following pages.

These files contain all of the source code for the PQAC tests.

T000000.TST	T030305.TST	T030803.TST	T050300.TST	T060703.TST
T010100.TST	T030306.TST	T030804.TST	T060100.TST	T060801.TST
T020100.TST	T030307.TST	T040101.TST	T060201.TST	T060802.TST
T020200.TST	T030308.TST	T040102.TST	T060202.TST	T060900.TST
T020300.TST	T030309.TST	T040103.TST	T060203.TST	T061001.TST
T020401.TST	T030310.TST	T040104.TST	T060301.TST	T061002.TST
T020402.TST	T030311.TST	T040105.TST	T060302.TST	T061003.TST
T020403.TST	T030401.TST	T040106.TST	T060303.TST	T061004.TST
T020501.TST	T030402.TST	T040201.TST	T060304.TST	T061101.TST
T020502.TST	T030403.TST	T040202.TST	T060305.TST	T061102.TST
T030101.TST	T030404.TST	T040203.TST	T060306.TST	T061201.TST
T030102.TST	T030405.TST	T040204.TST	T060307.TST	T061202.TST
T030103.TST	T030406.TST	T040205.TST	T060308.TST	T061203.TST
T030104.TST	T030407.TST	T040206.TST	T060309.TST	T061204.TST
T030105.TST	T030408.TST	T040207.TST	T060310.TST	T061205.TST
T030106.TST	T030501.TST	T040208.TST	T060401.TST	T061206.TST
T030201.TST	T030502.TST	T040209.TST	T060402.TST	T061207.TST
T030202.TST	T030601.TST	T040301.TST	T060403.TST	T061208.TST
T030203.TST	T030602.TST	T040302.TST	T060404.TST	T070100.TST
T030204.TST	T030701.TST	T040303.TST	T060501.TST	T070200.TST
T030205.TST	T030702.TST	T040304.TST	T060502.TST	T070300.TST
T030206.TST	T030703.TST	T040305.TST	T060503.TST	T070400.TST
T030207.TST	T030704.TST	T050101.TST	T060504.TST	T070500.TST
T030208.TST	T030705.TST	T050102.TST	T060505.TST	T080100.TST
T030209.TST	T030706.TST	T050103.TST	T060506.TST	T080200.TST
T030301.TST	T030707.TST	T050104.TST	T060601.TST	T080300.TST
T030302.TST	T030708.TST	T050201.TST	T060602.TST	T080400.TST
T030303.TST	T030709.TST	T050202.TST	T060603.TST	T080500.TST
T030304.TST	T030801.TST	T050203.TST	T060701.TST	T080600.TST
T030802.TST	T050204.TST	T060702.TST	T080700.TST	T080800.TST

Source File: T000000.TST

```
-- T000000
--
-- The following code is for use in tests T020401, T020402, T020403,
-- T020501 and T020502.
--
--* COMPILE COMPADA
--* COMPARE OPTIMIZE_SPACE TEMP1
--* COMPARE OPTIMIZE_TIME TEMP2
--* COMPARE NO_OPTIMIZE TEMP3
--* COMPARE SYNTAX_ONLY TEMP4
--! EQUATE Count IS 10
--! EQUATE Steps IS 50
--! EQUATE Sizes IS 25
--* BEGIN DEC_VAX_V1_4
--! EQUATE Digit IS 15;
--! EQUATE Words IS 8;
--* END
--* BEGIN TELEGEN2_V3_15
--! EQUATE Digit IS 8;
--! EQUATE Words IS 4;
--* END
PROCEDURE CompAda IS

  --! LOOP 1 START Digit [1]
  TYPE Real IS DIGITS [1];
  --! END [1]

  --! LOOP 1 START Sizes [1]
  Size : CONSTANT := [1];
  --! END [1]

  TYPE A0 IS ARRAY( 1 .. Size ) OF Real;
  TYPE A1 IS ARRAY( 1 .. Size, 1 .. Size ) OF Real;
  TYPE A2 IS ARRAY( 1 .. Size, 1 .. Size, 1 .. Size ) OF Real;

  V1 : A1;
  V2 : A2;

  --! LOOP Count [1]
  PROCEDURE Init[1]( X : IN OUT A1; Y : IN OUT A2 ) IS
  BEGIN
    FOR I IN 1 .. Size LOOP
      FOR J IN 1 .. Size LOOP
        X( J, I ) := 0.[1] / Real( I + J );
      END LOOP;
    END LOOP;
    FOR I IN 1 .. Size LOOP
      FOR J IN 1 .. Size LOOP
        FOR K IN 1 .. Size LOOP
          Y( I, J, K ) := X( I, J ) * X( J, K ) + X( K, I );
        END LOOP;
      END LOOP;
    END LOOP;
  END Init[1];

  --! END [1]

  --! LOOP Count [1]
  PROCEDURE Work[1]( X : IN OUT A1; Y : IN OUT A2 ) IS
  V : A0;
  Z : A2;
  T : Real;
  BEGIN
    T := 0.0;
    --! LOOP Steps [2]
    T := T + [1].0 / [2].0;
    --! END [2]
    FOR I IN 1 .. Size LOOP
      V( I ) := T / Real( I );
    END LOOP;
    FOR I IN 1 .. Size LOOP
      FOR J IN 1 .. Size LOOP
        T := T + V( I ) * V( J );
        IF T > [1].0 THEN
```

```

        T := [1].0 / T;
      END IF;
      X( I, J ) := X( J, I ) + T + V( I ) + V( J );
      IF X( I, J ) > [1].0 THEN
        X( I, J ) := [1].0 / X( I, J );
      END IF;
      FOR K IN 1 .. Size LOOP
        Z( K, J, I ) := Y( I, J, K ) * X( I, K ) + X( J, K );
      END LOOP;
    END LOOP;
  FOR I IN 1 .. Size LOOP
    FOR J IN 1 .. Size LOOP
      FOR K IN 1 .. Size LOOP
        Y( I, J, K ) := Z( K, J, I );
        IF Z( K, J, I ) > [1].0 THEN
          Y( I, J, K ) := [1].0 / Z( K, J, I );
        END IF;
      END LOOP;
    END LOOP;
  END LOOP;
END Work[1];

--! END [1]

```

BEGIN

```

--! LOOP Count [1]
Init[1]( V1, V2 );
Work[1]( V1, V2 );
--! END [1]

```

END CompAda;

--* FORTRAN COMPFOR
--* COMPARE OPTIMIZE_TIME TEMP5

```

--! LOOP 1 START Words [1]
REAL*[1] V1
REAL*[1] V2
--! END [1]

INTEGER Size

--! LOOP 1 START Sizes [1]
PARAMETER ( Size = [1] )
--! END [1]

DIMENSION V1( Size, Size )
DIMENSION V2( Size, Size, Size )

--! LOOP Count [1]
CALL Init[1]( V1, V2 )
CALL Work[1]( V1, V2 )
--! END [1]

END

```

--! LOOP Count [1]

```

SUBROUTINE Init[1] ( X, Y )

--! LOOP 1 START Words [2]
REAL*[2] X
REAL*[2] Y
--! END [2]

INTEGER I
INTEGER J
INTEGER K
INTEGER Size

```

Source File: T000000.TST

```
--! LOOP 1 START Sizes [2]
PARAMETER ( Size = [2] )
--! END [2]

DIMENSION X( Size, Size )
DIMENSION Y( Size, Size, Size )

DO 20 I=1,Size
DO 10 J=1,Size

X( J, I ) = 0.[1] / Real( I + J )

10 CONTINUE
20 CONTINUE

DO 50 I = 1, Size
DO 40 J = 1, Size
DO 30 K = 1, Size

Y( I, J, K ) = X( I, J ) * X( J, K ) + X( K, I )

30 CONTINUE
40 CONTINUE
50 CONTINUE

RETURN
END

--! END [1]
```

C-----

--! LOOP Count [1]

```
SUBROUTINE Work[1] ( X, Y )

--! LOOP 1 START Words [2]
REAL*[2] V
REAL*[2] X
REAL*[2] Y
REAL*[2] Z
REAL*[2] T
--! END [2]

INTEGER I
INTEGER J
INTEGER K
INTEGER Size

--! LOOP 1 START Sizes [2]
PARAMETER ( Size = [2] )
--! END [2]

DIMENSION V( Size )
DIMENSION X( Size, Size )
DIMENSION Y( Size, Size, Size )
DIMENSION Z( Size, Size, Size )

T = 0.0
--! LOOP Steps [2]
T = T + [1].0 / [2].0
--! END [2]

DO 10 I = 1, Size

V( I ) = T / Real( I )

10 CONTINUE

DO 40 I = 1, Size
DO 30 J = 1, Size

T = T + V( I ) * V( J )
IF ( T .GT. [1].0 ) T = [1].0 / T
X( I, J ) = X( J, I ) + T + V( I ) + V( J )
```


Source File: T000000.TST

```
      IF ( X( I, J ) .GT. [1].0 ) X( I, J ) = [1].0 / X( I, J )
      DO 20 K = 1, Size
      Z( K, J, I ) = Y( I, J, K ) * X( I, K ) + X( J, K )
20  CONTINUE
30  CONTINUE
40  CONTINUE

      DO 70 I = 1, Size
      DO 60 J = 1, Size
      DO 50 K = 1, Size

      Y( I, J, K ) = Z( K, J, I )
      IF ( Z( K, J, I ) .GT. [1].0 ) Y( I, J, K ) = [1].0 / Z( K, J, I )

50  CONTINUE
60  CONTINUE
70  CONTINUE

      RETURN
      END
```

--! END [1]

C-----
--* COMPILE T000000
--* EXECUTE T000000
WITH Times;
WITH Result;
WITH Compare;
WITH PQAC_IO;

PROCEDURE T000000 IS

```
File   : PQAC_IO.File_Type;
ASCII  : Natural := Character'POS( '1' );

Size   : Natural;
```

```
FUNCTION "&"( Text : String; Number : Integer ) RETURN String IS
BEGIN
  RETURN Text & Result.Image( Number, 8 );
END "&"
```

```
FUNCTION "&"( Text : String; Time : Times.Time_Type ) RETURN String IS
BEGIN
  RETURN Text & Times.Image( Time );
END "&"
```

```
FUNCTION "&"( Text : String; Number : Float ) RETURN String IS
BEGIN
  RETURN Text & Integer( Number * 100.0 );
END "&"
```

```
PROCEDURE Process( Kind : String; Name : String ) IS
  Before : Times.Time_Type;
  After  : Times.Time_Type;
  Temp   : String( 1 .. 20 ) := ( OTHERS => ' ' );
  Rate   : Natural;
  Diff   : Float;
  Size   : Result.File_Length;
BEGIN
  Temp( 1 .. Kind'LENGTH ) := Kind;
  Result.Print( "Statistics for " & Kind & ":" );
  Rate := Times.Compute_Rate( Name & "A", Name & "B", Name & "C" );
  Times.Get_Time( Name & "D", Before );
  Times.Get_Time( Name & "E", After );
  Times.Get_Size( Name & "F", Size );
```

Source File: T000000.TST

```
        Diff := Times.Difference( After, Before );
        Result.Print( "" );
        PQAC_IO.Put_Line( File, Temp & " " & Rate & Diff & Size );
    END Process;

BEGIN
    PQAC_IO.Open_Output( File, Compare.Result_File );
    FOR Index IN Compare.Compiler_Version LOOP
        Process( Compare.Compiler_Version'IMAGE( Index ),
            "TEMP" & Character'VAL( ASCII ) );
        ASCII := ASCII + 1;
    END LOOP;
    Process( "FORTRAN", "TEMP5" );
    PQAC_IO.Close( File );
    Result.Passed( "T000000", True );
EXCEPTION
    WHEN OTHERS => Result.Inconclusive( "T000000", "Program Error." );
END T000000;
```

Source File: T010100.TST -

```
-- T010100
--
-- An Ada source statement shall be defined to mean: a basic declaration,
-- a record component declaration, a simple statement, a compound statement,
-- an entry declaration, terminate alternative, WITH clause, USE clause,
-- generic parameter declaration, proper body or body stub, representation
-- clause, alignment clause, or component clause.
--
-- Method:
--
-- Definition.
--
--* COMPILE T010100
--* EXECUTE T010100
WITH Result;
PROCEDURE T010100 IS
BEGIN
    Result.Not_Applicable( "T010100", "Definition." );
END T010100;
```

Source File: T020100.TST

```
-- T020100
--
-- All performance requirements of this section shall be met using the
-- programs of the test suite formulated by the Performance Issues Working
-- Group (PIWG) of the SIGAda Users' Committee.
--
-- Method:
--
-- Definition. This requirement is impossible to follow, as there
-- are no programs in the PIWG test suite which satisfy all of the
-- requirements of this section. New programs have been written.
--
--* COMPILE T020100
--* EXECUTE T020100
WITH Result;
PROCEDURE T020100 IS
BEGIN
    Result.Not_Applicable( "T020100", "Definition." );
END T020100;
```

Source File: T020200.TST

```
-- T020200
--
-- The requirements in this section assume a single compilation unit without
-- any context clauses (WITH clauses) or generic instantiations.
--
-- Method:
--
-- Definition.
--
--* COMPILE T020200
--* EXECUTE T020200
WITH Result;
PROCEDURE T020200 IS
BEGIN
    Result.Not_Applicable( "T020200", "Definition." );
END T020200;
```

Source File: T020300.TST

```
-- T020300
--
-- All speed requirements of this section shall be measured in terms of
-- elapsed (wall-clock) time.
--
-- Method:
--
-- Definition.
--
--* COMPILE T020300
--* EXECUTE T020300
WITH Result;
PROCEDURE T020300 IS
BEGIN
    Result.Not_Applicable( "T020300", "Definition." );
END T020300;
```

Source File: T020401.TST

```
-- T020401
--
-- The compiler shall compile a syntactically and semantically correct Ada
-- program of at least 200 Ada source statements at a rate of at least 200
-- statements per minute (elapsed time), for each 1 MIPS of rated processing
-- speed of the specified host computer, while meeting the object code
-- requirements in 2.5.1 and 2.5.2.
--
-- Method:
--
-- The data collected from compiling the comparison code is examined.
--
--* COMPILE T020401
--* EXECUTE T020401
WITH Result;
WITH Compare;
PROCEDURE T020401 IS
    Space_Percent : Result.Percentage;
    Speed_Percent : Result.Percentage;

    FUNCTION "&"( Text : String; Item : Integer ) RETURN String IS
    BEGIN
        RETURN Text & Result.Image( Item, 3 );
    END "&";

    PROCEDURE Show( A, B, Ave : Result.Percentage ) IS
    BEGIN
        Result.Print( "" );
        Result.Print( "Combined Success = " & A & " + " & B & " / 2 = " & Ave );
        Result.Print( "" );
        Result.Passed( "T020401", Ave );
    END Show;

BEGIN
    Space_Percent :=
        Compare.Percentage
        ( Compiler_Option      => Compare.Optimize_Space,
          Minimum_Compile_Rate => 200,
          Minimum_Size_Percent => 130,
          Minimum_Time_Percent => 0 );
    Speed_Percent :=
        Compare.Percentage
        ( Compiler_Option      => Compare.Optimize_Time,
          Minimum_Compile_Rate => 200,
          Minimum_Size_Percent => 0,
          Minimum_Time_Percent => 115 );
    Show( Space_Percent, Speed_Percent, ( Space_Percent + Speed_Percent ) / 2 );
EXCEPTION
    WHEN Compare.Undefined_Data =>
        Result.Inconclusive( "T020401", "FORTRAN Comparisons not run." );
END T020401;
```

Source File: T020402.TST

```
-- T020402
--
-- The compiler shall compile a syntactically and semantically correct Ada
-- program of at least 200 Ada source statements at a rate of at least 500
-- statements per minute (elapsed time), for each 1 MIPS of rated processing
-- speed of the specified host computer, in the absence of requirements on
-- object code efficiency.
--
-- Method:
--
-- The data collected from compiling the comparison code is examined.
--
--* COMPILE T020402
--* EXECUTE T020402
WITH Result;
WITH Compare;
PROCEDURE T020402 IS
BEGIN
    Result.Passed( "T020402",
        Compare.Percentage
        ( Compiler_Option      => Compare.No_Optimize,
          Minimum_Compile_Rate => 500,
          Minimum_Size_Percent => 0,
          Minimum_Time_Percent => 0 ) );
EXCEPTION
    WHEN Compare.Undefined_Data =>
        Result.Inconclusive( "T020402", "FORTRAN Comparisons not run." );
END T020402;
```


Source File: T020403.TST

```
-- T020403
--
-- The compiler shall compile a syntactically and semantically correct Ada
-- program of at least 200 Ada source statements at a rate of at least 1000
-- statements per minute (elapsed time), for each 1 MIPS of rated processing
-- speed of the specified host computer, with no requirement to generate
-- object code.
--
-- Method:
--
-- The data collected from compiling the comparison code is examined.
--
--* COMPILE T020403
--* EXECUTE T020403
WITH Result;
WITH Compare;
PROCEDURE T020403 IS
BEGIN
    Result.Passed( "T020403",
        Compare.Percentage
        ( Compiler_Option      => Compare.Syntax_Only,
          Minimum_Compile_Rate => 1000,
          Minimum_Size_Percent =>    0,
          Minimum_Time_Percent =>    0 ) );
EXCEPTION
    WHEN Compare.Undefined_Data =>
        Result.Inconclusive( "T020403", "FORTRAN Comparisons not run." );
END T020403;
```

Source File: T020501.TST

```
-- T020501
--
-- The compiler shall produce an object code program that requires no more
-- than 30% additional target computer memory space over an equivalent program
-- written in assembly language.
--
-- Method:
--
-- The data collected from compiling the comparison code is examined.
--
--* COMPILE T020501
--* EXECUTE T020501
WITH Result;
WITH Compare;
PROCEDURE T020501 IS
BEGIN
    Result.Passed
        ( "T020501",
          Compare.Percentage
            ( Compiler_Option => Compare.Optimize_Space,
              Minimum_Compile_Rate => 0,
              Minimum_Size_Percent => 130,
              Minimum_Time_Percent => 0 ) );
EXCEPTION
    WHEN Compare.Undefined_Data =>
        Result.Inconclusive( "T020501", "FORTRAN Comparisons not run." );
END T020501;
```

Source File: T020502.TST

```
-- T020502
--
-- The compiler shall produce an object code program that requires no more
-- than 15% additional execution time over an equivalent program written in
-- assembly language.
--
-- Method:
--
-- The data collected from compiling the comparison code is examined.
--
--* COMPILE T020502
--* EXECUTE T020502
WITH Result;
WITH Compare;
PROCEDURE T020502 IS
BEGIN
    Result.Passed
        ( "T020502",
          Compare.Percentage
            ( Compiler_Option => Compare.Optimize_Time,
              Minimum_Compile_Rate => 0,
              Minimum_Size_Percent => 0,
              Minimum_Time_Percent => 115 ) );
EXCEPTION
    WHEN Compare.Undefined_Data =>
        Result.Inconclusive( "T020502", "FORTRAN Comparisons not run." );
END T020502;
```

Source File: T030101.TST

```
-- T030101
--
-- library units in a program library = 2048
--
-- Method:
--
-- Compile 2044 packages, each package containing one constant.
-- There are 4 library units used in the support software used here.
-- The packages are split between four files to avoid large file size
-- problems. The compiler shall be determined to have passed this
-- requirement if the compilation succeeds without error.
--
--* NEW_LIBRARY
--! EQUATE Split IS 4
--! EQUATE Count IS 2044 / Split
--! LOOP Split [1]
--* COMPILE T030101
--! LOOP Count [2]
PACKAGE Package_[1]_[2] IS
    Constant_[2]: CONSTANT := [2];
END Package_[1]_[2];
--! END [2]
--! END [1]
--* COMPILE T030101
--* EXECUTE T030101
--! LOOP 1 START Split [1]
--! LOOP 1 START Count [2]
WITH Result;
WITH Package_[1]_[2];
--! END [2]
--! END [1]
PROCEDURE T030101 IS
BEGIN
    Result.Passed( "T030101", 100 );
END T030101;
--* NEW_LIBRARY
```

Source File: T030102.TST

```
-- T030102
--
-- compilation units in a program = 1024
--
-- Method:
--
-- Compile 30 packages each WITHing 33 other packages declaring one constant.
-- The support software includes 3 compilations units. Combined with the
-- main procedure we have  $30 \times 33 + 30 + 3 + 1 = 1024$  compilation units.
--
--* COMPILE T030102
--* EXECUTE T030102
--! EQUATE Outer IS 30
--! EQUATE Inner IS 33
--! LOOP Outer [1]
--! LOOP Inner [2]
PACKAGE Package_[1]_Sub_[2] IS
    Item_[2] : CONSTANT := [2];
END Package_[1]_Sub_[2];
--! END [2]
--! LOOP Inner [2]
WITH Package_[1]_Sub_[2];
--! END [2]
PACKAGE Package_[1] IS
    Item_[1] : CONSTANT := Package_[1]_Sub_1.Item_1;
END Package_[1];
--! END [1]

--! LOOP Outer [1]
WITH Package_[1];
--! END [1]
WITH Result;
PROCEDURE T030102 IS
    Variable : Integer;
BEGIN
    --! LOOP Outer [1]
    Variable := Package_[1].Item_[1];
    --! END [1]
    Result.Passed( "T030102", 100 );
END T030102;
```

Source File: T030103.TST

```
-- T030103
--
-- Ada source statements in a program = 2,500,000
--
-- Method:
--
-- Compile a program consisting of 2.5 million lines of code without
-- violating any of the other requirements in the PQAC definition.
-- The code is split into several different files before compilation.
--
-- Statement Count:
--   In First File:      Size2 * Size3 + Size2
--   In Last File:       Size2 * 2 + 3
--   Each Middle File:   104 * Size2 * Size3 + 3 * Size2
--   Number of Middle Files: Size1
--
--   Total Statements:   104 * Size1 * Size2 * Size3 +
--                       3 * Size1 * Size2 +
--                       Size2 * Size3 +
--                       3 * Size2 + 3
--
--   With Size1 = 20, Size2 = 30, and Size3 = 40 there are 2,499,9093
--   total statements. There are at least 1000 statements in the
--   support code bringing the total to 2.5 million lines of code.
--
--* NEW_LIBRARY
--* COMPILE FIRST TIME_COMPILE
--! EQUATE Size1 IS 20
--! EQUATE Size2 IS 30
--! EQUATE Size3 IS 40
--
--! LOOP 1 Start 0 [1]
--! LOOP Size2 [2]
PACKAGE Package_11_2 IS
--! LOOP Size3 [3]
  A[3] : Boolean;
--! END [3]
END Package_11_2;
--! END [2]
--! END [1]
--
--! LOOP Size1 [1]
--* COMPILE MIDDLE TIME_COMPILE
--! LOOP Size2 [2]
PACKAGE Package_11_2 IS
--! LOOP Size3 [3]
  A[3] : Boolean;
--! END [3]
END Package_11_2;
--! END [2]
--
--! LOOP Size2 [2]
WITH Package_11_2;
PACKAGE BODY Package_11_2 IS
--! LOOP Size3 [3]
  PROCEDURE P[3]( X : Boolean ) IS -- 102 Lines
    A : Boolean := X;
    B : Boolean := NOT A;
    C : Boolean := NOT B;
    D : Boolean := NOT C;
    E : Boolean := NOT D;
  BEGIN
    --! LOOP 18 [4]
    A:=B;B:=C;C:=D;D:=E;E:=NOT A;
    --! END [4]
    A[3]:=X OR A;
    A[3]:=X OR B;
    A[3]:=X OR C;
    A[3]:=X OR D;
    A[3]:=X OR E;
    Package_11_2.A[3]:=X;
  END P[3];
--! END [3]
BEGIN
```

Source File: T030103.TST

```
--! LOOP Size3 [3]
P[3]( True );
--! END [3]
END Package_[1]_[2];
--! END [2]
--! END [1]

--* COMPILE T030103 TIME_COMPILE
--* EXECUTE T030103
--! LOOP 1 START Size1 [1]
--! LOOP Size2 [2]
WITH Package_[1]_[2];
--! END [2]
--! END [1]
WITH Result;
PROCEDURE T030103 IS
BEGIN
  --! LOOP 1 START Size1 [1]
  --! LOOP Size2 [2]
  Package_[1]_[2].A1 := Package_[1]_[2].A2;
  --! END [2]
  --! END [1]
  Result.Passed( "T030103", 100 );
END T030103;
--* NEW_LIBRARY
```

Source File: T030104.TST

```
-- T030104
--
-- maximum size (in words) of a program = 2,500,000
--
-- Method:
--
-- Compile a program containing 400 objects of a size large enough to
-- produce 2,500,000 words in the object code. If the compilation and
-- execution succeed without error and the size of the object code is
-- greater or equal to 2,500,000 words the test has passed.
--
--* NEW_LIBRARY
--* COMPILE T030104 TIME_COMPILE
--* EXECUTE T030104
--! EQUATE Count IS 400
WITH Result;
WITH System;
PROCEDURE T030104 IS
  --! LOOP 1 START Count [1]
  Count      : CONSTANT := [1];
  --! END [1]

  Goal_Size : CONSTANT := 2_500_000;           -- words
  Word_Size : CONSTANT := System.Storage_Unit; -- bits per word
  Base_Size : CONSTANT := Integer'SIZE;        -- bits
  Unit_Size : CONSTANT := Goal_Size * Word_Size / Base_Size;
  Increment : CONSTANT := Unit_Size / Count;

  TYPE Big_Array IS ARRAY( 1 .. Increment ) OF Integer;

  TYPE Big_Record IS RECORD
    List : Big_Array := ( OTHERS => 1 );
  END RECORD;

  --! LOOP Count [1]
  R_[1] : Big_Record := ( List => ( OTHERS => [1] ) );
  --! END [1]

  Size_Found : Result.File_Length;
BEGIN
  --! LOOP Count [1]
  R_[1].List := ( OTHERS => [1] + 1 );
  --! END [1]

  Result.Print_Code_Size( "T030104", Size_Found );
  IF Size_Found >= Goal_Size THEN
    Result.Passed( "T030104", 100 );
  ELSE
    Result.Inconclusive( "T030104" );
  END IF;
END T030104;
--* NEW_LIBRARY
```


Source File: T030105.TST

```
-- T030105
--
-- Elaborate PRAGMAs = 512
--
-- Method:
--
-- Compile 16 packages, with each package WITHing and giving an
-- ELABORATION order for 32 other packages. This results in a total of
--  $32 \times 16 = 512$  ELABORATION PRAGMAs used. All of these packages declare
-- one constant. The 16 top level packages are then WITHed by a main
-- level procedure in order to include all 512 ELABORATION programs in
-- one program. This requirement cannot be tested by using all 512
-- ELABORATION PRAGMAs on one compilation unit since the number of
-- "library units WITHed by a compilation unit = 256" is tested separately.
-- The compiler shall be determined to have passed this requirement if
-- the compilation and execution succeeds without error.
--
--* COMPILE T030105
--* EXECUTE T030105
--! EQUATE Outer IS 16
--! EQUATE Inner IS 32
--! LOOP Outer [1]
--! LOOP Inner [2]
PACKAGE Package_[1] Sub_[2] IS
    Const_[2] : CONSTANT := [2];
END Package_[1] Sub_[2];
--! END [2]
--! LOOP Inner [2]
WITH Package_[1] Sub_[2];
--! END [2]
    --! START Inner LOOP Inner STEP -1 [2]
    PRAGMA Elaborate ( Package_[1] Sub_[2] );
    --! END [2]
PACKAGE Package_[1] IS
    Const_[1] : CONSTANT := Package_[1] Sub_[1].Const_1;
END Package_[1];
--! END [1]
--! LOOP Outer [1]
WITH Package_[1];
--! END [1]
WITH Result;
PROCEDURE T030105 IS
    Variable : Integer;
BEGIN
    --! LOOP Outer [1]
    Variable := Package_[1].Const_[1];
    --! END [1]
    Result.Passed( "T030105", 100 );
END T030105;
```

Source File: T030106.TST

```
-- T030106
--
-- width of source line (& length of identifier) = 120
--
-- Method:
--
-- Compile a procedure containing an identifier of length 120. The
-- identifier is used in an assignment statement. The compiler shall be
-- determined to have passed this requirement if the compilation and
-- execution succeeds without error.
--
--* COMPILE T030106
--* EXECUTE T030106
WITH Result;
PROCEDURE T030106 IS
A123456789B123456789C123456789D123456789E123456789F123456789G123456789H123456789
: Integer := 1;
BEGIN
A123456789B123456789C123456789D123456789E123456789F123456789G123456789H123456789
:=
A123456789B123456789C123456789D123456789E123456789F123456789G123456789H123456789
+
A123456789B123456789C123456789D123456789E123456789F123456789G123456789H123456789
;
Result.Passed( "T030106", 100 );
END T030106;
```

Source File: T030201.TST

```
-- T030201
--
-- library units in a single context clause = 16
--
-- Method:
--
-- Compile 16 packages, each package containing one constant. These
-- packages are then withed by a main procedure using a single context
-- clause. The compiler shall be determined to have passed this
-- requirement if the compilation and execution succeeds without error.
--
--* COMPILE T030201
--* EXECUTE T030201
--! EQUATE Iter IS 16
--! LOOP Iter [1]
PACKAGE Package_[1] IS
    Constant_[1] : CONSTANT := [1];
END Package_[1];
--! END [1]
WITH
    --! LOOP Iter-1 [1]
        Package_[1],
    --! END [1]
    --! START Iter LOOP 1 [1]
        Package_[1];
    --! END [1]
WITH Result;
PROCEDURE T030201 IS
    I : Integer := 0;
BEGIN
    --! LOOP Iter [1]
        I := I + Package_[1].Constant_[1];
    --! END [1]
    Result.Passed( "T030201", 100 );
END T030201;
```

Source File: T030202.TST

```
-- T030202
--
-- library units WITHed by a compilation unit = 256
--
-- Method:
--
-- Compile 255 packages, each package containing one constant. WITH the
-- packages into a main procedure using 255 WITH statements. Including the
-- support software, the number of units WITHed will be 256. The compiler
-- shall be determined to have passed this requirement if the compilation
-- and execution succeeds without error.
--
--* COMPILE T030202
--* EXECUTE T030202
--! EQUATE Iter IS 255
--! LOOP Iter [1]
PACKAGE Package_[1] IS
  Constant_[1] : CONSTANT := [1];
END Package_[1];
--! END [1]
--! LOOP Iter [1]
WITH Package_[1];
--! END [1]
WITH Result;
PROCEDURE T030202 IS
  I : Integer := 0;
BEGIN
  --! LOOP Iter [1]
  I := Package_[1].Constant_[1] - I;
  --! END [1]
  Result.Passed( "T030202", 100 );
END T030202;
--* NEW_LIBRARY
```

Source File: T030203.TST

```
-- T030203
--
-- external names = 4096
--
-- Method:
--
-- Compile 16 packages, each package containing 15 enumeration types
-- with 16 values. WITH and USE these packages in another package body.
-- Number of names external to the package body:
--
--      package names      16          = 16
--      type names         16 * 15      = 240
--      enumeration values  16 * 15 * 16 = 3840
--      TOTAL              4096
--
-- The compiler shall be determined to have passed this requirement
-- if the compilation and execution succeeds without error.
--
--* COMPILE T030203
--* EXECUTE T030203
--! EQUATE Inner  IS 16
--! EQUATE Middle IS 15
--! EQUATE Outer  IS 16
--! LOOP Outer [1]
PACKAGE Package_ [1] IS
  --! LOOP Middle [2]
  TYPE Pack_ [1]_Enum_ [2] IS (
    --! LOOP Inner-1 [3]
    Enum_ [1]_ [2]_ [3],
    --! END [3]
    --! START Inner LOOP 1 [3]
    Enum_ [1]_ [2]_ [3] );
    --! END [3]
  --! END [2]
END Package_ [1];
--! END [1]

PACKAGE Test_Package IS
  FUNCTION Successful RETURN Boolean;
END Test_Package;

--! LOOP Outer [1]
WITH Package_ [1]; USE Package_ [1];
--! END [1]
PACKAGE BODY Test_Package IS
  --! LOOP Outer [1]
  --! LOOP Middle [2]
  Variable_ [1]_ [2] : Pack_ [1]_Enum_ [2];
  --! END [2]
  --! END [1]
  FUNCTION Successful RETURN Boolean IS
  BEGIN
    --! LOOP Outer [1]
    --! LOOP Middle [2]
    Variable_ [1]_ [2] := Enum_ [1]_ [2]_ 1;
    --! END [2]
    --! END [1]
    RETURN True;
  END Successful;
END Test_Package;

WITH Result;
WITH Test_Package;
PROCEDURE T030203 IS
BEGIN
  IF Test_Package.Successful THEN
    Result.Passed( "T030203", 100 );
  ELSE
    Result.Passed( "T030203", 0 );
  END IF;
END T030203;
```

Source File: T030204.TST

```
-- T030204
--
-- Ada source statements in a compilation unit = 4096
--
-- Method:
--
-- Declare a variable. Perform 4092 assignments to this variable.
-- There are 4 other statements in the procedure for a total of 4096.
-- The compiler shall be determined to have passed this requirement
-- if the compilation and execution succeeds without error.
--
--* COMPILE T030204 TIME_COMPILE
--* EXECUTE T030204
--! EQUATE Iter IS 4092 / 2
WITH Result;
PROCEDURE T030204 IS
  Variable : Integer := 0;
BEGIN
  --! LOOP Iter [1]
  Variable := [1];
  Variable := Variable + 1;
  --! END [1]
  Result.Passed( "T030204", 100 );
END T030204;
```

Source File: T030205.TST

```
-- T030205
--
-- identifiers (including those in WITHed units) = 4096
--
-- Method:
--
-- Compile 8 packages, each package declares 255 integers. WITH these 8
-- packages into a main procedure with 2047 integers declared for a total
-- of:
--
--      package identifiers                =      8
--      external integer identifiers      8 * 255 = 2040
--      procedure identifier               =      1
--      internal integer identifiers      = 2045
--      package name result               =      1
--      procedure name result.passed      =      1
--      TOTAL                             4096
--
-- The compiler shall be determined to have passed this requirement if
-- the compilation and execution succeeds without error.
--
--* COMPILE T030205
--* EXECUTE T030205
--! EQUATE Half IS 2045
--! EQUATE Inner IS 255
--! EQUATE Outer IS 8
--! LOOP Outer [1]
PACKAGE Package_ [1] IS
  --! LOOP Inner [2]
  Int_ [1]_ [2] : Integer;
  --! END [2]
END Package_ [1];
--! END [1]
--! LOOP Outer [1]
WITH Package_ [1]; USE Package_ [1];
--! END [1]
WITH Result;
PROCEDURE T030205 IS
  --! LOOP Half [1]
  Var_ [1] : Integer := [1];
  --! END [1]
BEGIN
  --! LOOP Outer [1]
  Int_ [1]_ 1 := Var_ [1];
  --! END [1]
  Result.Passed( "T030205", 100 );
END T030205;
```

Source File: T030206.TST

```
-- T030206
--
-- declarations (total) in a compilation unit = 4096
--
-- Method:
--
-- Compile a procedure containing 4095 integer declarations. The
-- procedure itself is a declaration for a total of 4096. The compiler
-- shall be determined to have passed this requirement if the compilation
-- and execution succeeds without error.
--
--* COMPILE T030206
--* EXECUTE T030206
--! EQUATE Iter IS 4095
WITH Result;
PROCEDURE T030206 IS
  --! LOOP Iter [1]
  Int_[1] : Integer := [1];
  --! END [1]
BEGIN
  Int_1 := 1;
  Result.Passed( "T030206", 100 );
END T030206;
```


Source File: T030207.TST

```
-- T030207
--
-- type declarations = 1024
--
-- Method:
--
-- Compile a procedure containing 512 range declarations and 512 array
-- declarations. The compiler shall be determined to have passed this
-- requirement if the compilation and execution succeeds without error.
--
--* COMPILE T030207
--* EXECUTE T030207
--! EQUATE Iter IS 1024
--! EQUATE Half IS Iter / 2
WITH Result;
PROCEDURE T030207 IS
  --! LOOP Half [1]
  TYPE Range_[1] IS RANGE 1 .. [1];
  --! END [1]
  --! LOOP Half [1]
  TYPE Array_[1] IS ARRAY( 1 .. [1] ) OF Boolean;
  --! END [1]
  --! LOOP Half [1]
  R_[1] : Range_[1] := [1];
  --! END [1]
  --! LOOP Half [1]
  A_[1] : Array_[1] := ( OTHERS => True );
  --! END [1]
BEGIN
  R_1 := 1;
  A_1( 1 ) := False;
  Result.Passed( "T030207", 100 );
END T030207;
```

Source File: T030208.TST

```
-- T030208
--
-- subtype declarations of a single type = 1024
--
-- Method:
--
-- Declare 1024 subtypes of integer. The compiler shall be determined to
-- have passed this requirement if the compilation and execution succeeds
-- without error.
--
--* COMPILE T030208
--* EXECUTE T030208
--! EQUATE Iter IS 1024
WITH Result;
PROCEDURE T030208 IS
  --! LOOP Iter [1]
    SUBTYPE Subrange_[1] IS Integer RANGE 1 .. [1];
  --! END [1]
  --! LOOP Iter [1]
    S_[1] : Subrange_[1] := [1];
  --! END [1]
BEGIN
  S_1 := 1;
  S_10 := 10;
  Result.Passed( "T030208", 100 );
END T030208;
```

Source File: T030209.TST

```
-- T030209
--
-- literals in a compilation unit = 1024
--
-- Method:
--
-- Assign a variable of type real with 1024 distinct literals. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030209
--* EXECUTE T030209
--! EQUATE Iter IS 1024
WITH Result;
PROCEDURE T030209 IS
    TYPE Real IS DIGITS 8;
    Variable : Real;
BEGIN
    --! LOOP Iter [1]
    Variable := [1].0;
    --! END [1]
    Result.Passed( "T030209", 100 );
END T030209;
--* NEW_LIBRARY
```

Source File: T030301.TST

```
-- T030301
--
-- depth of nesting of program units = 64
--
-- Method:
--
-- Compile 64 levels of nesting for both a package and a function. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030301
--* EXECUTE T030301
--! EQUATE Iter IS 64
--! LOOP Iter [1]
PACKAGE Pack_1 IS
  --! END [1]
  Variable : CONSTANT := 1;
  --! START Iter LOOP Iter STEP -1 [1]
END Pack_1;
--! END [1]

WITH Pack_1;
WITH Result;
PROCEDURE T030301 IS
  I : Integer;

  --! LOOP Iter-1 [1]
  FUNCTION Func_1 RETURN Integer IS
  --! END [1]
  BEGIN
    RETURN 1;
  --! START Iter-1 LOOP 1 [1]
  END Func_1;
  --! END [1]
  --! START Iter-2 LOOP Iter-2 STEP -1 [1]
  BEGIN
    RETURN Func_1[1];
  END Func_1;
  --! END [1]

  BEGIN
    I := Func_1;
    Result.Passed( "T030301", 100 );
  END T030301;
```

Source File: T030302.TST

```
-- T030302
--
-- depth of nesting of blocks = 64
--
-- Method:
--
-- Compile a procedure with 64 nested levels of labeled blocks. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030302
--* EXECUTE T030302
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030302 IS
  I, J : Integer := 1;
BEGIN
  --! LOOP Iter [1]
  Block_[1]: BEGIN
    --! END [1]
    I := J; J := I;
    --! START Iter LOOP Iter STEP -1 [1]
  END Block_[1];
  --! END [1]
  Result.Passed( "T030302", 100 );
END T030302;
```

Source File: T030303.TST

```
-- T030303
--
-- depth of nesting of case statements = 64
--
-- Method:
--
-- 64 nested case statements each containing one choice. The compiler
-- shall be determined to have passed this requirement if the compilation
-- and execution succeeds without error.
--
--* COMPILE T030303
--* EXECUTE T030303
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030303 IS
    Choice : Integer := 1;
BEGIN
    --! LOOP Iter [1]
    CASE Choice IS -- [1]
        WHEN OTHERS =>
            --! END [1]
            Choice := 0;
    --! START Iter LOOP Iter STEP -1 [1]
    END CASE; -- [1]
    --! END [1]
    Result.Passed( "T030303", 100 );
END T030303;
```

Source File: T030304.TST

```
-- T030304
--
-- depth of nesting of loop statements = 64
--
-- Method:
--
-- 64 nested while loop statements. The compiler shall be determined to
-- have passed this requirement if the compilation and execution succeeds
-- without error.
--
--* COMPILE T030304
--* EXECUTE T030304
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030304 IS
    Choice : Integer := 1;
BEGIN
    --! LOOP Iter [1]
    WHILE Choice = 1 LOOP
        --! END [1]
        Choice := 2;
        --! START Iter LOOP Iter STEP -1 [1]
    END LOOP;
    --! END [1]
    Result.Passed( "T030304", 100 );
END T030304;
```

Source File: T030305.TST

```
-- T030305
--
-- depth of nesting of if statements = 256
--
-- Method:
--
-- Compile a procedure containing 256 nested IF statements. The
-- compiler shall be determined to have passed this requirement if
-- the compilation and execution succeeds without error.
--
--* COMPILE T030305
--* EXECUTE T030305
--! EQUATE Iter IS 256
WITH Result;
PROCEDURE T030305 IS
    Choice : Integer := 0;
BEGIN
    --! LOOP Iter [1]
    IF Choice < [1] THEN
        --! END [1]
        Choice := 2;
        --! START Iter LOOP Iter STEP -1 [1]
    END IF;
    --! END [1]
    Result.Passed( "T030305", 100 );
END T030305;
```


Source File: T030306.TST

```
-- T030306
--
-- elsif alternatives = 256
--
-- Method:
--
-- Compile a procedure containing one IF statement with 256 ELSIFs. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030306
--* EXECUTE T030306
--! EQUATE Iter IS 256
WITH Result;
PROCEDURE T030306 IS
    Choice : Integer := 1;
BEGIN
    IF Choice = 0 THEN
        Choice := Choice + 1;
        --! LOOP Iter [1]
    ELSIF Choice = [1] THEN
        Choice := Choice + 1;
        --! END [1]
    END IF;
    Result.Passed( "T030306", 100 );
END T030306;
```

Source File: T030307.TST

```
-- T030307
--
-- exception declarations in a frame = 256
--
-- Method:
--
-- Declare 256 exceptions in a procedure. The compiler shall be
-- determined to have passed this requirement if the compilation and
-- execution succeeds without error.
--
--* COMPILE T030307
--* EXECUTE T030307
--! EQUATE Iter IS 256
WITH Result;
PROCEDURE T030307 IS
  I : Integer;
  --! LOOP Iter [1]
  Exception_[1] : EXCEPTION;
  --! END [1]
  FUNCTION Something RETURN Natural IS
  BEGIN
    RETURN 1000;
  END Something;
BEGIN
  I := Something;
  CASE I IS
    --! LOOP Iter [1]
    WHEN [1] => RAISE Exception_[1];
    --! END [1]
    WHEN OTHERS => I := 0;
  END CASE;
  Result.Passed( "T030307", 100 );
END T030307;
```

Source File: T030308.TST

```
-- T030308
--
-- exception handlers in a frame = 256
--
-- Method:
--
-- Declare 64 exceptions in each of 4 packages. WITH these packages into
-- a procedure that handles all 256 exceptions. The compiler shall be
-- determined to have passed this requirement if the compilation and
-- execution succeeds without error.
--
--* COMPILE T030308
--* EXECUTE T030308
--! EQUATE Iter IS 256
--! EQUATE Outer IS 4
--! EQUATE Inner IS Iter / Outer
--! LOOP Outer [1]
PACKAGE Package_11 IS
  --! LOOP Inner [2]
  Exception_11_2 : EXCEPTION;
  --! END [2]
END Package_11;
--! END [1]
--! LOOP Outer [1]
WITH Package_11; USE Package_11;
--! END [1]
WITH Result;
PROCEDURE T030308 IS
  I : Integer := 0;
BEGIN
  RAISE Exception_1_1;
EXCEPTION
  --! LOOP Outer [1]
  --! LOOP Inner [2]
  WHEN Exception_11_2 => I := I + [1];
  --! END [2]
  --! END [1]
  Result.Passed( "T030308", 100 );
END T030308;
```

Source File: T030309.TST

```
-- T030309
--
-- declarations in a declarative part = 1024
--
-- Method:
--
-- Compile a procedure containing 1024 integer declarations. The compiler
-- shall be determined to have passed this requirement if the compilation
-- and execution succeeds without error.
--
--* COMPILE T030309
--* EXECUTE T030309
--! EQUATE Iter IS 1024
--! EQUATE Part IS Iter / 4
WITH Result;
PROCEDURE T030309 IS
  --! LOOP Iter [1]
  Int_[1] : Integer := [1];
  --! END [1]
BEGIN
  --! LOOP Part STEP 4 [1]
  Int_[1] := Int_[1+1] + Int_[1+2] + Int_[1+3];
  --! END [1]
  Result.Passed( "T030309", 100 );
END T030309;
```

Source File: T030310.TST

```
-- T030310
--
-- identifiers in a declarative part = 1024
--
-- Method:
--
-- Same as test T030309. If that test passes, this one does. It is
-- essentially the same requirement since you cannot declare an object
-- without introducing a new identifier.
--
--* COMPILE T030310
--* EXECUTE T030310
WITH Result;
PROCEDURE T030310 IS
BEGIN
    Result.Equivalent( "T030310", "T030309" );
END T030310;
```

Source File: T030311.TST

```
-- T030311
--
-- frames an exception can propagate through = unlimited
--
-- Method:
--
-- Recursively call a procedure until a storage error occurs. When it
-- does, raise a user defined exception which does NOT get handled until
-- the top level. If "Test_Exception Handled" gets printed, the exception
-- has been propagated correctly. This means that the exception has been
-- propagated through as many frames as possible until running out of
-- storage. The compiler shall be determined to have passed this
-- requirement if the compilation succeeds without error, and when
-- executed, "Test_Exception Handled" gets printed.
--
--* COMPILE T030311
--* EXECUTE T030311
WITH Result;
PROCEDURE T030311 IS

    Test_Exception : EXCEPTION;

    PROCEDURE Sub_Test IS
    BEGIN
        Sub_Test;
    EXCEPTION
        WHEN Storage_Error =>
            Result.Print( "Storage_Error Raised" );
            RAISE Test_Exception;
    END Sub_Test;

BEGIN
    Sub_Test;
EXCEPTION
    WHEN Test_Exception =>
        Result.Print( "Test_Exception Handled" );
        Result.Passed( "T030311", 100 );
    WHEN OTHERS =>
        Result.Print( "Test_Exception NOT Handled" );
        Result.Passed( "T030311", 0 );
END T030311;
--* NEW_LIBRARY
```

Source File: T030401.TST

```
-- T030401
--
-- values in subtype System.Priority = 16
--
-- Method:
--
-- Compile and execute a procedure to print out the range of values in
-- System.Priority. The compiler shall be determined to have passed this
-- requirement if the compilation succeeds without error and when executed,
-- there are at least 16 values in the range of System.Priority.
--
--* COMPILE T030401
--* EXECUTE T030401
WITH System;
WITH Result;
PROCEDURE T030401 IS
    First : Natural := System.Priority'FIRST;
    Last  : Natural := System.Priority'LAST;
    Size  : Natural := Last - First + 1;
BEGIN
    Result.Print( "Values in System.Priority:" &
                  Result.Image( Size, 4 ) & " : " &
                  Result.Image( First, 4 ) & " .. " &
                  Result.Image( Last, 4 ) & " ." );
    IF Size < 16 THEN
        Result.Passed( "T030401", Size * 100 / 16 );
    ELSE
        Result.Passed( "T030401", 100 );
    END IF;
END T030401;
```

Source File: T030402.TST

```
-- T030402
--
-- simultaneously active tasks in a program = 512
--
-- Method:
--
-- Declare a task type with one simple entry. The body of the task
-- consists of a single accept statement. Compile and execute a
-- procedure with 512 tasks of this type declared. In the body of the
-- procedure, 512 task entry calls are made. The compiler shall be
-- determined to have passed this requirement if the compilation and
-- execution succeeds without error.
--
--* COMPILE T030402
--* EXECUTE T030402
--! EQUATE Iter IS 512
WITH Result;
PROCEDURE T030402 IS
    TASK TYPE Task_Type IS
        ENTRY Hello;
    END Task_Type;

    --! LOOP Iter [1]
    Task_[1] : Task_Type;
    --! END [1]

    TASK BODY Task_Type IS
    BEGIN
        ACCEPT Hello;
    END Task_Type;
BEGIN
    --! LOOP Iter [1]
    Task_[1].Hello;
    --! END [1]
    Result.Passed( "T030402", 100 );
END T030402;
```


Source File: T030403.TST

```
-- T030403
--
-- accept statements in a task = 64
--
-- Method:
--
-- Declare a task type with one simple entry. The body of the task
-- consists of 64 accept statement. Compile and execute a procedure
-- with 64 task entry calls made. The compiler shall be determined to
-- have passed this requirement if the compilation and execution succeeds
-- without error.
--
--* COMPILE T030403
--* EXECUTE T030403
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030403 IS

    TASK TYPE Task_Type IS
        ENTRY Hello;
    END Task_Type;

    The_Task : Task_Type;

    TASK BODY Task_Type IS
    BEGIN
        --! LOOP Iter [1]
            ACCEPT Hello; -- [1]
        --! END [1]
    END Task_Type;

BEGIN
    --! LOOP Iter [1]
        The_Task.Hello; -- [1]
    --! END [1]
    Result.Passed( "T030403", 100 );
END T030403;
```

Source File: T030404.TST

```
-- T030404
--
-- entry declarations in a task = 64
--
-- Method:
--
-- Declare a task type with 64 entries. The body of the task consists
-- of 64 accept statements. Compile and execute a procedure with 64 task
-- entry calls made. This test will fail if T030403 fails. The compiler
-- shall be determined to have passed this requirement if the compilation
-- and execution succeeds without error.
--
--* COMPILE T030404
--* EXECUTE T030404
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030404 IS
    TASK TYPE Task_Type IS
        --! LOOP Iter [1]
        ENTRY Hello_[1];
        --! END [1]
    END Task_Type;

    The_Task : Task_Type;

    TASK BODY Task_Type IS
    BEGIN
        --! LOOP Iter [1]
        ACCEPT Hello_[1];
        --! END [1]
    END Task_Type;

BEGIN
    --! LOOP Iter [1]
    The_Task.Hello_[1];
    --! END [1]
    Result.Passed( "T030404", 100 );
END T030404;
```

Source File: T030405.TST

```
-- T030405
--
-- formal parameters in an entry declaration = 64
--
-- Method:
--
-- Declare a task type with an entry with 64 formal parameters. Compile
-- and execute a procedure with 1 task entry call made. The compiler
-- shall be determined to have passed this requirement if the compilation
-- and execution succeeds without error.
--
--* COMPILE T030405
--* EXECUTE T030405
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030405 IS
    TASK TYPE Task_Type IS
        ENTRY Hello(
            --! LOOP Iter-1 [1]
            Parm_1] : IN Integer;
            --! END [1]
            --! START Iter LOOP 1 [1]
            Parm_1] : IN integer );
            --! END [1]
        END Task_Type;

    The_Task : Task_Type;

    TASK BODY Task_Type IS
    BEGIN
        ACCEPT Hello(
            --! LOOP Iter-1 [1]
            Parm_1] : IN Integer;
            --! END [1]
            --! START Iter LOOP 1 [1]
            Parm_1] : IN Integer );
            --! END [1]
        END Task_Type;

    BEGIN
        The_Task.Hello(
            --! LOOP Iter-1 [1]
            [1],
            --! END [1]
            --! START Iter LOOP 1 [1]
            [1] );
            --! END [1]
        Result.Passed( "T030405", 100 );
    END T030405;
```

Source File: T030406.TST

```
-- T030406
--
-- formal parameters in an accept statement = 64
--
-- Method:
--
-- This test passes if T030405 does. It is impossible to test T030405
-- without using all 64 formal parameters in an accept statement.
--
--* COMPILE T030406
--* EXECUTE T030406
WITH Result;
PROCEDURE T030406 IS
BEGIN
    Result.Equivalent( "T030406", "T030405" );
END T030406;
```

Source File: T030407.TST

```
-- T030407
--
-- delay statements in a task = 64
--
-- Method:
--
-- Declare a task type containing 64 delay statements. Compile and execute
-- a procedure with 1 task entry call made. The compiler shall be
-- determined to have passed this requirement if the compilation and
-- execution succeeds without error.
--
--* COMPILE T030407
--* EXECUTE T030407
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030407 IS

    TASK TYPE Task_Type IS
        ENTRY Hello;
    END Task_Type;

    The_Task : Task_Type;

    TASK BODY Task_Type IS
    BEGIN
        ACCEPT Hello;
        --! LOOP Iter [1]
        DELAY 0.1; -- [1]
        --! END [1]
    END Task_Type;

BEGIN
    The_Task.Hello;
    Result.Passed( "T030407", 100 );
END T030407;
```

Source File: T030408.TST

```
-- T030408
--
-- alternatives in a select statement = 64
--
-- Method:
--
-- Declare a task type containing 1 entry with a select statement
-- containing 64 alternatives, all of which are the same entry. Compile
-- and execute a procedure with 1 task entry call made. The compiler
-- shall be determined to have passed this requirement if the compilation
-- and execution succeeds without error.
--
--* COMPILE T030408
--* EXECUTE T030408
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030408 IS

    TASK TYPE Task_Type IS
        ENTRY Hello;
    END Task_Type;

    The_Task : Task_Type;

    TASK BODY Task_Type IS
    BEGIN
        SELECT
            --! LOOP Iter-1 [1]
            ACCEPT Hello; -- [1]
        OR
            --! END [1]
            --! START Iter LOOP 1 [1]
            ACCEPT Hello; -- [1]
            --! END [1]
        END SELECT;
    END Task_Type;

BEGIN
    The_Task.Hello;
    Result.Passed( "T030408", 100 );
END T030408;
```

Source File: T030501.TST

```
-- T030501
--
-- formal parameters = 64
--
-- Method:
--
-- Declare and execute a procedure with 64 formal parameters. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030501
--* EXECUTE T030501
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030501 IS
    I : Integer := 0;

    PROCEDURE Hello(
        --! LOOP Iter-1 [1]
        Parm_1] : IN Integer;
        --! END [1]
        --! START Iter LOOP 1 [1]
        Parm_1] : IN Integer ) IS
        --! END [1]
    BEGIN
        --! LOOP Iter [1]
        I := I + Parm_1];
        --! END [1]
    END Hello;

BEGIN
    Hello(
        --! LOOP Iter-1 [1]
        1,
        --! END [1]
        1 );
    Result.Passed( "T030501", 100 );
END T030501;
```

Source File: T030502.TST

```
-- T030502
--
-- levels in a call chain = unlimited
--
-- Method:
--
-- Recursively call a procedure until a storage error occurs. When it
-- does, handle the exception and continue. If, after handling the
-- exception, control returns correctly to the top level, then we can
-- determine that the number of levels in a call chain is unlimited
-- since some other error occurs before a "levels in call chain exceeded"
-- type of error occurs. The compiler shall be determined to have passed
-- this requirement if the compilation and execution succeeds without
-- error.
--
--* COMPILE T030502
--* EXECUTE T030502
WITH Result;
PROCEDURE T030502 IS

    PROCEDURE Sub_Test IS
    BEGIN
        Sub_Test;
    EXCEPTION
        WHEN Storage_Error => NULL;
    END Sub_Test;

BEGIN
    Sub_Test;
    Result.Passed( "T030502", 100 );
END T030502;
```


Source File: T030601.TST

```
-- T030601
--
-- visible declarations = 1024
--
-- Method:
--
-- Compile a package containing 1024 procedure declarations. The compiler
-- shall be determined to have passed this requirement if the compilation
-- succeeds without error.
--
--* COMPILE T030601
--* EXECUTE T030601
--! EQUATE Iter IS 1024
PACKAGE Test_Package IS
  --! LOOP Iter [1]
  PROCEDURE Proc_[1];
  --! END [1]
END Test_Package;

PACKAGE BODY Test_Package IS
  Save : Natural := 0;
  --! LOOP Iter [1]
  PROCEDURE Proc_[1] IS
  BEGIN
    Save := [1];
  END Proc_[1];
  --! END [1]
END Test_Package;

WITH Result;
WITH Test_Package;
PROCEDURE T030601 IS
BEGIN
  Test_Package.Proc_1;
  Result.Passed( "T030601", 100 );
END T030601;
```

Source File: T030602.TST

```
-- T030602
--
-- private declarations = 1024
--
-- Method:
--
-- Compile a package containing 1024 private procedure declarations.
-- The compiler shall be determined to have passed this requirement
-- if the compilation succeeds without error.
--
--* COMPILE T030602
--* EXECUTE T030602
--! EQUATE Iter IS 1024
PACKAGE Test_Package IS
PRIVATE
  --! LOOP Iter [1]
  PROCEDURE Proc_[1];
  --! END [1]
END Test_Package;

PACKAGE BODY Test_Package IS
  Save : Natural := 0;
  --! LOOP Iter [1]
  PROCEDURE Proc_[1] IS
  BEGIN
    Save := [1];
  END Proc_[1];
  --! END [1]
END Test_Package;

WITH Result;
WITH Test_Package;
PROCEDURE T030602 IS
BEGIN
  Result.Passed( "T030602", 100 );
END T030602;
```

Source File: T030701.TST

```
-- T030701
--
-- declarations in a block = 1024
--
-- Method:
--
-- Compile a procedure containing a block with 1024 Integer declarations.
-- The compiler shall be determined to have passed this requirement if
-- the compilation succeeds without error.
--
--* COMPILE T030701
--* EXECUTE T030701
--! EQUATE Iter IS 1024
WITH Result;
PROCEDURE T030701 IS
BEGIN
  DECLARE
    --! LOOP Iter [1]
    Int_[1] : Integer;
    --! END [1]
  BEGIN
    --! LOOP Iter [1]
    Int_[1] := [1];
    --! END [1]
  END;
  Result.Passed( "T030701", 100 );
END T030701;
```

Source File: T030702.TST

```
-- T030702
--
-- enumeration literals in a single type = 512
--
-- Method:
--
-- Compile a procedure containing an enumeration type with 512 literals.
-- The compiler shall be determined to have passed this requirement if
-- the compilation and execution succeeds without error.
--
--* COMPILE T030702
--* EXECUTE T030702
--! EQUATE Iter IS 512
WITH Result;
PROCEDURE T030702 IS
    TYPE Enum IS (
        --! LOOP Iter-1 [1]
        Enum_ [1],
        --! END [1]
        --! START Iter LOOP 1 [1]
        Enum_ [1] );
        --! END [1]
    Var : Enum;
BEGIN
    --! LOOP Iter [1]
    Var := Enum_ [1];
    --! END [1]
    Result.Passed( "T030702", 100 );
END T030702;
```

Source File: T030703.TST

```
-- T030703
--
-- dimensions in an array = 32
--
-- Method:
--
-- Compile a procedure containing an array type with 32 dimensions. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030703
--* EXECUTE T030703
--! EQUATE Iter IS 32
WITH Result;
PROCEDURE T030703 IS
  TYPE Array_Type IS ARRAY(
    --! LOOP Iter-1 [1]
    1 .. 1, -- [1]
    --! END [1]
    --! START Iter LOOP 1 [1]
    1 .. 1 ) -- [1]
    --! END [1]
  OF Boolean;
  Var : Array_Type;
BEGIN
  Var :=
    --! LOOP Iter [1]
    ( 1 .. 1 =>
      --! END [1]
      True
    --! LOOP Iter [1]
    )
    --! END [1]
  ;
  Result.Passed( "T030703", 100 );
END T030703;
```

Source File: T030704.TST

```
-- T030704
--
-- total elements in an array = 65535
--
-- Method:
--
-- Compile a procedure containing an array with 65535 elements. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030704
--* EXECUTE T030704
WITH Result;
PROCEDURE T030704 IS
  TYPE Array_Type IS ARRAY( 1 .. 65535 ) OF Boolean;
  Var : Array_Type;
BEGIN
  Var := ( OTHERS => True );
  Result.Passed( "T030704", 100 );
END T030704;
```

Source File: T030705.TST

```
-- T030705
--
-- components in a record type = 256
--
-- Method:
--
-- Compile a procedure containing a record with 256 components. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030705
--* EXECUTE T030705
--! EQUATE Iter IS 256
WITH Result;
PROCEDURE T030705 IS
  TYPE Record_Type IS RECORD
    --! LOOP Iter [1]
    Comp [1] : Integer;
    --! END [1]
  END RECORD;
  Var : Record_Type;
BEGIN
  --! LOOP Iter [1]
  Var.Comp [1] := [1];
  --! END [1]
  Result.Passed( "T030705", 100 );
END T030705;
```

Source File: T030706.TST

```
-- T030706
--
-- discriminants in a record type = 64
--
-- Method:
--
-- Compile a procedure containing a record with 64 discriminants. The
-- compiler shall be determined to have passed this requirement if the
-- compilation and execution succeeds without error.
--
--* COMPILE T030706
--* EXECUTE T030706
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030706 IS
  TYPE Record_Type(
    --! LOOP Iter-1 [1]
    Disc [1] : Integer := 1;
    --! END [1]
    --! START Iter LOOP 1 [1]
    Disc [1] : Integer := 1 ) IS RECORD
    --! END [1]
    CASE Disc_1 IS
      WHEN 1 =>      Comp_1 : Integer;
      WHEN OTHERS => Comp_2 : Boolean;
    END CASE;
  END RECORD;
  Var_1 : Record_Type;
  Var_2 : Record_Type(
    --! LOOP Iter-1 [1]
    Disc [1] => 4,
    --! END [1]
    --! START Iter LOOP 1 [1]
    Disc [1] => 4 );
    --! END [1]
BEGIN
  Var_1.Comp_1 := 1;
  Var_2.Comp_2 := True;
  Result.Passed( "T030706", 100 );
END T030706;
```


Source File: T030707.TST

```
-- T030707
--
-- variant parts in a record type = 64
--
-- Method:
--
-- Compile a procedure containing a record with 64 discriminants and 64
-- variant parts (i.e. 64 nested case statements). If T030706 fails then
-- T030707 will fail. The compiler shall be determined to have passed
-- this requirement if the compilation and execution succeeds without
-- error.
--
--* COMPILE T030707
--* EXECUTE T030707
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030707 IS
  TYPE Record_Type(
    --! LOOP Iter-1 [1]
    Disc [1] : Integer := 1;
    --! END [1]
    --! START Iter LOOP 1 [1]
    Disc [1] : Integer := 1 ) IS RECORD
    --! END [1]
    --! LOOP Iter-1 [1]
    CASE Disc [1] IS
      WHEN OTHERS =>
        --! END [1]
        --! START Iter LOOP 1 [1]
        CASE Disc [1] IS
          --! END [1]
          WHEN 1      => Comp_1 : Integer;
          WHEN OTHERS => Comp_2 : Boolean;
          --! LOOP Iter [1]
        END CASE; -- [1]
        --! END [1]
    END RECORD;
  Var_1 : Record_Type;
  Var_2 : Record_Type(
    --! LOOP Iter-1 [1]
    Disc [1] => 4,
    --! END [1]
    --! START Iter LOOP 1 [1]
    Disc [1] => 4 );
    --! END [1]
BEGIN
  Var_1.Comp_1 := 1;
  Var_2.Comp_2 := True;
  Result.Passed( "T030707", 100 );
END T030707;
```

Source File: T030708.TST

```
-- T030708
--
-- size of any object in bits = 65535
--
-- Method:
--
-- Declare a Record with component String of size 65535/(word size).
-- The compiler shall be determined to have passed this requirement if
-- the compilation and execution succeeds without error and the printed
-- object size is greater than or equal to 65535.
--
--* COMPILE T030708
--* EXECUTE T030708
WITH Result;
PROCEDURE T030708 IS

    Test_Char : Character := ' ';

    TYPE Large_Type IS RECORD
        Comp : String( 1 .. 65536 / Test_Char'SIZE );
    END RECORD;

    Var : Large_Type;
    Size : Natural;

BEGIN
    Var.Comp := ( OTHERS => ' ' );
    Size := Integer( Var'SIZE );
    Result.Print( "Size of Object: " & Result.Image( Size ) );
    IF Size >= 65535 THEN
        Result.Passed( "T030708", 100 );
    ELSE
        Result.Inconclusive( "T030708" );
    END IF;
END T030708;
```

Source File: T030709.TST

```
-- T030709
--
-- characters in a value of type STRING = 65535
--
-- Method:
--
-- Declare a variable of type STRING( 1 .. 65535 ) and assign all of the
-- elements in the string to some value. The compiler shall be determined
-- to have passed this requirement if the compilation and execution
-- succeeds without error.
--
--* COMPILE T030709
--* EXECUTE T030709
WITH Result;
PROCEDURE T030709 IS
  Var : String( 1 .. 65535 );
BEGIN
  Var := ( OTHERS => 'A' );
  Result.Passed( "T030709", 100 );
END T030709;
```

Source File: T030801.TST

```
-- T030801
--
-- operators in an expression = 128
--
-- Method:
--
-- Place 128 +'s in an assignment statement. The compiler shall be
-- determined to have passed this requirement if the compilation and
-- execution succeeds without error.
--
--* COMPILE T030801
--* EXECUTE T030801
--! EQUATE Iter IS 128
WITH Result;
PROCEDURE T030801 IS
  Var : Integer;
BEGIN
  Var := 2;
  Var :=
    --! LOOP Iter-1 [1]
    Var + -- [1]
    --! END [1]
    --! START Iter LOOP 1 [1]
    Var; -- [1]
    --! END [1]
  Result.Passed( "T030801", 100 );
END T030801;
```

Source File: T030802.TST

```
-- T030802
--
-- function calls in an expression = 128
--
-- Method:
--
-- Place 128 function calls added together in an assignment statement.
-- The compiler shall be determined to have passed this requirement if
-- the compilation and execution succeeds without error.
--
--* COMPILE T030802
--* EXECUTE T030802
--! EQUATE Iter IS 128
WITH Result;
PROCEDURE T030802 IS
    Var : Integer;

    FUNCTION Func RETURN Integer IS
    BEGIN
        RETURN 2;
    END Func;
BEGIN
    Var := 2;
    Var :=
        --! LOOP Iter-1 [1]
        Func + -- [1]
        --! END [1]
        --! START Iter LOOP 1 [1]
        Func; -- [1]
        --! END [1]
    Result.Passed( "T030802", 100 );
END T030802;
```

Source File: T030803.TST

```
-- T030803
--
-- primaries in an expression = 128
--
-- Method:
--
-- Place 128 distinct numeric literals added together in an assignment
-- statement. The compiler shall be determined to have passed this
-- requirement if the compilation and execution succeeds without error.
--
--* COMPILE T030803
--* EXECUTE T030803
--! EQUATE Iter IS 128
WITH Result;
PROCEDURE T030803 IS
  Var : Integer;
BEGIN
  Var :=
    --! LOOP Iter-1 [1]
    [1] +
    --! END [1]
    --! START Iter LOOP 1 [1]
    [1];
    --! END [1]
  Result.Passed( "T030803", 100 );
END T030803;
```

Source File: T030804.TST

```
-- T030804
--
-- depth of parentheses nesting = 64
--
-- Method:
--
-- Place an addition inside 64 layers of parentheses. The compiler shall
-- be determined to have passed this requirement if the compilation and
-- execution succeeds without error.
--
--* COMPILE T030804
--* EXECUTE T030804
--! EQUATE Iter IS 64
WITH Result;
PROCEDURE T030804 IS
  Var : Integer;
BEGIN
  Var := 2;
  Var :=
    --! LOOP Iter [1]
    ( -- [1]
    --! END [1]
    Var + Var
    --! LOOP Iter [1]
    ) -- [1]
    --! END [1]
  ;
  Result.Passed( "T030804", 100 );
END T030804;
--* NEW_LIBRARY
```

Source File: T040101.TST

```
-- T040101
--
-- The compiler shall be invokable from either a batch file command or an
-- interactive command.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040101
--* EXECUTE T040101
WITH Result;
PROCEDURE T040101 IS
BEGIN
    Result.Manual_Test( "T040101" );
END T040101;
```


Source File: T040102.TST

-- T040102

--
-- The compiler shall be sharable (re-entrant) by multiple users, if the
-- host operating system supports multiple users.

-- Method:

-- Inspection.

--* COMPILE T040102

--* EXECUTE T040102

WITH Result;

PROCEDURE T040102 IS

BEGIN

 Result.Manual_Test("T040102");

END T040102;

Source File: T040103.TST

```
-- T040103
--
-- The compiler shall implement options to perform the same function as
-- PRAGMAS Suppress and Optimize.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040103
--* EXECUTE T040103
WITH Result;
PROCEDURE T040103 IS
BEGIN
    Result.Manual_Test( "T040103" );
END T040103;
```

Source File: T040104.TST

```
-- T040104
--
-- The compiler shall implement an option to recover from non-fatal errors
-- as defined in 4.3.3. The recovery action taken shall be identified.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040104
--* EXECUTE T040104
WITH Result;
PROCEDURE T040104 IS
BEGIN
    Result.Manual_Test( "T040104" );
END T040104;
```

Source File: T040105.TST

```
-- T040105
--
-- The compiler shall implement an option to disable the generation of
-- diagnostic messages of a specified severity level.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040105
--* EXECUTE T040105
WITH Result;
PROCEDURE T040105 IS
BEGIN
    Result.Manual_Test( "T040105" );
END T040105;
```

Source File: T040106.TST

-- T040106

--

-- The compiler shall implement an option to select or suspend the generation
-- of object code and/or assembly code.

--

-- Method:

--

-- Inspection.

--

--* COMPILE T040106

--* EXECUTE T040106

WITH Result;

PROCEDURE T040106 IS

BEGIN

 Result.Manual_Test("T040106");

END T040106;

Source File: T040201.TST

```
-- T040201
--
-- The compiler shall be able to produce at the option of the user a
-- compilation listing showing the source code with line numbers.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040201
--* EXECUTE T040201
WITH Result;
PROCEDURE T040201 IS
BEGIN
    Result.Manual_Test( "T040201" );
END T040201;
```

Source File: T040202.TST

-- T040202

--

-- The compiler shall be able to produce at the option of the user a list of
-- diagnostic messages either at the position in the source code where the
-- condition occurred, and/or at the end of the compilation listing, even if
-- the compilation terminates abnormally.

--

-- Method:

--

-- Inspection.

--

--* COMPILE T040202

--* EXECUTE T040202

WITH Result;

PROCEDURE T040202 IS

BEGIN

 Result.Manual_Test("T040202");

END T040202;

Source File: T040203.TST

```
-- T040203
--
-- The compiler shall be able to produce at the option of the user an assembly
-- or pseudo-assembly output listing.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040203
--* EXECUTE T040203
WITH Result;
PROCEDURE T040203 IS
BEGIN
    Result.Manual_Test( "T040203" );
END T040203;
```


Source File: T040204.TST

-- T040204

--

-- The compiler shall be able to produce at the option of the user an assembly
-- or pseudo-assembly output listing with embedded Ada source statements
-- adjacent to the assembly code they generated.

--

-- Method:

--

-- Inspection.

--

--* COMPILE T040204

--* EXECUTE T040204

WITH Result;

PROCEDURE T040204 IS

BEGIN

 Result.Manual_Test("T040204");

END T040204;

Source File: T040205.TST

```
-- T040205
--
-- The compiler shall be able to produce at the option of the user a cross
-- reference (set/use) listing.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040205
--* EXECUTE T040205
WITH Result;
PROCEDURE T040205 IS
BEGIN
    Result.Manual_Test( "T040205" );
END T040205;
```

Source File: T040206.TST

```
-- T040206
--
-- The compiler shall be able to produce at the option of the user a map of
-- relative addresses of variables and constants.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T040206
--* EXECUTE T040206
WITH Result;
PROCEDURE T040206 IS
BEGIN
    Result.Manual_Test( "T040206" );
END T040206;
```

Source File: T040207.TST

-- T040207

--
-- For each compilation, the compiler shall be able to produce at the option
-- of the users a statistics summary listing with the following information:

--
-- a. Number of statements
-- b. Number of source lines
-- c. Compile time per program module (CPU time)
-- d. Total compile time (CPU and elapsed time)
-- e. Total number of instructions generated
-- f. Total number of data words generated
-- g. Total size of object module generated
--

-- Method:

-- Inspect the compiler listing generated by this test and complete questions.

--* COMPILE T040207 STATISTICS COMPILER_LISTING

--* EXECUTE T040207

WITH Result;

PROCEDURE T040207 IS

BEGIN

 --! LOOP 70 [1]

 -- Enough lines to cause a page break.

 --! END [1]

 Result.Print("This is a test.");

 Result.Manual_Test("T040207");

END T040207;

Source File: T040208.TST

```
-- T040208
--
-- All listings shall include the following header information on every page:
--
--     a. Date and time of compilation
--     b. Compilation unit name
--     c. Type of listing
--     d. Page number within total listing
--     e. User identification
--
-- Method:
--
-- Inspect the compiler listing generated by this test.
--
--* COMPILE T040208 COMPILER_LISTING
--* EXECUTE T040208
WITH Result;
PROCEDURE T040208 IS
BEGIN
    --! LOOP 70 [1]
    -- Enough lines to cause a page break.
    --! END [1]
    Result.Print( "This is a test." );
    Result.Manual_Test( "T040208" );
END T040208;
```

Source File: T040209.TST

```
-- T040209
--
-- All listings shall have the following additional information within the
-- listing:
--
--     a. Compiler name, version number, release date
--     b. Host and target computer configurations
--     c. Specified and default control options
--     d. Source file name
--     e. Object file name
--
-- Method:
--
-- Inspect the compiler listing generated by the previous test (T040208).
--
--* COMPILE T040209
--* EXECUTE T040209
WITH Result;
PROCEDURE T040209 IS
BEGIN
    Result.Manual_Test( "T040209" );
END T040209;
```

Source File: T040301.TST

```
-- T040301
--
-- Each diagnostic message shall contain the messages text, a reference
-- number for additional information in the compiler documentation, and
-- a severity level.
--
-- Method:
--
-- If the first compiler message for the following code contains the
-- necessary information the test has passed.
--
--* COMPILE T040301
--* EXECUTE T040301
WITH Result;
PROCEDURE T040301 IS
BEGIN
    Result.Manual_Test( "T040301" );
END T040301;
--* COMPILE TEST
PROCEDURE Test IS
    TYPE Bad_Type IS String( 1 .. 2 ); -- bad declaration
    Var : Bad_Type;
BEGIN
    Var := 'A'; -- bad assignment
END Test;
```

Source File: T040302.TST

```
-- T040302
--
-- The diagnostic message text shall be sufficiently informative to enable the
-- user to analyze the problem without consulting compiler documentation.
--
-- Method:
--
-- If the compiler message for the incorrect assignment statement
-- informs the user of the type conflict, and informs the user of
-- the type of "var" and the type of "A" the test has passed.
--
--* COMPILE T040302
--* EXECUTE T040302
WITH Result;
PROCEDURE T040302 IS
BEGIN
    Result.Manual_Test( "T040302" );
END T040302;
--* COMPILE TEST
PROCEDURE Test IS
    SUBTYPE Bad_Type IS String( 1 .. 2 );
    Var : Bad_Type;
BEGIN
    Var := 'A';
END Test;
-- bad assignment
```


Source File: T040303.TST

-- T040303

-- The severity levels of diagnostic messages shall include the following
-- error classes:

- a. Note: Information to the user; the compilation process continues
-- and the object program is not affected.
- b. Warning: Information about the validity of the program. The source
-- program is well-defined and semantically correct; the object
-- program may not behave as intended.
- c. Error: An illegal syntactic or semantic construct with a well-
-- defined recovery action. Compilation continues and the object
-- program contains code for the illegal construct; the object
-- program may behave meaninglessly at run-time.
- d. Serious Error: Illegal construct with no well-defined recovery
-- action. Syntax analysis continues but no object program is
-- generated.
- e. Fatal Error: Illegal construct with no reasonable syntactic
-- recovery action. Compilation terminates and no outputs other
-- than the source listing and diagnostic messages are produced.

-- Method:

-- Inspection of documentation. It would be difficult to construct
-- code fragments that would clearly belong to each of these classes of
-- messages for each compiler.

--* COMPILE T040303

--* EXECUTE T040303

WITH Result;

PROCEDURE T040303 IS

BEGIN

 Result.Manual_Test("T040303");
END T040303;

Source File: T040304.TST

```
-- T040304
--
-- The compiler shall issue a diagnostic message to indicate any capacity
-- requirements that have been exceeded.
--
-- Method:
--
-- Compile a program containing an array with 1000 dimensions. If
-- the compiler compiles this without error, increase this number
-- until an error occurs. If the compiler issues an informative
-- error to the effect that the number of dimensions in the array
-- is too large, the compiler has passed the test.
--
--* COMPILE T040304
--* EXECUTE T040304
WITH Result;
PROCEDURE T040304 IS
BEGIN
    Result.Manual_Test( "T040304" );
END T040304;
--* COMPILE TEST
--* EXECUTE TEST
--! EQUATE Excess IS 1000
WITH Result;
PROCEDURE Test IS
    TYPE Big_Arr IS ARRAY(
        --! LOOP Excess [1]
        1 .. 2,
        --! END [1]
        1 .. 2 ) OF Integer;

    Big1, Big2 : Big_Arr;
BEGIN
    Big1 := Big2;
    Result.Inconclusive( "T040304" );
END Test;
```

```
-- T040305
--
-- The compiler shall not abort regardless of the type or number of errors
-- encountered.
--
-- Method:
--
-- Inspect the documentation to make sure that the compiler can be set
-- to continue compilation regardless of the number of errors encountered.
-- If the compiler flaggs all the lines indicated as errors in the code
-- below, and the first part of this method is satisfied, the compiler
-- has passed the test.
--
--* COMPILE T040305
--* EXECUTE T040305
WITH Result;
PROCEDURE T040305 IS
BEGIN
    Result.Manual_Test( "T040305" );
END T040305;
--* COMPILE TEST
PROCEDURE Test IS

    Typeeee Junk IS RANGE 0 .. 100; -- should be flagged
    TYPE Ok IS RANGE 0 .. 100;      -- should not be flagged
    Var : Ok := 'a';                -- should be flagged

BEGIN
    Vr := 10;                        -- should be flagged
    Var := 10;                       -- should not be flagged
    Var := "abc";                    -- should be flagged
    last_line;                       -- last line should be flagged
END Test;
```

Source File: T050101.TST

```
-- T050101
--
-- The compiler and/or external tool shall be able to produce a source listing
-- with indentations to show control constructs.
--
-- Method:
--
-- Inspection of documentation. This test cannot be automated since
-- the requirement allows for an external tool to perform the function.
--
--* COMPILE T050101
--* EXECUTE T050101
WITH Result;
PROCEDURE T050101 IS
BEGIN
    Result.Manual_Test( "T050101" );
END T050101;
```

Source File: T050102.TST

```
-- T050102
--
-- The compiler, linker/loader, and/or external tool shall be able to produce
-- an absolute assembly code listing.
--
-- Method:
--
-- Inspection of documentation. This test cannot be automated since
-- the requirement allows for an external tool to perform the function.
--
--* COMPILE T050102
--* EXECUTE T050102
WITH Result;
PROCEDURE T050102 IS
BEGIN
    Result.Manual_Test( "T050102" );
END T050102;
```

Source File: T050103.TST

```
-- T050103
--
-- The compiler and/or library manager shall be able to produce at the option
-- of the user a dependency listing showing which library units are WITHed by
-- other units.
--
-- Method:
--
-- Inspection of documentation. This test cannot be automated since
-- the requirement allows for an external tool to perform the function.
--
--* COMPILE T050103
--* EXECUTE T050103
WITH Result;
PROCEDURE T050103 IS
BEGIN
    Result.Manual_Test( "T050103" );
END T050103;
```

Source File: T050104.TST

```
-- T050104
--
-- The compiler and/or library manager shall have the capability of listing
-- all out-of-date (obsolete) library units with the option of selectively
-- recompiling such units before linking.
--
-- Method:
--
-- Inspection of documentation. This test cannot be automated since
-- the requirement allows for an external tool to perform the function.
--
--* COMPILE T050104
--* EXECUTE T050104
WITH Result;
PROCEDURE T050104 IS
BEGIN
    Result.Manual_Test( "T050104" );
END T050104;
```

Source File: T050201.TST

```
-- T050201
--
-- The compiler and/or linker/loader shall include in the load module only
-- those subprograms that are actually referenced by the object program.
--
-- Method:
--
-- Compile and execute three procedures containing:
--
--     (1) a reference to 1 subprogram from a package with 1 subprogram,
--     (2) a reference to 1 subprogram from a package with 25 subprograms,
--     (3) references to 25 subprograms from a package with 25 subprograms.
--
-- The compiler has passed the test if size (2) is closer to size (1)
-- then it it to size (3).
--
--* COMPILE T050201 OPTIMIZE_SPACE
--* EXECUTE Size_1
--* EXECUTE Size_2
--* EXECUTE Size_3
--* EXECUTE T050201
--! EQUATE Repeat IS 25
PACKAGE Share IS
    TYPE List IS ARRAY( 1 .. 1000 ) OF Integer;
    TYPE Pair IS RECORD
        Head : List := ( OTHERS => 20 );
        Tail : List := ( OTHERS => 30 );
    END RECORD;
END Share;

PACKAGE Code_A IS
    PROCEDURE Do_1( Item : IN OUT Integer );
END Code_A;

WITH Share;
PACKAGE BODY Code_A IS
    PROCEDURE Do_1( Item : IN OUT Integer ) IS
        X, Y, Z : Share.Pair;
    BEGIN
        X.Head := ( OTHERS => Item );
        X.Tail := ( OTHERS => Item + 1 );
        Y := X; Z := Y; Item := Z.Tail( 1 );
    END Do_1;
END Code_A;

PACKAGE Code_B IS
    --! LOOP Repeat [1]
    PROCEDURE Do_1( Item : IN OUT Integer );
    --! END [1]
END Code_B;

WITH Share;
PACKAGE BODY Code_B IS
    --! LOOP Repeat [1]
    PROCEDURE Do_1( Item : IN OUT Integer ) IS
        X, Y, Z : Share.Pair;
    BEGIN
        X.Head := ( OTHERS => Item );
        X.Tail := ( OTHERS => Item + [1] );
        Y := X; Z := Y; Item := Z.Tail( [1] );
    END Do_1;
    --! END [1]
END Code_B;

WITH Code_A;
PROCEDURE Size_1 IS
    Item : Integer := 1;
BEGIN
    --! LOOP Repeat [1]
    Code_A.Do_1( Item ); -- [1]
    --! END [1]
```


Source File: T050201.TST

END Size_1;

```
WITH Code_B;
PROCEDURE Size_2 IS
    Item : Integer := 1;
BEGIN
    --! LOOP Repeat [1]
    Code_B.Do_1( Item ); -- [1]
    --! END [1]
END Size_2;
```

```
WITH Code_B;
PROCEDURE Size_3 IS
    Item : Integer := 1;
BEGIN
    --! LOOP Repeat [1]
    Code_B.Do_1( Item ); -- [1]
    --! END [1]
END Size_3;
```

```
WITH Result;
PROCEDURE T050201 IS
    Size_1 : Natural;
    Size_2 : Natural;
    Size_3 : Natural;
BEGIN
    Result.Print_Code_Size( "SIZE_1", Size_1 );
    Result.Print_Code_Size( "SIZE_2", Size_2 );
    Result.Print_Code_Size( "SIZE_3", Size_3 );
    IF Size_1 = Size_3 THEN
        Result.Inconclusive( "T050201", "Code sizes are the same." );
    ELSE
        Result.Passed( "T050201", Size_2 - Size_1 < Size_3 - Size_2 );
    END IF;
END T050201;
```

Source File: T050202.TST

```
-- T050202
--
-- The compiler and/or linker/loader shall include in the load module only
-- those run-time system modules that are referenced by the object program.
--
-- Method:
--
-- Compile and execute two procedures containing:
--
--     (1) a simple task,
--     (2) a simple subprogram.
--
-- The load module size of the procedure containing the task should
-- be larger due to the added size of the tasking run-time modules.
-- The compiler has passed the test if size (1) is larger than size (2).
--
--* COMPILER T050202 OPTIMIZE_SPACE
--* EXECUTE Size_1
--* EXECUTE Size_2
--* EXECUTE T050202
PROCEDURE Size_1 IS
    I : Integer;

    PROCEDURE Simple_Proc IS
    BEGIN
        I := 10;
    END Simple_Proc;

BEGIN
    Simple_Proc;
END Size_1;

PROCEDURE Size_2 IS
    I : Integer;

    TASK Simple_Task IS
        ENTRY START;
    END Simple_Task;

    TASK BODY Simple_Task IS
    BEGIN
        ACCEPT START;
        I := 10;
    END Simple_Task;

BEGIN
    Simple_Task.START;
END Size_2;

WITH Result;
PROCEDURE T050202 IS
    Size_1 : Natural;
    Size_2 : Natural;
BEGIN
    Result.Print_Code_Size( "SIZE_1", Size_1 );
    Result.Print_Code_Size( "SIZE_2", Size_2 );
    Result.Passed( "T050202", Size_1 > Size_2 );
END T050202;
```

Source File: T050203.TST

```
-- T050203
--
-- The compiler and/or linker/loader shall support the partial linking of
-- object modules as specified by the user.
--
-- Method:
--
-- Inspection of documentation. This test can not be automated since
-- the method of specifying the partial linking is compiler dependent.
--
--* COMPILE T050203
--* EXECUTE T050203
WITH Result;
PROCEDURE T050203 IS
BEGIN
    Result.Manual_Test( "T050203" );
END T050203;
```

Source File: T050204.TST

```
-- T050204
--
-- The compiler and/or linker/loader shall support the linking of designated
-- object modules without including them in the load module.
--
-- Method:
--
-- Inspection of documentation. This test can not be automated since
-- the method of specifying the designated linking is compiler dependent.
--
--* COMPILE T050204
--* EXECUTE T050204
WITH Result;
PROCEDURE T050204 IS
BEGIN
    Result.Manual_Test( "T050204" );
END T050204;
```

Source File: T050300.TST

```
-- T050300
--
-- The compiler shall be able to produce object code files and other types of
-- data necessary to debug those files with an available source-level(symbolic)
-- debugger.
--
-- Method:
--
-- Inspection of documentation.
--
--* COMPILE T050300
--* EXECUTE T050300
WITH Result;
PROCEDURE T050300 IS
BEGIN
    Result.Manual_Test( "T050300" );
END T050300;
--* NEW_LIBRARY
```

Source File: T060100.TST

```
-- T060100
--
-- The compiler shall eliminate statements or subprograms that will never be
-- excuted (dead code) because their execution depends on a condition known
-- to be false at compilation time.
--
-- Method:
--
-- Compile a procedure consisting of some code dependent on a Boolean
-- constant. If any of the dead code string literals containing x's
-- are present in the assembly listing, the compiler has failed the test.
-- Multiple lines are used to help pick out the code in the listing.
--
--* COMPILE T060100 ASSEMBLY_LISTING
--* EXECUTE T060100
WITH Result;
PROCEDURE T060100 IS

    Debug : CONSTANT Boolean := False;

    PROCEDURE Used IS
    BEGIN
        Result.Print( "Live Procedure: ..... " );
        Result.Print( "Live Procedure: ..... " );
        Result.Print( "Live Procedure: ..... " );
    END Used;

    PROCEDURE Unused IS
    BEGIN
        Result.Print( "Dead Procedure: ***** " );
        Result.Print( "Dead Procedure: ***** " );
        Result.Print( "Dead Procedure: ***** " );
    END Unused;

BEGIN
    Result.Print( "Live Statement: ..... " );
    Result.Print( "Live Statement: ..... " );
    Result.Print( "Live Statement: ..... " );
    IF Debug THEN
        Result.Print( "Dead Statement: ***** " );
        Result.Print( "Dead Statement: ***** " );
        Result.Print( "Dead Statement: ***** " );
    END IF;
    Result.Print( "Live Statement: ..... " );
    Result.Print( "Live Statement: ..... " );
    Result.Print( "Live Statement: ..... " );
    Used;

    Result.Manual_Test( "T060100" );
END T060100;
```

Source File: T060201.TST

```
-- T060201
--
-- The compiler shall allow the Ada program text to contain any of the 95
-- graphic characters and 5 form effectors of the ISO 7-bit character set
-- ( ISO Standard 646 ) to the extent supported by the host computer.
--
-- Method:
--
-- Compile a program containing these 100 characters in comments. Note:
-- The format effectors will not show up on hard copy. The compiler shall
-- be determined to have passed this test if the compilation proceeds
-- without error.
--
--* COMPILE T060201
--* EXECUTE T060201
WITH Result;
PROCEDURE T060201 IS
BEGIN
  -- Upper Case (26)
  -- ABCDEFGHIJKLMNOPQRSTUVWXYZ
  -- Lower Case (26)
  -- abcdefghijklmnopqrstuvwxyz
  -- Digits (10)
  -- 0123456789
  -- Special Characters (19)
  -- '&'()*+,-./:;<=>_!"
  -- Other Special Characters (13)
  -- !$%?'@[\]- {}
  -- Blank Space (01)
  -- ' '
  -- Form Effectors (05)
  -- horizontal tab ' '
  -- vertical tab '␣'
  -- carriage return '␣'
  -- line feed '␣'
  -- form feed '␣'
  Result.Passed( "T060201", True );
END T060201;
```

Source File: T060202.TST

```
-- T060202
--
-- The predefined packages TEXT_IO, DIRECT_IO, and SEQUENTIAL_IO shall support
-- input and output of data containing any of the 128 ASCII character literals
-- of the predefined type STANDARD.CHARACTER.
--
-- Method:
--
-- Using each of the three modes, write these characters to a file and
-- read them back in. The test will record its success or failure.
--
--* COMPILE T060202
--* EXECUTE T060202
WITH Result;
WITH Text_IO;
WITH Direct_IO;
WITH Sequential_IO;
PROCEDURE T060202 IS
```

```
PACKAGE Dir_IO IS NEW Direct_IO( Character );
PACKAGE Seq_IO IS NEW Sequential_IO( Character );
```

```
FUNCTION Test_Text_IO RETURN Boolean IS
    Success : Boolean := True;
    Char    : Character;
    File    : Text_IO.File_Type;
BEGIN
    Text_IO.Create( File, Text_IO.Out_File );
    FOR Counter IN ASCII.Nul .. ASCII.Del LOOP
        Text_IO.Put( File, Counter );
    END LOOP;
    Text_IO.Reset( File, Text_IO.In_File );
    FOR Counter IN ASCII.Nul .. ASCII.Del LOOP
        Text_IO.Get( File, Char );
        Success := Success AND ( Char = Counter );
    END LOOP;
    Text_IO.Delete( File );
    RETURN Success;
END Test_Text_IO;
```

```
FUNCTION Test_Direct_IO RETURN Boolean IS
    Success : Boolean := True;
    Char    : Character;
    File    : Dir_IO.File_Type;
BEGIN
    Dir_IO.Create( File, Dir_IO.Out_File );
    FOR Counter IN ASCII.Nul .. ASCII.Del LOOP
        Dir_IO.Write( File, Counter );
    END LOOP;
    Dir_IO.Reset( File, Dir_IO.In_File );
    FOR Counter IN ASCII.Nul .. ASCII.Del LOOP
        Dir_IO.Read( File, Char );
        Success := Success AND ( Char = Counter );
    END LOOP;
    Dir_IO.Delete( File );
    RETURN Success;
END Test_Direct_IO;
```

```
FUNCTION Test_Sequential_IO RETURN Boolean IS
    Success : Boolean := True;
    Char    : Character;
    File    : Seq_IO.File_Type;
BEGIN
    Seq_IO.Create( File, Seq_IO.Out_File );
    FOR Counter IN ASCII.Nul .. ASCII.Del LOOP
        Seq_IO.Write( File, Counter );
    END LOOP;
    Seq_IO.Reset( File, Seq_IO.In_File );
    FOR Counter IN ASCII.Nul .. ASCII.Del LOOP
        Seq_IO.Read( File, Char );
```


Source File: T060202.TST

```
        Success := Success AND ( Char = Counter );
    END LOOP;
    Seq_IO.Delete( File );
    RETURN Success;
END Test_Sequential_IO;

FUNCTION Value( Success : Boolean; Name : String ) RETURN Natural IS
BEGIN
    CASE Success IS
        WHEN True => Result.Print( Name & " PASSED" ); RETURN 100;
        WHEN False => Result.Print( Name & " FAILED" ); RETURN 0;
    END CASE;
END Value;

BEGIN
    Result.Passed
    ( "T060202",
      ( Value( Test_Text_IO,      "Text_IO"      ) +
        Value( Test_Direct_IO,    "Direct_IO"    ) +
        Value( Test_Sequential_IO, "Sequential_IO" ) ) / 3 );
END T060202;
```

Source File: T060203.TST

```
-- T060203
--
-- The compiler shall allow comments and values of the predefined type STRING
-- to contain any of the 128 ASCII characters contained in the predefined type
-- STANDARD.CHARACTER.
--
-- Method:
--
-- Compile a program containing these 128 characters assigned to a string
-- variable and in a comment. The compiler will have passed this
-- requirement if the compilation and execution proceed without error.
--
-- Note: This requirement does not apply to string LITERALS. LITERALS
-- are confined to the 95 graphic characters (LRM 2.6). The allowable
-- characters in a comment is tested in T060201, so that part of the
-- requirement is ignored by this test.
--
--* COMPILE T060203
--* EXECUTE T060203
WITH Result;
PROCEDURE T060203 IS
    USE ASCII;

    S1 : String( 1 .. 128 );
BEGIN
    S1 := Nul & Soh & Stx & Etx & Eot & Enq & Ack & Bel &
        Bs & Ht & Lf & Vt & Ff & Cr & So & Si &
        Dle & Dcl & Dc2 & Dc3 & Dc4 & Nak & Syn & Etb &
        Can & Em & Sub & Esc & Fs & Gs & Rs & Us &
        ' ' & '!' & '"' & '#' & '$' & '%' & '&' & "'" &
        "(" & ")" & "+" & "-" & "." & "/" &
        "01234567" &
        "89:;<=>?" &
        " @ABCDEFGHIJ" &
        "KLMNOPQRST" &
        "UVWXYZ[\]_`" &
        " abcdefgh" &
        "ijklmno" &
        "pqrstuvw" &
        "xyz{!}" & Del;
    Result.Passed( "T060203", True );
END T060203;
```

```
-- T060301
--
-- The compiler shall provide predefined types in package STANDARD for all the
-- integer and floating point types provided by the target computer.
--
-- Method:
--
-- This test is machine dependent. For each compiler, modify the
-- declaration of integer and float variables so that all of the compiler
-- supported types in package STANDARD are represented. This list of
-- supported types must be checked manually against the machine supported
-- types. If there are any machine types not represented here, the test
-- fails.
--
-- The following code must be modified for each implementation. Each
-- integer and floating-point type supported by the compiler should be
-- included here.
--* BEGIN
--
-- This comment is not visible in the test output. There are currently
-- nine types provided by this test, five integer and four float types.
-- For each compiler, add a section at the top commented out as specific
-- to that compiler in which all types supported by the compiler are
-- represented. For each type represented, add the compiler name to
-- the "--* BEGIN ... comp-name" section where the values are printed.
-- If another type other than the nine given here is needed, it may
-- be included by following the present format.
--
-- Note: Text between "--* BEGIN" and "--* END" (no compiler
-- is given in the BEGIN statement) is excluded from the test.
--* END
--
--* COMPILE T060301
--* EXECUTE T060301
WITH Result;
PROCEDURE T060301 IS

--* BEGIN Dec_Vax_V1_4
-- Big_Int should be set to the largest predefined Integer type
TYPE Big_Int IS NEW Integer;

Int_1 : Short_Short_Integer;
Int_2 : Short_Integer;
Int_3 : Integer;
-- Int_4 : Long_Integer;
-- Int_5 : Long_Long_Integer;

-- Flt_1 : Short_Float;
Flt_2 : Float;
Flt_3 : Long_Float;
Flt_4 : Long_Long_Float;
--* END
--* BEGIN TeleGen2_V3_15
-- Big_Int should be set to the largest predefined Integer type
TYPE Big_Int IS NEW Long_Integer;

-- Int_1 : Short_Short_Integer;
-- Int_2 : Short_Integer;
Int_3 : Integer;
Int_4 : Long_Integer;
-- Int_5 : Long_Long_Integer;

-- Flt_1 : Short_Float;
Flt_2 : Float;
Flt_3 : Long_Float;
-- Flt_4 : Long_Long_Float;
--* END

PROCEDURE Show( Line : String; Int : Big_Int ) IS

FUNCTION Format( Image : String ) RETURN String IS
Result : String( 1 .. 20 ) := ( OTHERS => ' ' );
BEGIN
```

```

        Result( Result'LAST - Image'LENGTH + 1 .. Result'LAST ) := Image;
        RETURN Result;
    END Format;

BEGIN
    Result.Print( Line & " => " & Format( Big_Int'IMAGE( Int ) ) );
END Show;

BEGIN
    Result.Print( "Check the numeric types supported by the hardware." );
    Result.Print( "If there are none missing as listed here the test passes." );
    Result.Print( "" );
    Result.Print( "" );

--* BEGIN Dec_Vax_V1_4
-- Short_Short_Integer
Show( "Short_Short_Integer'SIZE ", Big_Int( Short_Short_Integer'SIZE ) );
Show( "Short_Short_Integer'FIRST", Big_Int( Short_Short_Integer'FIRST ) );
Show( "Short_Short_Integer'LAST ", Big_Int( Short_Short_Integer'LAST ) );
Result.Print( "" );

--* END
--* BEGIN Dec_Vax_V1_4
-- Short_Integer
Show( "Short_Integer'SIZE ", Big_Int( Short_Integer'SIZE ) );
Show( "Short_Integer'FIRST ", Big_Int( Short_Integer'FIRST ) );
Show( "Short_Integer'LAST ", Big_Int( Short_Integer'LAST ) );
Result.Print( "" );

--* END
--* BEGIN Dec_Vax_V1_4 TeleGen2_V3_15
-- Integer
Show( "Integer'SIZE ", Big_Int( Integer'SIZE ) );
Show( "Integer'FIRST ", Big_Int( Integer'FIRST ) );
Show( "Integer'LAST ", Big_Int( Integer'LAST ) );
Result.Print( "" );

--* END
--* BEGIN TeleGen2_V3_15
-- Long_Integer
Show( "Long_Integer'SIZE ", Big_Int( Long_Integer'SIZE ) );
Show( "Long_Integer'FIRST ", Big_Int( Long_Integer'FIRST ) );
Show( "Long_Integer'LAST ", Big_Int( Long_Integer'LAST ) );
Result.Print( "" );

--* END
--* BEGIN
-- Long_Long_Integer
Show( "Long_Long_Integer'SIZE ", Big_Int( Long_Long_Integer'SIZE ) );
Show( "Long_Long_Integer'FIRST ", Big_Int( Long_Long_Integer'FIRST ) );
Show( "Long_Long_Integer'LAST ", Big_Int( Long_Long_Integer'LAST ) );
Result.Print( "" );

--* END
--* BEGIN
-- Short_Float
Show( "Short_Float'SIZE ", Big_Int( Short_Float'SIZE ) );
Show( "Short_Float'DIGITS ", Big_Int( Short_Float'DIGITS ) );
Show( "Short_Float'EMAX ", Big_Int( Short_Float'EMAX ) );
Result.Print( "" );

--* END
--* BEGIN Dec_Vax_V1_4 TeleGen2_V3_15
-- Float
Show( "Float'SIZE ", Big_Int( Float'SIZE ) );
Show( "Float'DIGITS ", Big_Int( Float'DIGITS ) );
Show( "Float'EMAX ", Big_Int( Float'EMAX ) );
Result.Print( "" );

--* END
--* BEGIN Dec_Vax_V1_4 TeleGen2_V3_15
-- Long_Float
Show( "Long_Float'SIZE ", Big_Int( Long_Float'SIZE ) );
Show( "Long_Float'DIGITS ", Big_Int( Long_Float'DIGITS ) );

```

Source File: T060301.TST

```
        Show( "Long_Float'EMAX"          ", Big_Int( Long_Float'EMAX ) );
        Result.Print( "" );

--* END
--* BEGIN  Dec_Vax_V1_4
-- Long_Long_Float
Show( "Long_Long_Float'SIZE"          ", Big_Int( Long_Long_Float'SIZE ) );
Show( "Long_Long_Float'DIGITS"        ", Big_Int( Long_Long_Float'DIGITS ) );
Show( "Long_Long_Float'EMAX"          ", Big_Int( Long_Long_Float'EMAX ) );
        Result.Print( "" );

--* END
        Result.Manual_Test( "T060301" );
END T060301;
```

Source File: T060302.TST

```
-- T060302
--
-- The compiler shall support universal integer calculations requiring up to
-- 64 bits of accuracy.
--
-- Method:
--
-- Note: The compiler should be able to perform calculations requiring
-- 64 bits of accuracy regardless of the maximum integer size of the
-- machine. The result of the calculation should be within the maximum
-- integer size of the machine.
--
-- Compile a procedure containing a statement requiring 64 bits of
-- accuracy in a calculation with the result fitting into 31 bits (max).
-- The result should be a 1.
--
--* COMPILE T060302
--* EXECUTE T060302
WITH Result;
PROCEDURE T060302 IS
  S : CONSTANT := ( 16#FFFFFFFFFFFFFFFF# - 16#FFFFFFFFFFFFFFE# );
  T : Integer;
BEGIN
  T := S;
  Result.Print( "Result of calculations = " & Result.Image( T ) );
  Result.Passed( "T060302", T = 1 );
EXCEPTION
  WHEN OTHERS => Result.Passed( "T060302", False );
END T060302;
```

Source File: T060303.TST

```
-- T060303
--
-- The components of array types with BOOLEAN components named in a PRAGMA
-- Pack shall be stored in contiguous memory bits, i.e., each component
-- shall occupy only one bit of storage.
--
-- Method:
--
-- Compile a procedure containing a packed Boolean array with 100 elements.
-- The size of the packed boolean array should be 100.
--
--* COMPILE T060303
--* EXECUTE T060303
WITH Result;
PROCEDURE T060303 IS
    TYPE List IS ARRAY( 1 .. 100 ) OF Boolean;
    PRAGMA Pack( List );
BEGIN
    Result.Print( "Packed Size: (100) =" & Result.Image( List'SIZE, 5 ) );
    Result.Passed( "T060303", List'SIZE = 100 );
END T060303;
```

Source File: T060304.TST

```
-- T060304
--
-- The compiler should support address clauses.
--
-- Method:
--
-- Compile and execute a procedure containing an address clause.
-- Assignment to an aliased variable should assign the other also.
--
--* COMPILE T060304
--* EXECUTE T060304
WITH System;
WITH Result;
PROCEDURE T060304 IS
    I1 : Integer := 0;
    I2 : Integer := 0;
    FOR I2 USE AT I1'ADDRESS;
BEGIN
    I1 := 1024;
    Result.Passed( "T060304", I1 = I2 );
END T060304;
```


Source File: T060305.TST

```
-- T060305
--
-- The compiler should support length clauses, enumeration representation
-- clauses, and record representation clauses.
--
-- Method:
--
-- Compile a procedure consisting of one of each of these three types.
-- If the procedure compiles and executes with the values printed the
-- same as expected, the test has passed.
--
--* COMPILE T060305
--* EXECUTE T060305
WITH Result;
WITH Unchecked_Conversion;
PROCEDURE T060305 IS

    FUNCTION Test_Length_Clause RETURN Boolean IS
        TYPE Small_Type IS RANGE 0 .. 15;
        FOR Small_Type'SIZE USE 4;
    BEGIN
        RETURN Small_Type'SIZE = 4;
    END Test_Length_Clause;

    FUNCTION Test_Enumeration_Clause RETURN Boolean IS
        TYPE Enum_A IS ( Four_A, Five_A, Six_A );
        FOR Enum_A USE ( Four_A => 4, Five_A => 5, Six_A => 6 );

        TYPE Enum_B IS ( Five_B, Six_B, Seven_B );
        FOR Enum_B USE ( Five_B => 5, Six_B => 6, Seven_B => 7 );

        FUNCTION Convert IS NEW Unchecked_Conversion( Enum_A, Enum_B );
    BEGIN
        RETURN Convert( Six_A ) = Six_B;
    END Test_Enumeration_Clause;

    FUNCTION Test_Record_Clause RETURN Boolean IS

        TYPE Integer_1 IS RANGE 0 .. 15; -- Size = 4 Bits
        TYPE Integer_2 IS RANGE 0 .. 255; -- Size = 8 Bits

        TYPE A_Record IS RECORD
            Entry_1 : Integer_1;
            Entry_2 : Integer_1;
            Entry_3 : Integer_2;
        END RECORD;

        FOR A_Record USE RECORD AT MOD 4;
            Entry_1 AT 0 RANGE 0 .. 3;
            Entry_2 AT 0 RANGE 4 .. 7;
            Entry_3 AT 16 RANGE 0 .. 7;
        END RECORD;

        TYPE B_Record IS RECORD
            Entry_1 : Integer_1;
            Entry_2 : Integer_1;
            Entry_3 : Integer_2;
        END RECORD;

        FOR B_Record USE RECORD AT MOD 4;
            Entry_1 AT 16 RANGE 0 .. 3;
            Entry_2 AT 16 RANGE 4 .. 7;
            Entry_3 AT 0 RANGE 0 .. 7;
        END RECORD;

        FUNCTION Convert IS NEW Unchecked_Conversion( A_Record, B_Record );

        A_1 : A_Record := A_Record'( 15, 15, 0 );
        A_2 : A_Record := A_Record'( 0, 0, 255 );
        B_1 : B_Record := Convert( A_1 );
        B_2 : B_Record := Convert( A_2 );
```

Source File: T060305.TST

```

    R_1 : B_Record := B_Record'( 0, 0, 255 );
    R_2 : B_Record := B_Record'( 15, 15, 0 );
BEGIN
    RETURN B_1 = R_1 AND B_2 = R_2;
END Test_Record_Clause;

FUNCTION Value( Success : Boolean; Name : String ) RETURN Natural IS
BEGIN
    CASE Success IS
        WHEN True => Result.Print( Name & " PASSED" ); RETURN 100;
        WHEN False => Result.Print( Name & " FAILED" ); RETURN 0;
    END CASE;
END Value;

BEGIN
    Result.Passed
    ( "T060305",
      ( Value( Test_Length_Clause, "Length Clause" ) +
        Value( Test_Enumeration_Clause, "Enumeration Clause" ) +
        Value( Test_Record_Clause, "Record Clause" ) ) / 3 );
END T060305;
```

Source File: T060306.TST

```
-- T060306
--
-- The range of integer code values allowed in an enumeration representation
-- clause shall be MIN_INT to MAX_INT.
--
-- Method:
--
-- Declare an enumeration type with an enumeration representation clause
-- assigning Min_Int and Max_Int as values.
--
--* COMPILE T060306
--* EXECUTE T060306
WITH Result;
WITH System;
WITH Unchecked_Conversion;
PROCEDURE T060306 IS

    TYPE New_Integer IS RANGE System.Min_Int .. System.Max_Int;

    TYPE Enum_Type IS ( First, Middle, Last );
    FOR Enum_Type USE ( First => System.Min_Int,
                        Middle => 0,
                        Last  => System.Max_Int );

    FUNCTION Convert IS NEW Unchecked_Conversion( Enum_Type, New_Integer );

BEGIN
    Result.Passed( "T060306", Convert( First ) = System.Min_Int );
EXCEPTION
    WHEN OTHERS => Result.Passed( "T060306", False );
END T060306;
```

Source File: T060307.TST

```
-- T060307
--
-- The compiler shall allow non-contiguous integer code values in an
-- enumeration representation clause.
--
-- Method:
--
-- Compile a procedure consisting of an enumeration representation
-- clause with non-contiguous values. The test has passed if the
-- conversion of a value to another shows the code values to be the same.
--
--* COMPILE T060307
--* EXECUTE T060307
WITH Result;
WITH Unchecked_Conversion;
PROCEDURE T060307 IS

    TYPE Enum_A IS ( One_A, Two_A, Three_A, Four_A );
    FOR Enum_A USE ( One_A => 1, Two_A => 4, Three_A => 8, Four_A => 64 );
    TYPE Enum_B IS ( One_B, Two_B, Three_B, Four_B );
    FOR Enum_B USE ( One_B => 1, Two_B => 8, Three_B => 16, Four_B => 64 );

    FUNCTION Convert IS NEW Unchecked_Conversion( Enum_A, Enum_B );

BEGIN
    Result.Passed( "T060307", Convert( Three_A ) = Two_B );
EXCEPTION
    WHEN OTHERS => Result.Passed( "T060307", False );
END T060307;
```

Source File: T060308.TST

```
-- T060308
--
-- The compiler should support the SIZE attribute designator for enumeration
-- types named in a length clause.
--
-- Method:
--
-- Compile a procedure consisting of a length clause and execute to
-- determine if the SIZE attribute is supported. If the compilation
-- and execution proceed without error, the test has passed.
--
--* COMPILE T060308
--* EXECUTE T060308
WITH Result;
PROCEDURE T060308 IS
    TYPE Small_Type IS ( Zero, One, Two, Three, Four, Five, Six, Seven, Eight );
    FOR Small_Type'SIZE USE 4;
BEGIN
    Result.Print( "Size should be 4:" & Result.Image( Small_Type'SIZE, 5 ) );
    Result.Passed( "T060308", Small_Type'SIZE = 4 );
END T060308;
```

Source File: T060309.TST

```
-- T060309
--
-- The compiler should support the SMALL attribute designator for fixed
-- point types.
--
-- Method:
--
-- Compile a procedure consisting of a fixed point type and execute
-- to determine if the SMALL attribute is supported. The compiler
-- will have passed this test if the compilation and execution proceed
-- without error and the attribute value is as expected.
--
--* COMPILE T060309
--* EXECUTE T060309
WITH Result;
PROCEDURE T060309 IS

    TYPE Small_Fixed IS DELTA 0.125 RANGE 0.0 .. 255.0;

    Expected : CONSTANT Float := 0.125;

BEGIN
    Result.Print( "Expected DELTA:" & Result.Image( Expected, 8, 3 ) );
    Result.Print( "Observed DELTA:" & Result.Image( Small_Fixed'SMALL, 8, 3 ) );
    Result.Passed( "T060309", Expected = Small_Fixed'SMALL );
END T060309;
```

Source File: T060310.TST

```
-- T060310
--
-- Memory space for the creation of objects designated by an access
-- type shall not be allocated until allocators (new statements) for
-- that type are executed.
--
-- Method:
--
-- Declare an array of access variables to a big record type.
-- If execution is able to start, but a memory error occurs
-- before each of these elements is allocated with a NEW statement,
-- then the compiler has not allocated memory space before the
-- NEW statements and passes the test.
--
--* COMPILE T060310
--* EXECUTE T060310
WITH Result;
PROCEDURE T060310 IS

    SUBTYPE Big_Range IS Integer RANGE 1 .. 10000;

    TYPE Big_Record IS RECORD
        Variable : String( Big_Range );
    END RECORD;

    TYPE Big_Access IS ACCESS Big_Record;
    TYPE Big_Array IS ARRAY( Big_Range ) OF Big_Access;

    Big_Var : Big_Array;

BEGIN
    FOR Index IN Big_Range LOOP
        Big_Var( Index ) := NEW Big_Record;
    END LOOP;
    Result.Inconclusive( "T060310" );
EXCEPTION
    WHEN Storage_Error => Result.Passed( "T060310", True );
    WHEN OTHERS        => Result.Passed( "T060310", False );
END T060310;
```

Source File: T060401.TST

```
-- T060401
--
-- The compiler shall expand inline any subprogram or generic subprogram
-- instantiation that is named in a PRAGMA Inline and that meets the criteria
-- of 6.4.2
--
-- Method:
--
-- Compile a procedure containing a subprogram and a generic meeting
-- the requirements of 6.4.2, check for a call statement in the assembly
-- language code. SUM_1 and SWAP_1 should be expanded inline, SUM_2 and
-- SWAP_2 should not be expanded.
--
--* COMPILE T060401 ASSEMBLY_LISTING
--* EXECUTE T060401
WITH Result;
PROCEDURE T060401 IS

    Var_1, Var_2, Var_3 : Integer := 64;

    FUNCTION Sum_1( X, Y : Integer ) RETURN Integer IS
        Total, Sum : Integer;
    BEGIN
        Sum := X + Y;
        FOR Counter IN 1 .. 100 LOOP
            Sum := Sum + X;
        END LOOP;
        Total := Sum / X;
        RETURN Total;
    END Sum_1;

    FUNCTION Sum_2( X, Y : Integer ) RETURN Integer IS
        Total, Sum : Integer;
    BEGIN
        Sum := X + Y;
        FOR Counter IN 1 .. 100 LOOP
            Sum := Sum + X;
        END LOOP;
        Total := Sum / X;
        RETURN Total;
    END Sum_2;

    GENERIC
        TYPE Item IS PRIVATE;
    PROCEDURE Exchange( X, Y : IN OUT Item );

    PROCEDURE Exchange( X, Y : IN OUT Item ) IS
        T : Item;
    BEGIN
        T := X; X := Y; Y := T;
    END Exchange;

    PROCEDURE Swap_1 IS NEW Exchange( Integer );
    PROCEDURE Swap_2 IS NEW Exchange( Integer );

    PRAGMA INLINE( Sum_1, Swap_1 );

BEGIN
    Var_1 := Sum_1( Var_2, Var_3 ); -- Inline
    Var_2 := Sum_1( Var_1, Var_3 ); -- Inline
    --
    Var_1 := Sum_2( Var_2, Var_3 ); -- not Inline
    Var_2 := Sum_2( Var_1, Var_3 ); -- not Inline
    --
    Swap_1( Var_1, Var_2 );          -- Inline
    Swap_1( Var_2, Var_3 );          -- Inline
    --
    Swap_2( Var_1, Var_2 );          -- not Inline
    Swap_2( Var_2, Var_3 );          -- not Inline
    --
    Result.Manual_Test( "T060401" );
END T060401;
```


Source File: T060402.TST

```
-- T060402
--
-- A subprogram or generic instantiation is a candidate for inline expansion
-- if it meets the following criteria:
--   a. Its body is declared in either the current unit or the compilation
--      library.
--   b. Its parameters or result type (for functions) are not task types,
--      composite types with task type components, unconstrained array
--      types, or unconstrained types with discriminants.
--   c. It does not contain another subprogram body, package body, body
--      stub, generic declaration, generic instantiation, exception
--      declaration, or access type declaration.
--   d. It does not contain declarations that imply the creation of
--      dependent tasks.
--   e. It does not contain any subprogram calls that result in direct or
--      indirect recursion.
--
-- Method:
--
-- Definition.
--
--* COMPILE T060402
--* EXECUTE T060402
WITH Result;
PROCEDURE T060402 IS
BEGIN
    Result.Not_Applicable( "T060402", "Definition." );
END T060402;
```

Source File: T060403.TST

```
-- T060403
--
-- The compiler shall expand inline any subprogram that meets the
-- requirements of 6.4.2 and that is called only once.
--
-- Method:
--
-- Compile a procedure containing a subprogram meeting the requirements
-- of 6.4.2, check for a call statement in the assembly language code.
-- The SUM_1 function is called only once and should be expanded inline.
--
--* COMPILER T060403 ASSEMBLY_LISTING OPTIMIZE_TIME
--* EXECUTE T060403
WITH Result;
PROCEDURE T060403 IS
    Var_1, Var_2, Var_3 : Integer := 64;

    FUNCTION Sum_1( X, Y : Integer ) RETURN Integer IS
        Total, Sum : Integer;
    BEGIN
        Sum := X + Y;
        FOR Counter IN 1 .. 100 LOOP
            Sum := Sum + X;
        END LOOP;
        Total := Sum / X;
        RETURN Total;
    END Sum_1;

    FUNCTION Sum_2( X, Y : Integer ) RETURN Integer IS
        Total, Sum : Integer;
    BEGIN
        Sum := X + Y;
        FOR Counter IN 1 .. 100 LOOP
            Sum := Sum + X;
        END LOOP;
        Total := Sum / X;
        RETURN Total;
    END Sum_2;

BEGIN
    Var_1 := Sum_1( Var_2, Var_3 ); -- Inline
    --
    Var_1 := Sum_2( Var_2, Var_3 ); -- not Inline
    Var_2 := Sum_2( Var_1, Var_3 ); -- not Inline

    Result.Manual_Test( "T060403" );
END T060403;
```

Source File: T060404.TST

```
-- T060404
--
-- The compiler shall provide the capability for main subprograms to return
-- a value to the target computer run-time system indicating the completion
-- status of the program.
--
-- Method:
--
-- Define a function which returns an integer value. This value is
-- dependent on the Operating System. Find a value for success and
-- failure and insert them into the test code. If there exists a
-- value which indicates success and one for failure, the test passes.
--
--* COMPILE T060404
--* EXECUTE T060404
--* EXECUTE Normal_Return
--* EXECUTE Error_Return

WITH Result;
PROCEDURE T060404 IS
BEGIN
    Result.Manual_Test( "T060404" );
END T060404;

--* BEGIN  Dec_Vax_V1_4  TeleGen2_V3_15

FUNCTION Normal_Return RETURN Integer IS
BEGIN
    RETURN 1;
END Normal_Return;

FUNCTION Error_Return RETURN Integer IS
BEGIN
    RETURN 0;
END Error_Return;

--* END
```

Source File: T060501.TST

```
-- T060501
--
-- The compiler shall provide a capability for handling target computer
-- hardware or operating system interrupts as calls to Ada task entries.
--
-- Method:
--
-- Check the compiler documentation for a method of handling the
-- interrupts as Ada task entry calls.
--
--* BEGIN  Dec_Vax_V1_4  TeleGen2_V3_15
--* COMPILE T060501
--* EXECUTE T060501
WITH Result;
PROCEDURE T060501 IS
BEGIN
    Result.Not_Applicable( "T060501", "Not appropriate for VAX VMS V1.4." );
END T060501;
--* END
```

```
-- T060502
--
-- The execution-time overhead to perform a context switch or to terminate
-- or abort a task shall be no more than that required to call or return
-- from a subprogram.
--
-- Method:
--
-- Define a task and a procedure that perform identical functions. Place
-- a procedure call in a loop and time its execution. Do the same for
-- an entry call to the task. This control loop overhead time is subtracted
-- from the observed time.
--
-- NO_OPTIMIZE may be used as an option to ensure the procedure call is
-- not expanded inline. This test should be compiled once for each
-- compiler with ASSEMBLY_LISTING as an option to verify that the
-- procedure calls are not being expanded inline.
--
--* COMPILE T060502 NO_OPTIMIZE
--* EXECUTE T060502
WITH Times;
WITH Result;
PROCEDURE T060502 IS

    SUBTYPE Bounds IS Integer RANGE 1 .. 50000;

    Checks : Times.Time_List( 1 .. 3 );
    Time_1 : Times.Time_Type;
    Time_2 : Times.Time_Type;

    X : Integer := 1;
    Y : Integer := 2;
    T : Integer := 3;

    TASK A_Task IS
        ENTRY An_Entry( X, Y : IN OUT Integer );
    END A_Task;

    TASK BODY A_Task IS
        T : Integer;
    BEGIN
        LOOP
            ACCEPT An_Entry( X, Y : IN OUT Integer ) DO
                T := X; X := Y; Y := T;
            END An_Entry;
        END LOOP;
    END A_Task;

    PROCEDURE A_Procedure( X, Y : IN OUT Integer ) IS
        T : Integer;
    BEGIN
        T := X; X := Y; Y := T;
    END A_Procedure;

    FUNCTION "&"( Text : String; Value : Integer ) RETURN String IS
    BEGIN
        RETURN Text & Result.Image( Value, 8 );
    END "&";

    FUNCTION "&"( Text : String; Value : Float ) RETURN String IS
    BEGIN
        RETURN Text & Result.Image( Value, 8, 2 );
    END "&";

    FUNCTION "&"( Text : String; Time : Times.Time_Type ) RETURN String IS
    BEGIN
        RETURN Text & Times.Image( Time );
    END "&";
```

```

END "&";

PROCEDURE Print_Time( Time : Times.Time_Type; Name : String ) IS
BEGIN
    Result.Print( Name & " Iterations: " & Bounds'LAST & " Time: " & Time );
END Print_Time;

PROCEDURE Print_Result( Delta_1, Delta_2 : Float ) IS
    Cutoff : CONSTANT Float := 0.04;
    Percent : Float;
BEGIN
    Result.Print( "Procedure Time Minus Control Time =" & Delta_1 );
    Result.Print( "Task Time Minus Control Time =" & Delta_2 );
    IF Delta_1 < Cutoff OR ELSE Delta_2 < Cutoff THEN
        Result.Inconclusive( "T060502", "Insufficient time for test." );
    ELSIF NOT Times.Repeatable( Checks ) THEN
        Percent := Delta_1 / Delta_2 * 100.0;
        Result.Print( "Procedure/Task Ratio:" & Percent & "%" );
        Result.Inconclusive( "T060502", "Times not repeatable." );
    ELSE
        Percent := Delta_1 / Delta_2 * 100.0;
        Result.Print( "Procedure/Task Ratio:" & Percent & "%" );
        Result.Passed( "T060502", Result.Min( Natural( Percent ), 100 ) );
    END IF;
END Print_Result;

BEGIN

    FOR Control IN Checks'RANGE LOOP
        Times.Reset_Time;
        FOR Count IN Bounds LOOP
            T := X; X := Y; Y := T;
        END LOOP;
        Checks( Control ) := Times.Current_Time;
    END LOOP;

    Times.Reset_Time;
    FOR Count IN Bounds LOOP
        A_Procedure( X, Y );
    END LOOP;
    Time_1 := Times.Current_Time;

    Times.Reset_Time;
    FOR Count IN Bounds LOOP
        A_Task.An_Entry( X, Y );
    END LOOP;
    Time_2 := Times.Current_Time;

    FOR Control IN Checks'RANGE LOOP
        Print_Time( Checks( Control ), "Control " );
    END LOOP;
    Print_Time( Time_1, "Procedure" );
    Print_Time( Time_2, "Task " );

    Print_Result( Times.Difference( Time_1, Times.Min( Checks ) ),
                  Times.Difference( Time_2, Times.Min( Checks ) ) );

    ABORT A_Task;
EXCEPTION
    WHEN OTHERS =>
        ABORT A_Task;
        Result.Inconclusive( "T060502", "Program Error." );
END T060502;

```

```
-- T060503
--
-- The ordering of select alternatives in a selective wait statement
-- shall not impact the execution speed of the program.
--
-- Method:
--
-- Compile and run a program with two identical tasks except for the
-- ordering of the select statements. The same entry call is made
-- repeatedly for both tasks, except each entry select alternative is
-- in a different position. The first and last entry statement in
-- each task is timed. The times are taken more than once to ensure
-- repeatability.
--
-- The maximum variation in time for a single entry is divided by the
-- maximum variation in time for all entry times measured.
--
--* COMPILE T060503
--* EXECUTE T060503
--! EQUATE Count IS 10
WITH Times;
WITH Result;
PROCEDURE T060503 IS

    SUBTYPE Bounds IS Integer RANGE 1 .. 10000;

    SUBTYPE Checks    IS Integer RANGE 1 .. 4;
    SUBTYPE Task_IDs  IS Integer RANGE 1 .. 2;
    SUBTYPE Entry_IDs IS Integer RANGE 1 .. 2;

    Name : CONSTANT ARRAY( Entry_IDs ) OF String( 1..5 ) := ( "First","Last " );

    Time : ARRAY( Task_IDs, Entry_IDs ) OF Times.Time_List( Checks );

    X : Integer := 1;

    TASK Task_1 IS -- Ascending Order
        --! LOOP Count START 1 STEP 1 [1]
        ENTRY Entry_1( X : IN OUT Integer );
        --! END [1]
    END Task_1;

    TASK Task_2 IS -- Descending Order
        --! LOOP Count START Count STEP -1 [1]
        ENTRY Entry_1( X : IN OUT Integer );
        --! END [1]
    END Task_2;

    --! LOOP 2 [1]
    TASK BODY Task_1 IS
    BEGIN
        LOOP
            SELECT
                ACCEPT Entry_1( X : IN OUT Integer ) DO
                    X := 11;
                END Entry_1;
                --! START 2 LOOP Count-1 [2]
                OR ACCEPT Entry_2( X : IN OUT Integer ) DO
                    X := [2+10];
                END Entry_2;
            --! END [2]
        END SELECT;
    END LOOP;
    END Task_1;

    --! END [1]

    FUNCTION "&"( Text : String; Item : Integer ) RETURN String IS
    BEGIN
        RETURN Text & Result.Image( Item, 2 );
    END "&";
```

Source File: T060503.TST

```
FUNCTION "&"( Text : String; Item : Float ) RETURN String IS
BEGIN
    RETURN Text & Result.Image( Item, 8, 2 );
END "&;
```

```
FUNCTION "&"( Text : String; Item : Times.Time_Type ) RETURN String IS
BEGIN
    RETURN Text & Times.Image( Item );
END "&;
```

```
PROCEDURE Print_Results IS
    Repeatable : Boolean := True;
    Time_Bound : Boolean := True;
    Max_Time : Times.Time_Type := Times.Time_Type_First;
    Min_Time : Times.Time_Type := Times.Time_Type_Last;
```

```
FUNCTION Ratio( Min, Max : Times.Time_Type ) RETURN Float IS
    Low : Float := Times.Seconds( Min );
    High : Float := Times.Seconds( Max );
BEGIN
    RETURN 100.0 * Low / High;
EXCEPTION
    WHEN OTHERS => RETURN 0.0;
END Ratio;
```

```
PROCEDURE Process( List : Times.Time_List ) IS
    Next : Times.Time_Type := Times.Min( List );
BEGIN
    FOR Attempt IN List'RANGE LOOP
        Result.Print( "Time" & Attempt & " --> " & List( Attempt ) );
    END LOOP;
    Repeatable := Repeatable AND THEN Times.Repeatable( List );
    Time_Bound := Time_Bound AND THEN Times.Seconds( Next ) >= 1.0;
    Max_Time := Times.Max( Max_Time, Next );
    Min_Time := Times.Min( Min_Time, Next );
END Process;
```

```
BEGIN
    Result.Print( "Iterations:" & Integer'IMAGE( Bounds'LAST ) );
    FOR Task_ID IN Task_IDs LOOP
        FOR Entry_ID IN Entry_IDs LOOP
            Result.Print( "Task" & Task_ID & ": Entry" & Entry_ID & ":" );
            Process( Time( Task_ID, Entry_ID ) );
        END LOOP;
    END LOOP;
    Result.Print( "Lowest Minimum Time:" & Times.Image( Min_Time ) );
    Result.Print( "Highest Minimum Time:" & Times.Image( Max_Time ) );
    Result.Print( "Percent Difference: " & Ratio( Min_Time, Max_Time ) & "%" );
    IF NOT Time_Bound THEN
        Result.Inconclusive( "T060503", "Insufficient time for test." );
    ELSIF NOT Repeatable THEN
        Result.Inconclusive( "T060503", "Times not repeatable." );
    ELSE
        Result.Passed( "T060503", Natural( Ratio( Min_Time, Max_Time ) ) );
    END IF;
END Print_Results;
```

BEGIN

```
FOR Attempt IN Checks LOOP

    --! LOOP 2 [1]
    --! LOOP 1 START COUNT [2]
    Times.Reset_Time;
    FOR Count IN Bounds LOOP
        Task_[1].Entry_1( X ); -- First Entry
    END LOOP;
    Time( [1], 1 )( Attempt ) := Times.Current_Time;
```


Source File: T060503.TST

```
Times.Reset_Time;
FOR Count IN Bounds LOOP
    Task_[1].Entry_[2]( X ); -- Last Entry
END LOOP;
Time( [1], 2 )( Attempt ) := Times.Current_Time;

--! END [2]
--! END [1]

END LOOP;

Print_Results;

ABORT Task_1;
ABORT Task_2;
EXCEPTION
    WHEN OTHERS =>
        ABORT Task_1;
        ABORT Task_2;
        Result.Inconclusive( "T060503", "Program Error." );
END T060503;
```

Source File: T060504.TST

```
-- T060504
--
-- The compiler shall dispatch the execution of ready tasks in a manner that
-- will give each task an equal share of the processing resources consistent
-- with any priority pragmas.
--
-- Method:
--
-- Compile a procedure containing several tasks with the same priority.
-- If each task is invoked with the same frequency the test has passed.
--
--* COMPILE T060504
--* EXECUTE T060504
--! EQUATE Tasks IS 5
WITH Result;
PROCEDURE T060504 IS

    --! START Tasks LOOP 1 [1]
    Tasks      : CONSTANT := [1];
    --! END [1]
    Expected   : CONSTANT := 100;
    Total      : CONSTANT := Expected * Tasks;

    Frequency  : ARRAY( 1 .. Tasks ) OF Natural := ( OTHERS => 0 );

    TASK Controller IS
        ENTRY Hello;
        --! LOOP Tasks [1]
        ENTRY Wait_Sub_Task_1( Done : OUT Boolean );
        --! END [1]
        ENTRY Goodbye;
    END Controller;

    --! LOOP Tasks [1]
    TASK Sub_Task_1;
    --! END [1]

    TASK BODY Controller IS
    BEGIN
        ACCEPT Hello;
        FOR Count IN 1 .. Total LOOP
            SELECT
                ACCEPT Wait_Sub_Task_1( Done : OUT Boolean ) DO
                    Done := False;
                END Wait_Sub_Task_1;
                --! START 2 LOOP Tasks-1 [1]
                OR ACCEPT Wait_Sub_Task_1( Done : OUT Boolean ) DO
                    Done := False;
                END Wait_Sub_Task_1;
                --! END [1]
            END SELECT;
        END LOOP;
        --! LOOP Tasks [1]
        ACCEPT Wait_Sub_Task_1( Done : OUT Boolean ) DO
            Done := True;
        END Wait_Sub_Task_1;
        --! END [1]
        ACCEPT Goodbye;
    END Controller;

    --! LOOP Tasks [1]
    TASK BODY Sub_Task_1 IS
        Count : Natural := 0;
        Done   : Boolean := False;
    BEGIN
        LOOP
            Controller.Wait_Sub_Task_1( Done );
            EXIT WHEN Done;
            Frequency( [1] ) := Frequency( [1] ) + 1;
        END LOOP;
    END Sub_Task_1;
    --! END [1]
```

Source File: T060504.TST

```
FUNCTION "&"( Text : String; Value : Integer ) RETURN String IS
BEGIN
    RETURN Text & Result.Image( Value, 6 );
END "&";

PROCEDURE Print_Results IS
    Extra : Natural := 0;

    FUNCTION Ratio RETURN Natural IS
    BEGIN
        RETURN Result.Max( 0, 100 * ( Expected - Extra ) / Expected );
    END Ratio;

BEGIN
    FOR Count IN 1 .. Tasks LOOP
        Result.Print("Task "&Count&" Executed "&Frequency(Count)&" Times.");
        Extra := Result.Max( ABS( Frequency( Count ) - Expected ), Extra );
    END LOOP;
    Result.Passed( "T060504", Ratio );
END Print_Results;

BEGIN
    Controller.Hello;
    Controller.Goodbye;
    Print_Results;
EXCEPTION
    WHEN OTHERS => Result.Inconclusive( "T060504", "Program Error." );
END T060504;
```

Source File: T060505.TST

```
-- T060505
--
-- Tasks that are blocked, completed, terminated, or not activated shall not
-- impact the performance of the active tasks.
--
-- Method:
--
-- Execute two versions of a program, one version containing only a prime
-- task, and the other version containing the prime task in addition to
-- 20 other tasks. From these two version, take the following times:
--
--     1. Time of the task stand alone. (Control time = not activated)
--     2. Time of the task with 20 tasks blocked.
--     3. Time of the task with 10 blocked and 10 completed.
--     4. Time of the task with 10 completed and 10 aborted.
--
--* COMPILE T060505
--* EXECUTE GROUP_A
--* EXECUTE GROUP_B
--* EXECUTE T060505
--! EQUATE Iter IS 20
--! EQUATE Half IS Iter / 2
WITH Times;
WITH Result;
PACKAGE Shared IS

    SUBTYPE Bounds IS Integer RANGE 1 .. 1000000;
    SUBTYPE Checks IS Integer RANGE 1 .. 3;
    SUBTYPE Tests IS Integer RANGE 1 .. 4;

    Save : Times.Time_List( Checks );

    FUNCTION File_Name( Test : Tests ) RETURN String;

    PROCEDURE Record_Time
    ( Time : Times.Time_Type;
      Test : Tests;
      Try : Checks;
      Text : String );

END Shared;

PACKAGE BODY Shared IS

    FUNCTION "&"( Text : String; Value : Integer ) RETURN String IS
    BEGIN
        RETURN Text & Result.Image( Value, 1 );
    END "&";

    FUNCTION File_Name( Test : Tests ) RETURN String IS
    BEGIN
        RETURN "TIME" & Test;
    END File_Name;

    PROCEDURE Record_Time
    ( Time : Times.Time_Type;
      Test : Tests;
      Try : Checks;
      Text : String ) IS
    BEGIN
        Save( Try ) := Time;
        IF Try = Checks'FIRST THEN
            Result.Print( "" );
            Result.Print( "Test " & Test & ": " & Text );
            Result.Print( "Iterations: " & Integer'IMAGE( Bounds'LAST ) );
        END IF;
        Result.Print( "Time " & Try & " " & Times.Image( Time ) );
        IF Try = Checks'LAST THEN
            Times.Put_Time( File_Name( Test ), Times.Min( Save ) );
            IF NOT Times.Repeatable( Save ) THEN
```

```

        Result.Print( "**** Times were not repeatable ****" );
    END IF;
END IF;
END Record_Time;

END Shared;

WITH Times;
WITH Shared;
PROCEDURE Group_A IS
    A : Integer := 1;
    B : Integer := 2;

    TASK A_Task IS
        ENTRY An_Entry( X, Y : IN OUT Integer );
    END A_Task;

    TASK BODY A_Task IS
        T : Integer;
    BEGIN
        LOOP
            ACCEPT An_Entry( X, Y : IN OUT Integer ) DO
                FOR Count IN Shared.Bounds LOOP
                    T := X; X := Y; Y := T;
                END LOOP;
            END An_Entry;
        END LOOP;
    END A_Task;

    PROCEDURE Run_Test( Test : Shared.Tests; Text : String ) IS
    BEGIN
        FOR Index IN Shared.Checks LOOP
            Times.Reset_Time;
            A_Task.An_Entry( A, B );
            Shared.Record_Time( Times.Current_Time, Test, Index, Text );
        END LOOP;
    END Run_Test;

BEGIN
    Run_Test( 1, "Time with a single task in the system. (= Not Activated)" );
    ABORT A_Task;
END Group_A;

WITH Times;
WITH Shared;
PROCEDURE Group_B IS
    A : Integer := 1;
    B : Integer := 2;

    TASK A_Task IS
        ENTRY An_Entry( X, Y : IN OUT Integer );
    END A_Task;

    --! LOOP Iter [1]
    TASK Task_1 IS
        ENTRY Blocked;
    END Task_1;
    --! END [1]

    TASK BODY A_Task IS
        T : Integer;
    BEGIN
        LOOP
            ACCEPT An_Entry( X, Y : IN OUT Integer ) DO
                FOR Count IN Shared.Bounds LOOP
                    T := X; X := Y; Y := T;
                END LOOP;
            END An_Entry;

```

Source File: T060505.TST

```
        END LOOP;
    END A_Task;

    --! LOOP Iter [1]
    TASK BODY Task_[1] IS
    BEGIN
        ACCEPT Blocked;
    END Task_[1];
    --! END [1]

    PROCEDURE Run_Test( Test : Shared.Tests; Text : String ) IS
    BEGIN
        FOR Index IN Shared.Checks LOOP
            Times.Reset_Time;
            A_Task.An_Entry( A, B );
            Shared.Record_Time( Times.Current_Time, Test, Index, Text );
        END LOOP;
    END Run_Test;

BEGIN
    --! START Iter LOOP 1 [1]
    Run_Test( 2, "Time with [1] blocked tasks in the system. (= Blocked)" );
    --! END [1]

    -- Release half of the tasks
    --! START 1 LOOP Half STEP 2 [1]
    Task_[1].Blocked;
    --! END [1]

    --! START Half LOOP 1 [1]
    Run_Test( 3, "Time with [1] blocked tasks completed. (= Completed)" );
    --! END [1]

    -- Abort the remaining tasks
    --! START 2 LOOP Half STEP 2 [1]
    ABORT Task_[1];
    --! END [1]

    --! START Half LOOP 1 [1]
    Run_Test( 4, "Time with remaining [1] tasks aborted. (= Terminated)" );
    --! END [1]

    ABORT A_Task;
END Group_B;

WITH Times;
WITH Result;
WITH Shared;
PROCEDURE T060505 IS

    Save : Times.Time_List( Shared.Tests );

BEGIN
    FOR Index IN Shared.Tests LOOP
        Times.Get_Time
        ( Shared.File_Name( Index ), Save( Index ), Delete_File => True );
    END LOOP;
    Result.Passed( "T060505", Times.Repeatable_Percent( Save ) );
EXCEPTION
    WHEN OTHERS => Result.Inconclusive( "T060505", "Program Error." );
END T060505;
```

Source File: T060506.TST

```
-- T060506
--
-- The value of DURATION'DELTA shall not be greater than 1 millisecond.
--
-- Method:
--
-- Examine DURATION'DELTA.
--
--* COMPILE T060506
--* EXECUTE T060506
WITH Result;
PROCEDURE T060506 IS
BEGIN
    Result.Print( "Duration'Delta = " & Result.Image( Duration'DELTA, 8, 4 ) );
    Result.Passed( "T060506", Duration'DELTA <= 0.001 );
END T060506;
```

Source File: T060601.TST

```
-- T060601
--
-- An exception shall not impact execution speed until it is raised.
--
-- Method:
--
-- Compare the execution time of a procedure with exception handlers
-- to the execution time of a procedure without exception handlers.
--
--* COMPILE T060601
--* EXECUTE TEST_A
--* EXECUTE TEST_B
--* EXECUTE T060601
--! EQUATE Count IS 10
WITH Times;
WITH Result;
PACKAGE Shared IS

    SUBTYPE Bounds IS Integer RANGE 1 .. 1000000;
    SUBTYPE Checks IS Integer RANGE 1 .. 3;

    PROCEDURE Print_Result
        ( List : Times.Time_List;
          Text : String;
          Name : String );

END Shared;

PACKAGE BODY Shared IS

    PROCEDURE Print_Result
        ( List : Times.Time_List;
          Text : String;
          Name : String ) IS

        FUNCTION "&"( Text : String; Value : Integer ) RETURN String IS
        BEGIN
            RETURN Text & Result.Image( Value, 2 );
        END "&";

    BEGIN
        Result.Print( "" );
        Result.Print( "Iterations:" & Integer'IMAGE(Bounds'LAST)& ": " & Text );
        FOR Index IN List'RANGE LOOP
            Result.Print( "Time" & Index & ": " & Times.Image( List( Index ) ));
        END LOOP;
        IF NOT Times.Repeatable( List ) THEN
            Result.Print( "**** Times Not Repeatable ****" );
        END IF;
        Times.Put_Time( Name, Times.Min( List ) );
    END Print_Result;

END Shared;

WITH Times;
WITH Shared;
PROCEDURE Test_A IS

    X, Y, Z : Integer := 1;
    Save    : Times.Time_List( Shared.Checks );

    PROCEDURE Test( X, Y, Z : IN OUT Integer ) IS
    BEGIN
        FOR Index IN Shared.Bounds LOOP
            X := X + 1;
            Y := X - 1;
            Z := X - Y;
        END LOOP;
    END Test;

BEGIN
    FOR Index IN Shared.Checks LOOP
```


Source File: T060601.TST

```
        Times.Reset_Time;
        Test( X, Y, Z );
        Save( Index ) := Times.Current_Time;
    END LOOP;
    Shared.Print_Result( Save, "Procedure without exceptions.", "TESTA" );
END Test_A;

WITH Times;
WITH Shared;
PROCEDURE Test_B IS

    --! LOOP Count [1]
    Ex [1] : EXCEPTION;
    --! END [1]

    X, Y, Z : Integer := 1;
    Save    : Times.Time_List( Shared.Checks );

    PROCEDURE Test( X, Y, Z : IN OUT Integer ) IS
    BEGIN
        FOR Index IN Shared.Bounds LOOP
            X := X + 1;
            Y := X - 1;
            Z := X - Y;
        END LOOP;
    EXCEPTION
        --! LOOP Count [1]
        WHEN Ex [1] => Y := [1];
        --! END [1]
    END Test;

BEGIN
    FOR Index IN Shared.Checks LOOP
        Times.Reset_Time;
        Test( X, Y, Z );
        Save( Index ) := Times.Current_Time;
    END LOOP;
    Shared.Print_Result( Save, "Procedure with exceptions.", "TESTB" );
END Test_B;

WITH Times;
WITH Result;
PROCEDURE T060601 IS
    TimeA : Times.Time_Type;
    TimeB : Times.Time_Type;
BEGIN
    Times.Get_Time( "TESTA", TimeA, Delete_File => True );
    Times.Get_Time( "TESTB", TimeB, Delete_File => True );
    Result.Passed( "T060601", Times.Repeatable_Percent( ( TimeA, TimeB ) ) );
END T060601;
```

Source File: T060602.TST

```
-- T060602
--
-- The compiler shall provide the PRAGMA Suppress or an equivalent capability
-- to permit suppression of all predefined run-time checks in a designated
-- compilation unit.
--
-- Method:
--
-- Place the compiler specific suppression mechanism in the code given
-- below at its appropriate position. If the message "Checks Suppressed."
-- gets printed, the test has been successful.
--
--* COMPILE T060602
--* EXECUTE T060602
WITH Result;
PROCEDURE T060602 IS

    SUBTYPE Small_Range IS Integer RANGE 0 .. 2;
    X : Small_Range := 0;

BEGIN
    X := X + 1;
    X := X + 1;
    X := X + 1;
    X := X + 1;
    Result.Passed( "T060602", True );
EXCEPTION
    WHEN OTHERS => Result.Passed( "T060602", False );
END T060602;

--* BEGIN Dec_Vax_V1_4
PRAGMA Suppress_All;
--* END
--* BEGIN TeleGen2_V3_15
-- No pragma to suppress checks found.
--* END
```

Source File: T060603.TST

```
-- T060603
--
-- The compiler shall issue a warning message to indicate static expressions
-- that will always raise a constraint exception at run-time.
--
-- Method:
--
-- Compile a procedure containing a declaration that assigns an out of range
-- value to a variable. Examine the compiler listing for a warning message.
--
--* COMPILE T060603 COMPILER_LISTING
--* EXECUTE T060603
WITH Result;
PROCEDURE T060603 IS
    SUBTYPE Small_Range IS Integer RANGE 0 .. 2;

    PROCEDURE Sub_Proc IS
        X : Small_Range := 3;
    BEGIN
        X := X - 1;
    END Sub_Proc;

BEGIN
    Sub_Proc;
    Result.Inconclusive( "T060603", "This statement should not be executed." );
EXCEPTION
    WHEN OTHERS => Result.Manual_Test( "T060603" );
END T060603;
```

Source File: T060701.TST

```
-- T060701
--
-- The compiler shall share code between multiple instantiations of generic
-- units that do not differ in their underlying machine representation.
--
-- Method:
--
-- Declare two enumeration types with two elements. Create a generic
-- procedure to swap two elements. Instantiate this generic for both
-- enumeration types. Use the OPTIMIZE_SPACE compiler parameter to make
-- sure the compiler will use the same code if possible. Look through
-- the assembly code to see if the code is shared.
--
--* COMPILE T060701 OPTIMIZE_SPACE ASSEMBLY_LISTING
--* EXECUTE T060701
WITH Result;
PROCEDURE T060701 IS

    TYPE A_Type IS ( A, B );
    TYPE B_Type IS ( B, A );

    A1 : A_Type := A;
    A2 : A_Type := B;
    B1 : B_Type := A;
    B2 : B_Type := B;

    GENERIC
        TYPE Item IS PRIVATE;
    PROCEDURE Exchange( X, Y : IN OUT Item );

    PROCEDURE Exchange( X, Y : IN OUT Item ) IS
        T : Item;
    BEGIN
        T := X; X := Y; Y := T;
    END Exchange;

    PROCEDURE A_Swap IS NEW Exchange( A_Type );
    PROCEDURE B_Swap IS NEW Exchange( B_Type );

BEGIN
    A_Swap( A1, A2 );
    B_Swap( B1, B2 );
    A_Swap( A2, A1 );
    B_Swap( B2, B1 );
    Result.Manual_Test( "T060701" );
END T060701;
```

Source File: T060702.TST

```
-- T060702
--
-- The compiler shall allow generic specifications and bodies to be compiled
-- in completely separate compilations.
--
-- Method:
--
-- Compile a generic specification and its body separately. The test
-- has passed if the compilations proceed without error.
--
--* COMPILE PART_A
GENERIC
  TYPE Item IS PRIVATE;
  PROCEDURE Switch( X, Y : IN OUT Item );
--* COMPILE PART_B
PROCEDURE Switch( X, Y : IN OUT Item ) IS
  T : Item;
BEGIN
  T := X;
  X := Y;
  Y := T;
END Switch;
--* COMPILE T060702
--* EXECUTE T060702
WITH Switch;
WITH Result;
PROCEDURE T060702 IS
  A : Character := 'A';
  B : Character := 'B';
  PROCEDURE Switch_It IS NEW Switch( Character );
BEGIN
  Switch_It( A, B );
  Result.Passed( "T060702", A = 'B' AND THEN B = 'A' );
END T060702;
```

Source File: T060703.TST

```
-- T060703
--
-- The compiler shall allow subunits of a generic unit to be separately
-- compiled.
--
-- Method:
--
-- Compile a generic specification and its body with a separate subunit.
-- The test has passed if the compilations proceed without error.
--
--* COMPILE PART_A
GENERIC
    TYPE Item IS PRIVATE;
PACKAGE Hidden IS
    FUNCTION Hello( I : Item ) RETURN Natural;
END Hidden;

PACKAGE BODY Hidden IS
    S : Item;
    FUNCTION Hello( I : Item ) RETURN Natural IS SEPARATE;
END Hidden;

--* COMPILE PART_B
SEPARATE( Hidden )
FUNCTION Hello( I : Item ) RETURN Natural IS
BEGIN
    S := I;
    RETURN 1;
END Hello;

--* COMPILE T060703
--* EXECUTE T060703
WITH Result;
WITH Hidden;
PROCEDURE T060703 IS
    PACKAGE New_Hidden IS NEW Hidden( Boolean );
BEGIN
    Result.Passed( "T060703", New_Hidden.Hello( True ) = 1 );
END T060703;
```

Source File: T060801.TST

```
-- T060801
--
-- The compiler shall provide the PRAGMA Interface to allow importing of
-- assembly language programs already assembled into the object code format
-- of the target computer. The machine language interface for procedure and
-- function parameters and function result types shall be documented.
--
-- Method:
--
-- Inspection of documentation.
--
--* COMPILE T060801
--* EXECUTE T060801
WITH Result;
PROCEDURE T060801 IS
BEGIN
    Result.Manual_Test( "T060801" );
END T060801;
```

Source File: T060802.TST

```
-- T060802
--
-- The compiler shall provide the PRAGMA Interface, or an equivalent
-- mechanism, to allow incorporation of subprogram bodies compiled from the
-- standard system or application languages of the target computer.
--
-- Method:
--
-- Inspection of documentation.
--
--* COMPILE T060802
--* EXECUTE T060802
WITH Result;
PROCEDURE T060802 IS
BEGIN
    Result.Manual_Test( "T060802" );
END T060802;
```


Source File: T060900.TST

```
-- T060900
--
-- The generic library subprograms UNCHECKED_DELOCATION and
-- UNCHECKED_CONVERSION shall be implemented with no restrictions except that
-- both objects in an unchecked conversion may be required to be of the same
-- size.
--
-- Method:
--
-- Test Unchecked_Conversion by transferring a value between three
-- different types of the same size.  Unchecked_Deallocation was
-- used on an access type. The test has passed if the compilation
-- and execution have completed without error.
--
--* COMPILE T060900
--* EXECUTE T060900
WITH Result;
WITH UNCHECKED_CONVERSION;
WITH UNCHECKED_DEALLOCATION;
PROCEDURE T060900 IS

    TYPE Record_1 IS RECORD
        S : String( 1 .. 8 );
    END RECORD;

    TYPE Array_1 IS ARRAY( 1 .. 2, 1 .. 4 ) OF Character;
    TYPE Array_2 IS ARRAY( 1 .. 4, 1 .. 2 ) OF Character;

    TYPE Access_1 IS ACCESS Record_1;

    Acc_1 : Access_1;
    Rec_1 : Record_1;
    Rec_2 : Record_1;
    Arr_1 : Array_1;
    Arr_2 : Array_2;

    PROCEDURE Free IS NEW
        UNCHECKED_DEALLOCATION( Record_1, Access_1 );

    FUNCTION Rec1_To_Arr1 IS NEW
        UNCHECKED_CONVERSION( Record_1, Array_1 );
    FUNCTION Arr1_To_Arr2 IS NEW
        UNCHECKED_CONVERSION( Array_1, Array_2 );
    FUNCTION Arr2_To_Rec1 IS NEW
        UNCHECKED_CONVERSION( Array_2, Record_1 );

BEGIN
    Rec_1.S := "12345678";
    Arr_1 := Rec1_To_Arr1( Rec_1 );
    Arr_2 := Arr1_To_Arr2( Arr_1 );
    Rec_2 := Arr2_To_Rec1( Arr_2 );
    Acc_1 := NEW Record_1;
    Free( Acc_1 );
    Result.Passed( "T060900", Rec_2.S = "12345678" AND THEN Acc_1 = NULL );
END T060900;
```

Source File: T061001.TST

```
-- T061001
--
-- An implementation shall provide packages to allow input and output of
-- FORTRAN-formatted text files for each target computer that supports
-- input/output.
--
-- Method:
--
-- Inspection of Documentation.
--
--* COMPILE T061001
--* EXECUTE T061001
WITH Result;
PROCEDURE T061001 IS
BEGIN
    Result.Manual_Test( "T061001" );
END T061001;
```

Source File: T061002.TST

```
-- T061002
--
-- Package SEQUENTIAL_IO and package DIRECT_IO shall be able to be instantiated
-- with unconstrained array types or with unconstrained record types which have
-- discriminants without default values.
--
-- Method:
--
-- Declare an unconstrained array type, and an unconstrained record type.
-- Instantiate Sequential_IO and Direct_IO for both of these. If the
-- compilation and execution succeed without error, the compiler passes.
--
--* COMPILE T061002
--* EXECUTE T061002
WITH Result;
WITH Direct_IO;
WITH Sequential_IO;
PROCEDURE T061002 IS

    --* BEGIN Dec_Vax_V1_4 TeleGen2_V3_15
    Arguments : CONSTANT String( 1 .. 15 ) := "RECORD;SIZE 128";
    --* END

    TYPE Vector IS ARRAY( Integer RANGE < > ) OF Integer;
    TYPE Square( Order : Positive ) IS RECORD
        Vec_1 : Vector( 1 .. Order );
        Vec_2 : Vector( 1 .. Order );
    END RECORD;

    FUNCTION Test_Vector_Direct_IO RETURN Boolean IS

        PACKAGE Vec_Dir_IO IS NEW Direct_IO( Vector );

        A_Vector : Vector( 1 .. 5 );
        File      : Vec_Dir_IO.File_Type;

        FUNCTION Perform_Test RETURN Boolean IS
        BEGIN
            Vec_Dir_IO.Create
                ( File, Vec_Dir_IO.Out_File, Result.Temp_Name, Arguments );
            A_Vector := ( 0, 1, 2, 3, 4 );
            Vec_Dir_IO.Write( File, A_Vector, 1 );
            Vec_Dir_IO.Reset( File, Vec_Dir_IO.In_File );
            Vec_Dir_IO.Read( File, A_Vector, 1 );
            Vec_Dir_IO.Delete( File );
            RETURN True;
        EXCEPTION
            WHEN OTHERS =>
                Vec_Dir_IO.Delete( File );
                RETURN False;
        END Perform_Test;

    BEGIN
        RETURN Perform_Test;
    EXCEPTION
        WHEN OTHERS => RETURN False;
    END Test_Vector_Direct_IO;

    FUNCTION Test_Square_Direct_IO RETURN Boolean IS

        PACKAGE Squ_Dir_IO IS NEW Direct_IO( Square );

        A_Square : Square( 5 );
        File      : Squ_Dir_IO.File_Type;

        FUNCTION Perform_Test RETURN Boolean IS
        BEGIN
            Squ_Dir_IO.Create
                ( File, Squ_Dir_IO.Out_File, Result.Temp_Name, Arguments );
            A_Square.Vec_1 := ( 0, 1, 2, 3, 4 );
            A_Square.Vec_2 := ( 4, 3, 2, 1, 0 );
            Squ_Dir_IO.Write( File, A_Square, 1 );
```

Source File: T061002.TST

```
Squ_Dir_IO.Reset( File, Squ_Dir_IO.In_File );
Squ_Dir_IO.Read( File, A_Square, 1 );
Squ_Dir_IO.Delete( File );
RETURN True;
EXCEPTION
  WHEN OTHERS =>
    Squ_Dir_IO.Delete( File );
    RETURN False;
END Perform_Test;

BEGIN
  RETURN Perform_Test;
EXCEPTION
  WHEN OTHERS => RETURN False;
END Test_Square_Direct_IO;

FUNCTION Test_Vector_Sequential_IO RETURN Boolean IS
  PACKAGE Vec_Seq_IO IS NEW Sequential_IO( Vector );
  A_Vector : Vector( 1 .. 5 );
  File     : Vec_Seq_IO.File_Type;

  FUNCTION Perform_Test RETURN Boolean IS
    BEGIN
      Vec_Seq_IO.Create
        ( File, Vec_Seq_IO.Out_File, Result.Temp_Name, Arguments );
      A_Vector := ( 0, 1, 2, 3, 4 );
      Vec_Seq_IO.Write( File, A_Vector );
      Vec_Seq_IO.Reset( File, Vec_Seq_IO.In_File );
      Vec_Seq_IO.Read( File, A_Vector );
      Vec_Seq_IO.Delete( File );
      RETURN True;
    EXCEPTION
      WHEN OTHERS =>
        Vec_Seq_IO.Delete( File );
        RETURN False;
    END Perform_Test;

  BEGIN
    RETURN Perform_Test;
  EXCEPTION
    WHEN OTHERS => RETURN False;
  END Test_Vector_Sequential_IO;

FUNCTION Test_Square_Sequential_IO RETURN Boolean IS
  PACKAGE Squ_Seq_IO IS NEW Sequential_IO( Square );
  A_Square : Square( 5 );
  File     : Squ_Seq_IO.File_Type;

  FUNCTION Perform_Test RETURN Boolean IS
    BEGIN
      Squ_Seq_IO.Create
        ( File, Squ_Seq_IO.Out_File, Result.Temp_Name, Arguments );
      A_Square.Vec_1 := ( 0, 1, 2, 3, 4 );
      A_Square.Vec_2 := ( 4, 3, 2, 1, 0 );
      Squ_Seq_IO.Write( File, A_Square );
      Squ_Seq_IO.Reset( File, Squ_Seq_IO.In_File );
      Squ_Seq_IO.Read( File, A_Square );
      Squ_Seq_IO.Delete( File );
      RETURN True;
    EXCEPTION
      WHEN OTHERS =>
        Squ_Seq_IO.Delete( File );
        RETURN False;
    END Perform_Test;

  BEGIN
    RETURN Perform_Test;
  EXCEPTION
    WHEN OTHERS => RETURN False;
  END Test_Square_Sequential_IO;
```

Source File: T061002.TST

```
        WHEN OTHERS => RETURN False;
    END Test_Square_Sequential_IO;

FUNCTION Test( Line : String; Pass : Boolean ) RETURN Natural IS
BEGIN
    CASE Pass IS
        WHEN True  => Result.Print( Line & " PASSED." ); RETURN 25;
        WHEN False => Result.Print( Line & " FAILED." ); RETURN 0;
    END CASE;
END Test;

BEGIN
    Result.Passed( "T061002",
        Test( "Sequential_IO array instantiation", Test_Vector_Sequential_IO ) +
        Test( "Sequential_IO record instantiation", Test_Square_Sequential_IO ) +
        Test( "Direct_IO array instantiation",      Test_Vector_Direct_IO ) +
        Test( "Direct_IO record instantiation",      Test_Square_Direct_IO ) );
END T061002;
```

Source File: T061003.TST

```
-- T061003
--
-- The compiler shall allow more than one internal file to be associated with
-- each external file for DIRECT_IO and SEQUENTIAL_IO for both reading and
-- writing.
--
-- Method:
--
-- Compile and execute a program which uses two different handles to refer
-- to the same file. For each IO, perform multiple reads and multiple
-- writes for a total of four tests. Each test will display either
-- success or failure.
--
--* COMPILE T061003
--* EXECUTE T061003
WITH Result;
WITH Direct_IO;
WITH Sequential_IO;
PROCEDURE T061003 IS
```

```
PACKAGE Seq_IO IS NEW Sequential_IO( Integer );
PACKAGE Dir_IO IS NEW Direct_IO( Integer );
```

```
FUNCTION Test_Direct_IO_Read RETURN Boolean IS
```

```
    Var      : Integer;
    Passed    : Boolean;
    File      : Dir_IO.File_Type;

    FUNCTION Perform_Test RETURN Boolean IS
        Is_OK : Boolean := True;
        File_1 : Dir_IO.File_Type;
        File_2 : Dir_IO.File_Type;
    BEGIN
        DECLARE
            BEGIN
                Dir_IO.Open( File_1, Dir_IO.In_File, Result.Temp_Name );
                Dir_IO.Open( File_2, Dir_IO.In_File, Result.Temp_Name );
            EXCEPTION
                WHEN OTHERS => Is_OK := False;
        END;
        IF Is_OK THEN
            DECLARE
                BEGIN
                    Dir_IO.Read( File_1, Var, 1 );
                    Dir_IO.Read( File_2, Var, 2 );
                EXCEPTION
                    WHEN OTHERS => Is_OK := False;
            END;
            END IF;
            Dir_IO.Close( File_1 );
            Dir_IO.Close( File_2 );
            RETURN Is_OK;
        EXCEPTION
            WHEN OTHERS => RETURN False;
    END Perform_Test;
```

```
BEGIN
    Dir_IO.Create( File, Dir_IO.Out_File, Result.Temp_Name );
    Dir_IO.Write( File, 1, 1 );
    Dir_IO.Write( File, 2, 2 );
    Dir_IO.Close( File );
    Passed := Perform_Test;
    Dir_IO.Open( File, Dir_IO.Out_File, Result.Temp_Name );
    Dir_IO.Delete( File );
    RETURN Passed;
EXCEPTION
    WHEN OTHERS => RETURN False;
END Test_Direct_IO_Read;
```

```
FUNCTION Test_Direct_IO_Write RETURN Boolean IS
```

```

Var      : Integer;
Passed   : Boolean;
File     : Dir_IO.File_Type;

FUNCTION Perform_Test RETURN Boolean IS
  Is_OK   : Boolean := True;
  File_1  : Dir_IO.File_Type;
  File_2  : Dir_IO.File_Type;
BEGIN
  DECLARE
  BEGIN
    Dir_IO.Open( File_1, Dir_IO.Out_File, Result.Temp_Name );
    Dir_IO.Open( File_2, Dir_IO.Out_File, Result.Temp_Name );
  EXCEPTION
    WHEN OTHERS => Is_OK := False;
  END;
  IF Is_OK THEN
    DECLARE
    BEGIN
      Dir_IO.Write( File_1, 1, 1 );
      Dir_IO.Write( File_2, 2, 2 );
    EXCEPTION
      WHEN OTHERS => Is_OK := False;
    END;
  END IF;
  Dir_IO.Close( File_1 );
  Dir_IO.Close( File_2 );
  RETURN Is_OK;
EXCEPTION
  WHEN OTHERS => RETURN False;
END Perform_Test;

BEGIN
  Dir_IO.Create( File, Dir_IO.Out_File, Result.Temp_Name );
  Dir_IO.Write( File, 1, 1 );
  Dir_IO.Write( File, 2, 2 );
  Dir_IO.Close( File );
  Passed := Perform_Test;
  Dir_IO.Open( File, Dir_IO.Out_File, Result.Temp_Name );
  Dir_IO.Delete( File );
  RETURN Passed;
EXCEPTION
  WHEN OTHERS => RETURN False;
END Test_Direct_IO_Write;

FUNCTION Test_Sequential_IO_Read RETURN Boolean IS

  Var      : Integer;
  Passed   : Boolean;
  File     : Seq_IO.File_Type;

  FUNCTION Perform_Test RETURN Boolean IS
    Is_OK   : Boolean := True;
    File_1  : Seq_IO.File_Type;
    File_2  : Seq_IO.File_Type;
  BEGIN
    DECLARE
    BEGIN
      Seq_IO.Open( File_1, Seq_IO.In_File, Result.Temp_Name );
      Seq_IO.Open( File_2, Seq_IO.In_File, Result.Temp_Name );
    EXCEPTION
      WHEN OTHERS => Is_OK := False;
    END;
    IF Is_OK THEN
      DECLARE
      BEGIN
        Seq_IO.Read( File_1, Var );
        Seq_IO.Read( File_2, Var );
      EXCEPTION
        WHEN OTHERS => Is_OK := False;
      END;
    END IF;
  END Perform_Test;

```

Source File: T061003.TST

```
        Seq_IO.Close( File_1 );
        Seq_IO.Close( File_2 );
        RETURN Is_OK;
    EXCEPTION
        WHEN OTHERS => RETURN False;
    END Perform_Test;

BEGIN
    Seq_IO.Create( File, Seq_IO.Out_File, Result.Temp_Name );
    Seq_IO.Write( File, 1 );
    Seq_IO.Write( File, 2 );
    Seq_IO.Close( File );
    Passed := Perform_Test;
    Seq_IO.Open( File, Seq_IO.Out_File, Result.Temp_Name );
    Seq_IO.Delete( File );
    RETURN Passed;
EXCEPTION
    WHEN OTHERS => RETURN False;
END Test_Sequential_IO_Read;

FUNCTION Test_Sequential_IO_Write RETURN Boolean IS
    Var      : Integer;
    Passed   : Boolean;
    File     : Seq_IO.File_Type;

    FUNCTION Perform_Test RETURN Boolean IS
        Is_OK : Boolean := True;
        File_1 : Seq_IO.File_Type;
        File_2 : Seq_IO.File_Type;
    BEGIN
        DECLARE
            BEGIN
                Seq_IO.Open( File_1, Seq_IO.Out_File, Result.Temp_Name );
                Seq_IO.Open( File_2, Seq_IO.Out_File, Result.Temp_Name );
            EXCEPTION
                WHEN OTHERS => Is_OK := False;
            END;
            IF Is_OK THEN
                DECLARE
                    BEGIN
                        Seq_IO.Write( File_1, 1 );
                        Seq_IO.Write( File_2, 2 );
                    EXCEPTION
                        WHEN OTHERS => Is_OK := False;
                    END;
                END IF;
                Seq_IO.Close( File_1 );
                Seq_IO.Close( File_2 );
                RETURN Is_OK;
            EXCEPTION
                WHEN OTHERS => RETURN False;
            END Perform_Test;

    BEGIN
        Seq_IO.Create( File, Seq_IO.Out_File, Result.Temp_Name );
        Seq_IO.Write( File, 1 );
        Seq_IO.Write( File, 2 );
        Seq_IO.Close( File );
        Passed := Perform_Test;
        Seq_IO.Open( File, Seq_IO.Out_File, Result.Temp_Name );
        Seq_IO.Delete( File );
        RETURN Passed;
    EXCEPTION
        WHEN OTHERS => RETURN False;
    END Test_Sequential_IO_Write;

FUNCTION Test( Line : String; Pass : Boolean ) RETURN Natural IS
    BEGIN
        CASE Pass IS
            WHEN True => Result.Print( Line & " PASSED." ); RETURN 25;
            WHEN False => Result.Print( Line & " FAILED." ); RETURN 0;
```


Source File: T061003.TST

END CASE;
END Test;

BEGIN

Result.Passed("T061003",
Test("Direct Multiple Read", Test_Direct_IO_Read) +
Test("Direct Multiple Write", Test_Direct_IO_Write) +
Test("Sequential Multiple Read", Test_Sequential_IO_Read) +
Test("Sequential Multiple Write", Test_Sequential_IO_Write));
END T061003;

Source File: T061004.TST

```
-- T061004
--
-- The compiler shall allow an external file associated with more than one
-- internal file to be deleted.
--
-- Method:
--
-- Compile a program containing two internal file descriptors pointing to
-- the same external file. The program then deletes the external file.
-- The compiler will have passed the test if no errors are generated.
--
--* COMPILE T061004
--* EXECUTE T061004
WITH Result;
WITH Text_IO;
PROCEDURE T061004 IS

    File : Text_IO.File_Type;

    PROCEDURE Close_No_Error( Old_File : IN OUT Text_IO.File_Type ) IS
    BEGIN
        Text_IO.Close( Old_File );
    EXCEPTION
        WHEN OTHERS => NULL;
    END Close_No_Error;

    PROCEDURE Delete_No_Error( File_Name : String ) IS
        File : Text_IO.File_Type;
    BEGIN
        Text_IO.Open( File, Text_IO.In_File, File_Name );
        Text_IO.Delete( File );
    EXCEPTION
        WHEN OTHERS => NULL;
    END Delete_No_Error;

    FUNCTION Perform_Test RETURN Boolean IS
        File_1 : Text_IO.File_Type;
        File_2 : Text_IO.File_Type;
    BEGIN
        Text_IO.Open( File_1, Text_IO.In_File, Result.Temp_Name );
        Text_IO.Open( File_2, Text_IO.In_File, Result.Temp_Name );
        Text_IO.Delete( File_1 );
        Text_IO.Close( File_2 );
        RETURN True;
    EXCEPTION
        WHEN OTHERS =>
            Close_No_Error( File_1 );
            Close_No_Error( File_2 );
            Delete_No_Error( Result.Temp_Name );
            RETURN False;
    END Perform_Test;

BEGIN
    Text_IO.Create( File, Text_IO.Out_File, Result.Temp_Name );
    Text_IO.Put_Line( File, "String 1" );
    Text_IO.Put_Line( File, "String 2" );
    Text_IO.Close( File );
    Result.Passed( "T061004", Perform_Test );
END T061004;
```

Source File: T061101.TST

```
-- T061101
--
-- The named numbers defined in package SYSTEM shall not limit or restrict the
-- inherent capabilities of the target computer hardware or operating system.
--
-- Method:
--
-- In the rationale for the test, specific requirements are given.
-- The testing is not programmable, manual checks need to be made.
--
-- Storage_Unit >= # bits in smallest addressable storage unit
-- Memory_Size >= maximum # of addressable memory units
-- Min_Int <= smallest integer available
-- Max_Int >= largest integer available
-- Max_Digits >= # sig digits in mantissa of largest floating point
-- Max_Mantissa >= # binary digits in mantissa of fixed-point
-- Fine_Delta = smallest delta allowed for fixed point number types
-- Tick = smallest timing increment provided by target computer
--
-- More detail can be found on these requirements in the report
-- "The Definition of a Production Quality Compiler"
--
--* COMPILE T061101
--* EXECUTE T061101
WITH Result;
WITH System;
PROCEDURE T061101 IS

    Size : CONSTANT Natural := Integer'IMAGE( Integer'LAST )'LENGTH + 4;

    PROCEDURE Show( Line : String; Int : Integer ) IS
    BEGIN
        Result.Print( Line & Result.Image( Int, Size ) );
    END Show;

    PROCEDURE Show( Line : String; Flt : Float ) IS
    BEGIN
        Result.Print( Line & Result.Image( Flt, Size, 4, 3 ) );
    END Show;

BEGIN
    Show( "Storage_Unit = ", System.Storage_Unit );
    Show( "Memory_Size = ", System.Memory_Size );
    Show( "Min_Int = ", System.Min_Int );
    Show( "Max_Int = ", System.Max_Int );
    Show( "Max_Digits = ", System.Max_Digits );
    Show( "Max_Mantissa = ", System.Max_Mantissa );
    Show( "Fine_Delta = ", System.Fine_Delta );
    Show( "Tick = ", System.Tick );
    Result.Manual_Test( "T061101" );
END T061101;
```

Source File: T061102.TST

```
-- T061102
--
-- The enumeration type NAME defined in PACKAGE SYSTEM shall have values
-- for all target computers for which the compiler generated code.
--
-- Method:
--
-- Print all the values in System.Name and compare to the names given
-- in the compiler documentation.
--
--* COMPILE T061102
--* EXECUTE T061102
WITH Result;
WITH System;
PROCEDURE T061102 IS
BEGIN
    FOR Compiler IN System.Name LOOP
        Result.Print( System.Name'IMAGE( Compiler ) );
    END LOOP;
    Result.Manual_Test( "T061102" );
END T061102;
```

Source File: T061201.TST

```
-- T061201
--
-- An implementation shall provide the predefined PRAGMA Controlled.
--
-- Method:
--
-- Include the pragma in the code below.  Examine the code listing to
-- make sure no warnings have occurred.
--
--* COMPILE T061201 COMPILER_LISTING
--* EXECUTE T061201
WITH Result;
PROCEDURE T061201 IS
  TYPE Cell;
  TYPE Link IS ACCESS Cell;
  TYPE Cell IS RECORD
    Value : Integer;
    Succ  : Link;
    Pred  : Link;
  END RECORD;

  PRAGMA Controlled( Link );

  Head : Link := NEW Cell'( 0, NULL, NULL );
BEGIN
  Head.Value := 10;
  Result.Manual_Test( "T061201" );
END T061201;
```

Source File: T061202.TST

```
-- T061202
--
-- An implementation shall provide the predefined PRAGMA Elaborate.
--
-- Method:
--
-- Include the pragma in the code below.  Examine the code listing to
-- make sure no warnings have occurred.
--
--* COMPILE T061202 COMPILER_LISTING
--* EXECUTE T061202
WITH Result;
PRAGMA Elaborate( Result );
PROCEDURE T061202 IS
BEGIN
    Result.Manual_Test( "T061202" );
END T061202;
```

Source File: T061203.TST

```
-- T061203
--
-- An implementation shall provide the predefined PRAGMA List.
--
-- Method:
--
-- The pragma is included in the code below. The listing should be
-- compared against the output to show the hidden line. If the line
-- does not appear in the listing but it does in the output, then the
-- test has passed.
--
--* COMPILE T061203 COMPILER_LISTING
--* EXECUTE T061203
WITH Result;
PROCEDURE T061203 IS
BEGIN
    PRAGMA List( Off );
    -- This line should not be printed in listing
    -- This line should not be printed in listing
    -- This line should not be printed in listing
    -- This line should not be printed in listing
    PRAGMA List( On );
    -- This line should appear everywhere
    -- This line should appear everywhere
    -- This line should appear everywhere
    -- This line should appear everywhere
    Result.Manual_Test( "T061203" );
END T061203;
```

Source File: T061204.TST

```
-- T061204
--
-- An implementation shall provide the predefined pragma Memory_Size.
--
-- Method:
--
-- The pragma is included before the start of a compilation unit.
-- The test has passed if the compilation and execution succeeds
-- without warning and the printed memory size is as expected.
--
--* COMPILE T061204
--* EXECUTE T061204
--! EQUATE Size IS 32768
--! LOOP 1 START Size [1]
PRAGMA Memory_Size( [1] );
--! END [1]
WITH Result;
WITH System;
PROCEDURE T061204 IS
BEGIN
    --! LOOP 1 START Size [1]
    Result.Print
        ( "Memory Size Set to: [1] is: " & Integer'IMAGE( System.Memory_Size ) );
    Result.Passed( "T061204", [1] = System.Memory_Size );
    --! END [1]
END T061204;
```


Source File: T061205.TST

```
-- T061205
--
-- An implementation shall provide the predefined pragma OPTIMIZE.
--
-- Method:
--
-- The pragma is used with both the Space and Time options.
-- Examine the code listing to make sure no warnings have occurred.
--
--* COMPILE T061205 COMPILER_LISTING
--* EXECUTE T061205
WITH Result;
PROCEDURE T061205 IS

    Global_Number : Integer;

    PROCEDURE Test_Space IS
        PRAGMA Optimize( Space );
    BEGIN
        Global_Number := 20;
    END Test_Space;

    PROCEDURE Test_Time IS
        PRAGMA Optimize( Time );
    BEGIN
        Global_Number := 40;
    END Test_Time;

BEGIN
    Test_Space;
    Test_Time;
    Result.Manual_Test( "T061205" );
END T061205;
```

Source File: T061206.TST

```
-- T061206
--
-- An implementation shall provide the predefined pragma PAGE.
--
-- Method:
--
-- The pragma is included in the code below.  If the compiler listing
-- shows a new page at the point of the pragma, the test has passed.
--
--* COMPILE T061206 COMPILER_LISTING
--* EXECUTE T061206
WITH Result;
PROCEDURE T061206 IS
  -- Before Page
  -- Before Page
  -- Before Page
  PRAGMA Page;
  -- After Page
  -- After Page
  -- After Page
BEGIN
  Result.Manual_Test( "T061206" );
END T061206;
```

Source File: T061207.TST

```
-- T061207
--
-- An implementation shall provide the predefined pragma Storage_Unit.
--
-- Method:
--
-- The pragma is included before the start of a compilation unit.
-- The test has passed if the compilation and execution succeeds
-- without warning and the printed storage unit is as expected.
--
--* COMPILE T061207
--* EXECUTE T061207
--! EQUATE Size IS 16
--! LOOP 1 START Size [1]
PRAGMA Storage_Unit( [1] );
--! END [1]
WITH Result;
WITH System;
PROCEDURE T061207 IS
BEGIN
    --! LOOP 1 START Size [1]
    Result.Print
        ( "Storage Unit Set to: [1] is: " & Integer'IMAGE(System.Storage_Unit) );
    Result.Passed( "T061207", [1] = System.Storage_Unit );
    --! END [1]
END T061207;
```

Source File: T061208.TST

```
-- T061208
--
-- An implementation shall provide the predefined pragma System_Name.
--
-- Method:
--
-- The pragma is included before the start of a compilation unit.
-- The test has passed if the compilation and execution succeeds
-- without warning and the printed system name is as specified.
-- If there is only one name in the enumeration type System.Name
-- then this test is not applicable as nothing is being tested.
--
-- Test T061102 may be used to find the allowable enumeration values
-- to be used for this test. Place a name which is not the default
-- in the test below.
--
--* COMPILE T061208
--* EXECUTE T061208
--* BEGIN Dec_Vax_V1_4
PRAGMA System_Name( VAX_VMS );
--* END
--* BEGIN TeleGen2_v3_15
PRAGMA System_Name( TELEGEN2 );
--* END
WITH Result;
WITH System;
PROCEDURE T061208 IS

    --* BEGIN Dec_Vax_V1_4
    Current_Name : String( 1 .. 7 ) := "VAX_VMS";
    --* END
    --* BEGIN TeleGen2_V3_15
    Current_Name : String( 1 .. 8 ) := "TELEGEN2";
    --* END

    PROCEDURE Check_Name
    ( Expected : String;
      Actual   : String;
      Elements : Natural ) IS
    BEGIN
        IF Elements <= 1 THEN
            Result.Print( "NOTE: Only one value in System.Name" );
        END IF;
        Result.Print( "PRAGMA System_Name( " & Expected & " )" );
        Result.Print( "Observed Name: " & Actual & "." );
        Result.Passed( "T061208", Expected = Actual );
    END Check_Name;

BEGIN
    Check_Name
    ( Current_Name,
      System.Name'IMAGE( System.System_Name ),
      System.Name'POS( System.Name'LAST ) -
      System.Name'POS( System.Name'FIRST ) + 1 );
EXCEPTION
    WHEN OTHERS => Result.Passed( "T061208", False );
END T061208;
--* NEW_LIBRARY
```

Source File: T070100.TST

-- T070100

--

-- The compiler shall be validated by an Ada Validation Facility established
-- and operated under the direction of the DOD Ada Joint Program Office in all
-- configurations necessary to meet the requirements of this document.

--

-- Method:

--

-- Inspection.

--

--* COMPILE T070100

--* EXECUTE T070100

WITH Result;

PROCEDURE T070100 IS

BEGIN

 Result.Manual_Test("T070100");

END T070100;

Source File: T070200.TST

```
-- T070200
--
-- The compiler shall be subjected to a minimum of 20 site-months of
-- independent evaluation and usage in a realistic production working
-- environment before release for production use.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T070200
--* EXECUTE T070200
WITH Result;
PROCEDURE T070200 IS
BEGIN
    Result.Manual_Test( "T070200" );
END T070200;
```

Source File: T070300.TST

```
-- T070300
--
-- Provisions for on-going problem correction of the compiler shall be provided.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T070300
--* EXECUTE T070300
WITH Result;
PROCEDURE T070300 IS
BEGIN
    Result.Manual_Test( "T070300" );
END T070300;
```

Source File: T070400.TST

```
-- T070400
--
-- The maintaining organization shall provide configuration management for
-- the compiler, including maintenance of an up-to-date data base of compiler
-- errors showing the nature and status of each error.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T070400
--* EXECUTE T070400
WITH Result;
PROCEDURE T070400 IS
BEGIN
    Result.Manual_Test( "T070400" );
END T070400;
```


Source File: T070500.TST

```
-- T070500
--
-- The production quality compiler should exhibit an error rate of no more
-- than 1 verified new error for each 250,000 new lines of Ada compiled. This
-- rate shall decrease over time as the compiler matures.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T070500
--* EXECUTE T070500
WITH Result;
PROCEDURE T070500 IS
BEGIN
    Result.Manual_Test( "T070500" );
END T070500;
```

Source File: T080100.TST

```
-- T080100
--
-- The vendor shall provide a copy of the most recent version of the official
-- validation summary report prepared by the Ada Validation Organization that
-- validated the compiler. This report shall include both CPU and elapsed
-- times required to run the ACVC tests.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T080100
--* EXECUTE T080100
WITH Result;
PROCEDURE T080100 IS
BEGIN
    Result.Manual_Test( "T080100" );
END T080100;
```

Source File: T080200.TST

```
-- T080200
--
-- The compiler vendor shall supply a copy of the Ada Language Reference Manual
-- (ARM) (ANSI/MIL-STD 1815A) that includes implementation-specific details of
-- the compiler where applicable.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T080200
--* EXECUTE T080200
WITH Result;
PROCEDURE T080200 IS
BEGIN
    Result.Manual_Test( "T080200" );
END T080200;
```

Source File: T080300.TST

```
-- T080300
--
-- The vendor shall provide a User's Manual that describes how to use the
-- compiler to develop Ada applications programs, including information on
-- how to run the compiler. It shall include all system-dependent forms
-- implemented in the compiler (i.e., machine-specific functions), methods
-- of selecting debug aids, compiler options and parameters, and a complete
-- list of error and warning messages provided by the compiler, with a
-- description of each. Message descriptions shall reference the relevant
-- section of the ARM. The manual shall include examples of the commands
-- used to invoke the compiler and linker/loader system with various
-- combinations of compiler and linker options, respectively.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T080300
--* EXECUTE T080300
WITH Result;
PROCEDURE T080300 IS
BEGIN
    Result.Manual_Test( "T080300" );
END T080300;
```

Source File: T080400.TST

```
-- T080400
--
-- The vendor shall provide a Run-time System Manual for each target computer.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T080400
--* EXECUTE T080400
WITH Result;
PROCEDURE T080400 IS
BEGIN
    Result.Manual_Test( "T080400" );
END T080400;
```

Source File: T080500.TST

```
-- T080500
--
-- The vendor shall provide a Version Description Document for each compiler
-- configuration.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T080500
--* EXECUTE T080500
WITH Result;
PROCEDURE T080500 IS
BEGIN
    Result.Manual_Test( "T080500" );
END T080500;
```

Source File: T080600.TST

-- T080600

--

-- The vendor shall provide a detailed Installation Manual and all the
-- necessary software materials for installing each host configuration of
-- the Ada compiler, including several sample Ada programs with correct output.

--

-- Method:

--

-- Inspection.

--

--* COMPILE T080600

--* EXECUTE T080600

WITH Result;

PROCEDURE T080600 IS

BEGIN

 Result.Manual_Test("T080600");

END T080600;

Source File: T080700.TST

```
-- T080700
--
-- The vendor shall provide a Maintenance Manual which presents the methods to
-- be used in the general maintenance of all parts of the compiler. All major
-- data structures, such as the symbol table and the intermediate language,
-- shall be fully described. All debugging aids that have been inserted into
-- the compiler shall be described and their use fully stated. If the compiler
-- has a special "maintenance mode" of operation to assist in pinpointing
-- errors, this shall be fully described.
--
-- Method:
--
-- Inspection.
--
--* COMPILE T080700
--* EXECUTE T080700
WITH Result;
PROCEDURE T080700 IS
BEGIN
    Result.Manual_Test( "T080700" );
END T080700;
```


Source File: T080800.TST

-- T080800

--
-- The vendor shall provide a Software Product Specification for the compiler
-- in accordance the DOD-STD-2167 and Data Item Description DI-MCCR-80029.

-- Method:

-- Inspection.

--* COMPILE T080800

--* EXECUTE T080800

WITH Result;

PROCEDURE T080800 IS

BEGIN

 Result.Manual_Test("T080800");

END T080800;

--* NEW_LIBRARY