USER'S MANUAL FOR THE PROTOTYPE Ada* COMPILER EVALUATION CAPABILITY (ACEC) VERSION 1

Audrey A. Hook
Gregory A. Riccardi
Michael Vilot
Stephen Welke

October 1985

Prepared for
Office of the Under Secretary of Defense for Research and Engineering

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

*Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).
The purpose of the Prototype Ada Compiler Evaluation Capability (ACEC) is to provide users with 1) an organized suite of compiler performance tests, and 2) support software for executing these tests and collecting performance statistics. These performance tests were collected by the Ada Evaluation and Validation (E&V) Team from several sources. The test programs, which have been in the public domain for some time, have been organized as a test suite according to categories which are explained in Section II. The strategy for measuring test performance and obtaining differential statistics is described in Section III. Section IV describes the entire support software architecture, including machine-dependent modules. General instructions for executing the Prototype ACEC are provided in Section V.
USER'S MANUAL FOR THE PROTOTYPE Ada
COMPILER EVALUATION CAPABILITY (ACEC)
VERSION 1

Audrey A. Hook
Gregory A. Riccardi
Michael Vilot
Stephen Welke

October 1985

INSTITUTE FOR DEFENSE ANALYSES
Contract MDA 903 84 C 0031
Task T-5-328
ATTRIBUTION

The User's Manual was prepared by the IDA team who developed the Prototype ACEC. The members of this team were:

Audrey Hook, IDA
Max Robinson, IDA
Steve Welke, IDA
Jeff Clouse, IDA
Dr. Gregory Riccardi, Florida State University
Dr. Ugo Gagliardi, General Systems Group
Michael Vilot, General Systems Group

Others who made contributions to the development of the Prototype ACEC were:

Virginia Castor, AJPO
Jon Squire, Westinghouse Electric Corporation, who as chairman of the SIGADA Performance Issues Working Group (PIWG), obtained member volunteers who performed the Beta test for the Prototype ACEC and thereby contributed to the User's Manual. These volunteers were:

Robert Gable, Lear Siegler Instruments Division
Daniel Ehrenfried, Rational
### TABLE OF CONTENTS

**SECTION** | **PAGES**
---|---
I - INTRODUCTION | 1-2
II - TEST CATEGORIES AND ATTRIBUTES | 3-5
III - DATA COLLECTION AND EVALUATION | 6-7
IV - SOFTWARE ARCHITECTURE | 8-11
V - EXECUTION INSTRUCTIONS | 12-14

**APPENDIX**

A - TEST SUITE - LISTING OF TESTS
B - ADA PACKAGES FOR SUPPORT SOFTWARE
C - REPORT WRITER'S GUIDE
D - EXECUTION EXAMPLES
   DATA GENERAL MV10000
   VAX/VMS

**LIST OF FIGURES**

1 - Test Data Collection | 7
2 - Software Architecture | 9
3 - Compilation Order | 11
4 - Format of the Statistics Files | 13
SECTION I: INTRODUCTION

The purpose of the Prototype Ada Compiler Evaluation Capability (ACEC) is to provide users with 1) an organized suite of compiler performance tests, and 2) support software for executing these tests and collecting performance statistics. These performance tests were collected by the Ada Evaluation and Validation (E&V) team from several sources. The test programs, which have been in the public domain for some time, have been organized as a test suite according to categories which are explained in Section II (Appendix A is a listing of the names and descriptions of all the tests in the test suite). They have also been instrumented to provide execution statistics. The user can obtain differential execution statistics for the Ada language feature(s) used in a test by comparing different versions of the test. The strategy for measuring test performance and obtaining differential statistics is described in Section III.

The support software that is provided with the system consists of Ada packages (see Appendix B). This software includes an interface to a database of test attributes, an instrumentation package designed to collect execution statistics, and a report writer. All of these packages should run on any hardware host. However, to make the Prototype ACEC complete, there are host/target machine-dependent modules that a user must provide as part of the support software. These modules are required in order to collect the compilation and run-time statistics that are only available through the operating system and/or software monitors for that machine. The entire support software architecture, including the machine-dependent modules, is described in Section IV.

The intended users of this system will be programmers who are familiar with their Ada compilation system. They must know how to invoke the compiler and host/target dependent portions of the Prototype ACEC. A user who is familiar with the Ada Compiler Validation Capability (ACVC) should find that the Prototype ACEC is roughly equivalent in execution complexity to the ACVC; however, the number of tests to be executed by the
Prototype ACEC is an order of magnitude less than the number in the ACVC. General instructions for executing the Prototype ACEC are provided in Section V.

Several general concepts are useful for understanding the significance of the measurements that can be obtained from the Prototype ACEC. The current view of compiler evaluation tests is similar to the concept of benchmarks designed to demonstrate the performance characteristics of a computer system when it is used for processing a typical workload. Benchmarks are programs, or sets of programs, that are used to represent a real workload; therefore, they are a synthetic workload which may or may not accurately represent the future capacity and efficiency requirements for a specific application. Typically, benchmark programs provide general measurements of execution efficiency and are used to indicate the relative capabilities of different computer systems or alternative configurations. On the other hand, the Prototype ACEC allows the user to measure the effect of specific workloads on a particular computer system component, the compiler.

The design goal for the Prototype ACEC was to collect objective, quantifiable attributes of an Ada compiler/run-time combination that would allow an applications developer to evaluate the usefulness of a compiler for a future application. The usefulness of a compiler is a function of the language constructs that are most frequently used (i.e. required) by an application and the effect that they produce in demand for computer resources. These most frequently used language constructs are the "stress load" for a compiler which, in turn, may have the effect of a "bottleneck" in the computer system configuration. Since applications differences lead to remarkably different frequency distributions of language features, the Prototype ACEC was designed to allow a user to select tests for specific constructs and to obtain consistent measurements for the "costs" (e.g. time and space) associated with these constructs when they are used in various compiler/run-time combinations.

Therefore, the Prototype ACEC provides a user with two options for evaluating an Ada compiler. A user may select a set of tests which represents the frequency distribution of language constructs in a real application; or, the user may execute all tests to gain some insight regarding the language feature(s) which could be a stress load for a compiler/run-time combination, if these features were among the most frequently used. HOWEVER, Prototype ACEC MEASUREMENTS ARE ONLY AN INDICATION OF THE EFFECT PRODUCED BY AN ADA LANGUAGE FEATURE WHEN IT IS USED IN A PARTICULAR COMPILER/RUN-TIME COMBINATION. THESE MEASUREMENTS ARE NOT ABSOLUTE PERFORMANCE METRICS OF THE EFFICIENCY OF A PARTICULAR COMPILER ARCHITECTURE.
SECTION II: TEST CATEGORIES AND ATTRIBUTES

This section discusses the architecture categories that have been established for the Prototype ACEC test suite and the attributes of these tests which are available through the support software. The attributes of each test unit are keyed to the category and sub-category classification of that test.

A. ARCHITECTURE CATEGORIES

Test units have been organized into two major groups based upon the information that the test unit will provide to the user. The first group of tests will provide information about language features that must be present in a compiler if it is a full implementation of the ANSI/MIL-STD 1815A. Therefore, these tests are called "normative" since they will produce the lowest level of measurement statistics that can be collected to characterize the performance of a conforming compiler. The second group of tests will provide information about combinations of language constructs and/or compiler features that may be of interest to applications developers. These tests are called "optional." These two major categories, normative and optional, have been further sub-divided based upon the type of measurements that can be derived from the tests. A description of each category and sub-category is given below:

A.1 NORMATIVE

Normative tests will provide a means for determining the system cost for a particular language feature. The user should execute all normative tests to obtain a quantitative indication of the usefulness of a compiler. There are two types of tests in the normative category:

A.1.1 Sub-category: PERFORMANCE

Performance tests will collect speed and space attributes for various Ada language features.

A.1.2 Sub-category: CAPACITY

Capacity tests will indicate the limitations imposed by the compiler and the run-time system on applications developers (e.g. levels of recursion, size of stack, etc.). Note that these tests may overlap with the ACVC, class D tests.
A.2 OPTIONAL

Optional tests may be selected by a user to represent an applications profile consisting of most frequently used language features. They have been included in the Prototype ACEC test suite to provide measurements which are consistent with normative tests.

A.2.1 Sub-category: FEATURES

Features tests provide measurements of optional language features (features which are not a required part of an Ada compiler). They also provide measurements of the effects of certain compiling options. Refer to Chapter 13 of the Ada LRM for examples of optional language features.

A.2.2 Sub-category: SPECIAL ALGORITHMS

Special algorithms tests are combinations of language constructs that are characteristic of synthetic benchmark programs. They include such widely known benchmarks as Whetstone and the Sieve of Eratosthenes.

B. TEST ATTRIBUTES

For each test there are attributes, including an architecture category as described above, that provide a user with descriptive information about that test. These attributes are stored in a database of test names, and are available via the report writer. They can also be used as search criteria for selecting a set of tests with a specified attribute (see Appendix C under the description of the "LIST" option). The attributes are listed below:

B.1 DESCRIPTION - A description of the test objective.

B.2 ARCHITECTURE CATEGORY - Code indicating membership in a major test category and a particular sub-category.

B.3 E&V CRITERIA - The evaluation criteria for this test, according to the list developed by the E&V Team. Only certain Efficiency criteria were found to be applicable.

B.4 LANGUAGE FEATURE - The language feature(s) being tested. The Prototype ACEC does not cover all of the Ada language features.
B.5 VERSION - Identification of whether this is the test or the control program, or the test with an optimization feature (see Section III).

B.6 STATISTICS TYPE - A description of the kind of statistics being collected (Compilation, Execution, or Both).
SECTION III: DATA COLLECTION AND EVALUATION

The individual Ada main procedures that comprise the Prototype ACEC test suite were adapted from test programs that have been in the public domain for some time.(*) Each of these procedures (a test unit) can provide compilation and execution statistics. The ability to collect data is determined by the facilities of the host/target environment and the Ada support software in the Prototype ACEC.

Figure 1 (see page 7) shows the data collection scheme for a single test. The figure indicates the location of clock measurements used to derive the various elapsed and cpu times. Also shown are the possible size measurements of objects created or used in compiling and executing a test. Note that only the instrumentation elapsed time is calculated by the provided software; all other statistics must be determined by some host/target dependent mechanisms. (See Section IV and Appendix B for a description of the support software functionality and interface specifications for the host/target environment.)

The data collected for an individual test is not of much use unless used in a differential strategy which filters out aspects of compilation and execution that are not caused by the specific feature under evaluation. To make this strategy possible, most tests have more than one version. The first version, the control, is structured like the other versions of the test, but does not contain the specific feature being tested. The other versions (there may be only one) of the test contain the language feature, with or without various optimizations. Once the data for at least two versions has been collected, a statement can be made about the feature under test.

For instance, suppose the control version of an integer addition test takes 0.15 seconds to execute, while version 2 of the same test, the test version, takes 0.25 seconds to execute. This would indicate that the execution overhead for having the integer additions was 0.10 seconds. If version 3, with the PRAGMA SUPPRESS, executed in 0.23 seconds, it could be said that 0.02 seconds were saved by use of this optimization. This same type of differential measurement strategy is used on the other time and size data to form conclusions about the time and space "costs" of the Ada language feature being tested.

(*) These original tests were written by many people with different styles and objectives. Minimal changes were made to these tests to adapt them to the test suite architecture. New tests were not written for this Prototype ACEC.
Figure 1

Test Data Collection

TEST UNIT

CLOCK COMPILATION CLOCK

GENERATED CODE

CLOCK

LOADING ELABORATION

EXECUTION IMAGE

CLOCK EXECUTION CLOCK

WORKING DATA

CLOCK

EXECUTION IMAGE SIZE

WORKING DATA SIZE

ELAPSED & CPU TIMES

ELAPSED & CPU TIMES

ELAPSED & CPU TIMES

COMPILATION STATISTICS

INSTRUMENTATION STATISTICS

RUN-TIME STATISTICS

REPORT WRITER

REPORT
SECTION IV: SOFTWARE ARCHITECTURE

The Prototype ACEC consists of several software components, some of which are provided in Ada source code, and some of which are necessarily implementation-dependent. Figure 2 (see page 9) is a representation of the software components and their interrelation.

In reference to Figure 2, the software components provided as Ada source code are the database package, the report writer, the instrumentation package, and the set of benchmark tests. The database package and the instrumentation package are not Ada main programs; rather, they are library packages. The database package is intended primarily for use by the report writer, but is available for use by the selector as well. In contrast, the instrumentation package is intended for use solely by the benchmark test programs themselves. The report writer and the benchmark tests are Ada main programs. The report writer, database package, and instrumentation package are described in more detail below. Instructions for executing the support software are provided in Section V.

The Database Package:
The database package provides basic facilities to retrieve information about a named benchmark test. The information available through the database package corresponds to the attributes described in Section II. Included are a one-line description of the test, the architecture category that the test falls under, the E&V Criteria that the test provides information about, the language feature(s) that the test examines, the version of the test that this particular program represents, and an indication of what statistics are expected to be collected for this test. This package is contained in the file DATABASE.ADA and depends on the library package(s) contained in the files LIST_PACKAGE.ADA and SCHEMA.ADA.

The Instrumentation Package:
The instrumentation package provides a simple start-stop timing facility based on the built-in clock capabilities of the Ada package CALENDAR. This facility determines the total time that has elapsed during execution. The instrumentation package also provides a package CPU_TIME.ADA for collecting the cpu time used during execution.
The CPU_TIME.ADA package that comes with the system contains a dummy function called CPU_CLOCK that returns the value 0.0. In order to collect meaningful results, the user will have to replace the dummy CPU_CLOCK with a new CPU_CLOCK function that accesses the system accounting information. However, this replacement is not required; the system will function properly with the dummy function.
Figure 2
Software Architecture

Legend
- CONTROL
- DATA FLOW
- IMPLEMENTATION-DEPENDENT MODULES
- INTERFACE FORMAT PROVIDED
- PROVIDED ADA SOURCE CODE
The instrumentation package is not intended for use outside of the test suite. It is contained in the file INSTRUMENT.ADA and depends on the library package(s) contained in the file IO_PACKAGE.ADA.

The Report Writer:

The report writer is the visible interface to the user. Its function is to report both collected statistics and desired database attributes. A detailed explanation of the report writer and how to use it is given in Appendix C. The report writer is contained in the files INQUIRY.ADA and REPORT_WRITER.ADA. In addition, it depends on the library package(s) contained in the files DATABASE.ADA, ATTRIBUTE.ADA, and IO_PACKAGE.ADA.

Figure 3 (see page 11) presents the compilation dependencies of the packages provided with the test suite.

The remaining software (the compiler, the executor and the selector) is necessarily machine-dependent, and must be provided by the user. In reference to Figure 2, the compiler is that set of software that compiles a single benchmark test and stores the collected compilation statistics in a file. Similarly, the executor is that set of software that executes a single test and stores the collected run-time statistics in a file. Interface specifications for formatting the collected data, if this data is available, are given in Section V. The selector is the overall "system manager" that chooses tests to be run and operates the compiler and executor. It makes the compilation and run-time statistics files available to the report writer. As mentioned previously, the selector may also be used to access the database directly.

Typically, the selector will consist of both software and human input. For our example on the MV10000 AOS/VS system, we chose the tests ourselves, and ran a test harness to manage the running of the compiler and executor. The test harness also reformatted the system accounting information to comply with the interface specifications required by the report writer. The term "test harness" refers to the portion of the selector that sends tests through the compiler and executor. Test harness examples are provided in Appendix D.

The executor is the series of system specific commands needed to execute a compiled program and collect the resulting statistics. In environments where host and target are different machines, this sequence of steps can be quite complex.
Figure 3
Compilation Order

LIST_PACKAGE.ADA

SCHEMA.ADA

DATABASE.ADA  ATTRIBUTE.ADA

INQUIRY.ADA

REPORT_WRITER.ADA

IO_PACKAGE.ADA

INSTRUMENT.ADA

CPU.ADA

Benchmark test(s)
(eg. ADDSA1.ADA)

* See Section IV under the description of "The Instrumentation Package"

Note: In order to compile a particular package, the package(s) pointing to it must be compiled first.
SECTION V: EXECUTION INSTRUCTIONS

This section of the User's Manual provides basic instructions for executing the Prototype ACEC system. These instructions will help the users to 1) identify the software modules that must be produced, 2) become aware of the various options within the system, and 3) gain an overall view of how this evaluation system works.

The system specific test harness mentioned in Section IV processes the benchmark tests and collects system resource statistics. On most systems, the test harness will consist of one or more operating system command files that will compile and execute each test program.

Different operating systems make various accounting information available to their users. This information can be an important part of the evaluation statistics gathered by the benchmarks. The statistics which are generated can be captured and stored in text files. These files, if available, will become input to the report writer. The format for each of these text files - the "TEST DATA" blocks in Figure 2 (see page 9) - is specified in Figure 4. NOTE: The user-defined filenames must be EXACTLY six characters long.

With these interface files well defined, the portable report writer software will be able to report as much data as can be collected by the host and target environments. On those systems where such accounting information is unavailable or difficult to obtain, the report writer will still function (although the lack of statistics will seriously degrade the usefulness of the Prototype ACEC system).

The Compilation Statistics:
Compilation statistics may be available from the compiler that is being evaluated, or they may be available from host system accounting information. In either case, there must be some host-dependent software to convert the statistics into the file format expected by the report writer. Figure 4a (see page 13) defines this format. These statistics must be appended onto a user-defined compilation file after each test is compiled. If parts of this information are not available, dummy values (such as 0) must be placed in the file in order for the report writer to use the file.

The Instrumentation Statistics:
Each test program utilizes an instrumentation package which reports the elapsed execution time of the test program. This data is collected from the function CLOCK of package CALENDAR.
Figure 4

Formats of the Statistics Files

For all three of the statistics files, each line of the file contains a sequence of fields separated by one or more spaces:

(a) The Compilation-Time Statistics File

1. Test name (6 characters)
2. Total elapsed time, in seconds *
3. Total cpu time, in seconds *
4. Object code size, in bytes **
5. Comments, a string of (up to) 120 characters

(b) The Instrumentation Statistics File

1. Test name (6 characters)
2. Total elapsed time, in seconds *
3. Total cpu time, in seconds *
4. Comments, a string of (up to) 120 characters

(c) The Run-Time Statistics File

1. Test name (6 characters)
2. Total elapsed time, in seconds *
3. Total cpu time, in seconds *
4. Execution image size, in bytes **
5. Working data size, in bytes **
6. Comments, a string of (up to) 120 characters

* The report writer currently displays these real numbers with an accuracy of 1/100th of a second
** Integer numbers
If available, CPU time can also be reported by modifying the package CPU_TIME which is provided in the support software. This package contains the function CPU_CLOCK which can be replaced to return the execution cpu time using some host/target dependent facility. (The CPU_CLOCK function that is provided returns the dummy value 0.0). This data is automatically written onto a file named "INSTR" at the end of the execution of the test program. NOTE: The contents of this file ("INSTR") MUST be appended onto a user-defined instrumentation file after each test program is run. The instrumentation file format is shown in Figure 4b (see page 13).

The Run-time Statistics:

These statistics are meant as a supplement to the instrumentation statistics. Run-time statistics are collected and properly formatted by host/target dependent mechanisms. Again, these statistics must be appended onto the user's run-time statistics file after each test program is run. Uncollectable data must be entered as dummy values in the run-time file. The format of this file is given in Figure 4c (see page 15).

Performing the Evaluation:

To start the evaluation process, the Ada code for the support software must be compiled onto an Ada library. Figure 3 (see page 11) gives the compilation order. Next, the user may develop the command files (the test harness) to help execute the ACEC system. Once the full test harness is ready, tests can be compiled and executed with all the generated statistics being captured in text files. After any number of test programs have been processed, the report writer should be executed with the collected data files as input. The report writer and the test harness can be re-run any number of times and in any order to produce the final evaluation report(s).
APPENDIX A: TEST SUITE - LISTING OF TESTS

This is a list of the benchmark tests that are currently in the database. Included are the test names and descriptions. The test names in the database are the same as the file names containing the Ada main programs.

Each test name has three parts. The first four characters describe the test characteristic. Where possible, these four characters are organized into a mnemonic (i.e. SIEV = Sieve of Eratosthenes). Otherwise, an acronym has been constructed (i.e. BRUA = Block Reference to an Uplevel variable, Access type). The fifth character is a letter that represents a difference in the number of occurrences of the test characteristic. For example, LAVRA1 performs one local array variable reference, while LAVRB1 performs ten references. The sixth character is the version number. Version 1 is the control version and version 2 is the test version. The other versions (3, 4, etc.) test for the effects of certain compiler features (i.e. PRAGMAs).

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADSA1</td>
<td>10_000 floating pt. Additions (control)</td>
<td></td>
</tr>
<tr>
<td>ADSA2</td>
<td>10_000 floating pt. Additions (test)</td>
<td></td>
</tr>
<tr>
<td>ADSA3</td>
<td>10_000 floating pt. Additions (pragma suppress)</td>
<td></td>
</tr>
<tr>
<td>AKERA2</td>
<td>Ackermann function (test)</td>
<td></td>
</tr>
<tr>
<td>AKERA3</td>
<td>Ackermann function (pragma suppress)</td>
<td></td>
</tr>
<tr>
<td>AOCEA1</td>
<td>Arith. Optimization, Const. Elim. (control)</td>
<td></td>
</tr>
<tr>
<td>AOCEA2</td>
<td>Arith. Optimization, Const. Elim. (test)</td>
<td></td>
</tr>
<tr>
<td>AOIEA1</td>
<td>Arith. Optimization, Invariant Elim. (control)</td>
<td></td>
</tr>
<tr>
<td>AOIEA2</td>
<td>Arith. Optimization, Invariant Elim. (test)</td>
<td></td>
</tr>
<tr>
<td>ASSIA2</td>
<td>500 ASSIGNMENT STMTS (1 PER LINE)</td>
<td></td>
</tr>
<tr>
<td>ASSIA3</td>
<td>500 ASSIGNMENT STMTS (5 PER LINE)</td>
<td></td>
</tr>
<tr>
<td>ASSIA4</td>
<td>500 ASSIGNMENT STMTS INTERJECTED WITH COMMENTS</td>
<td></td>
</tr>
<tr>
<td>ASSIA5</td>
<td>500 ASSIGNMENT STMTS PRECEDED BY 500 COMMENTS</td>
<td></td>
</tr>
<tr>
<td>ASSIB2</td>
<td>1000 ASSIGNMENT STMTS INTERJECTED WITH COMMENTS</td>
<td></td>
</tr>
<tr>
<td>BALPA1</td>
<td>EVALUATES THE EFFICIENCY OF A SIMPLE LOOP STATEMENT (CONTROL)</td>
<td></td>
</tr>
<tr>
<td>BALPA2</td>
<td>THIS TEST EVALUATES THE EFFICIENCY OF A SIMPLE LOOP STATEMENT (TEST)</td>
<td></td>
</tr>
<tr>
<td>BLEMA2</td>
<td>65 EMBEDDED BLOCKS</td>
<td></td>
</tr>
<tr>
<td>BRUA1</td>
<td>Block Ref. to an Uplevel var., Access type (control)</td>
<td></td>
</tr>
<tr>
<td>BRUA2</td>
<td>Block Ref. to an Uplevel var., Access type (test)</td>
<td></td>
</tr>
<tr>
<td>BRUNA1</td>
<td>Block Ref. to an Uplevel var., Non-access type (control)</td>
<td></td>
</tr>
<tr>
<td>BRUNA2</td>
<td>Block Ref. to an Uplevel var., Non-access type (test)</td>
<td></td>
</tr>
<tr>
<td>BSRCA2</td>
<td>TEST BINARY SEARCH PKG AT EXTREME LIMITS OF ITS INDEX TYPE: LOWER</td>
<td></td>
</tr>
<tr>
<td>BSRCA3</td>
<td>TEST BINARY SEARCH PKG AT EXTREME LIMITS OF ITS INDEX TYPE: UPPER</td>
<td></td>
</tr>
<tr>
<td>C31PA2</td>
<td>CHECKS THAT 31 PARAMETERS CAN BE PASSED</td>
<td></td>
</tr>
<tr>
<td>CAPAA1</td>
<td>Constrained Array Param. Assoc. w/3 elements (control)</td>
<td></td>
</tr>
</tbody>
</table>
CAPAA2 Constrained Array Param. Assoc. w/3 elements (test)
CAPAB1 Constrained Array Param. Assoc. w/63 elements (control)
CAPAB2 Constrained Array Param. Assoc. w/63 elements (test)
CASEA2 CHECKS A CASE STATEMENT OF SIZE 256
CENTA2 CHECKS AN ENUMERATION TYPE OF 256 ELEMENTS
CENTB2 CHECKS ENUMERATION TYPES UP TO 2000 ELEMENTS
CHSSA1 Char. String Search (control)
CHSSA2 Char. String Search (test)
CHSSA3 Char. String Search (pragma suppress)
CSSTA1 Case Statement Binary Test (control)
CSSTA2 Case Statement Binary Test (test)
CSCTA1 Case Statement Cluster Test (control)
CSCTA2 Case Statement Cluster Test (test)
CSDTA1 Case Statement Dense Test (control)
CSDTA2 Case Statement Dense Test (test)
CSSTA1 Case Statement Exhaustive Test (control)
CSSTA2 Case Statement Exhaustive Test (test)
CSSTTA1 Case Statement Sparse Test w/range 1..5 (control)
CSSTTA2 Case Statement Sparse Test w/range 1..5 (test)
CSSTTB1 Case Statement Sparse Test w/range 1..20 (control)
CSSTTB2 Case Statement Sparse Test w/range 1..20 (test)
CSSTC1 Case Statement Sparse Test w/range 1..50 (control)
CSSTC2 Case Statement Sparse Test w/range 1..50 (test)
CSSTD1 Case Statement Sparse Test w/range 1..500 (control)
CSSTD2 Case Statement Sparse Test w/range 1..500 (test)
CSSTE1 Case Statement Sparse Test w/range 1..5000 (control)
CSSTE2 Case Statement Sparse Test w/range 1..5000 (test)
DRPCA1 Direct Recursive Procedure Call (control)
DRPCA2 Direct Recursive Procedure Call (test)
FL1TA1 EFFICIENCY OF LOOP STMT, FOR, LOOP PARAM USED IN LOOP BODY (CONTROL)
FL1UA2 EFFICIENCY OF LOOP STMT, FOR, LOOP PARAM USED IN LOOP BODY (CONTROL)
FACTA1 RECURSIVE FACTORIAL FUNCTION (CONTROL)
FACTA2 RECURSIVE FACTORIAL FUNCTION
FL2RA1 EFFICIENCY OF A FOR LOOP STMT, REVERSE, 2 ITERATIONS (CONTROL)
FL2RA2 EFFICIENCY OF A LOOP STMT USING FORR, REVERSE, 2 ITERATIONS
FLP1A1 EVALUATES THE EFFICIENCY OF A LOOP STMT USING FOR, 1 ITER. (CONTROL)
FLP1A2 EVALUATES THE EFFICIENCY OF A LOOP STMT USING FOR, 1 ITERATION
FLP2A1 EVALUATES THE EFFICIENCY OF A LOOP STMT USING FOR, 2 ITER. (CONTROL)
FLP2A2 EVALUATES THE EFFICIENCY OF A LOOP STMT USING FOR, 2 ITERATIONS
FPAAA1 Formal in/out Param. Assoc. w/1 param., Access type (control)
FPAAA2 Formal in/out Param. Assoc. w/1 param., Access type (test)
FPAAAB1 Formal in/out Param. Assoc. w/2 param., Access type (control)
FPAAAB2 Formal in/out Param. Assoc. w/2 param., Access type (test)
FPAAC1 Formal in/out Param. Assoc. w/5 param., Access type (control)
FPAAC2 Formal in/out Param. Assoc. w/5 param., Access type (test)
FPAAD1 Formal in/out Param. Assoc. w/10 param., Access type (control)
FPAAD2 Formal in/out Param. Assoc. w/10 param., Access type (test)
FPANA1 Formal in/out Param. Assoc. w/1 param., Non-access type (control)
FPANA2 Formal in/out Param. Assoc. w/1 param., Non-access type (test)
FPANB1 Formal in/out Param. Assoc. w/2 param., Non-access type (control)
FPANB2 Formal in/out Param. Assoc. w/2 param., Non-access type (test)
FPANC1 Formal in/out Param. Assoc. w/5 param., Non-access type (control)
FPANC2 Formal in/out Param. Assoc. w/1 param., Non-access type (test)
FPAND1 Formal in/out Param. Assoc. w/10 param., Non-access type (control)
FPAND2 Formal in/out Param. Assoc. w/10 param., Non-access type (test)
FPRAA1 Formal in/out Parameter Ref., Access type (control)
FPRAA2 Formal in/out Parameter Ref., Access type (test)
FPRNA1 Formal in/out Parameter Ref., Non-access type (control)
FPRNA2 Formal in/out Parameter Ref., Non-access type (test)
GVRAA1 Global Var. Ref., Access type (control)
GVRAA2 Global Var. Ref., Access type (test)
GVRNA1 Global Var. Ref., Non-access type (control)
GVRNA2 Global Var. Ref., Non-access type (test)
IADDA1 Integer Addition (control)
IADDA2 Integer Addition (test)
IDIVA1 Integer Division (control)
IDIVA2 Integer Division (test)
IEXPA1 Integer Exponentiation (control)
IEXPA2 Integer Exponentiation (test)
IMIXA1 Integer Mixed Expressions 01 (control)
IMIXA2 Integer Mixed Expressions 01 (test)
IMIXB1 Integer Mixed Expressions 02 (control)
IMIXB2 Integer Mixed Expressions 02 (test)
IMIXC1 Integer Mixed Expressions 03 (control)
IMIXC2 Integer Mixed Expressions 03 (test)
IMIXD1 Integer Mixed Expressions 04 (control)
IMIXD2 Integer Mixed Expressions 04 (test)
IMIXE1 Integer Mixed Expressions 05 (control)
IMIXE2 Integer Mixed Expressions 05 (test)
IMODA1 Integer Modulus (control)
IMODA2 Integer Modulus (test)
IMULA1 Integer Multiplication (control)
IMULA2 Integer Multiplication (test)
IREMA1 Integer Remainder (control)
INTDA2 CHECKS 150 INTEGER DECLARATIONS
INTDB2 500 DECLARATION STMTS FOR INTEGER
INTDB3 500 DECLARATION STMTS FOR INTEGER (10 PER LINE)
INTQA2 TEST A FULL INTEGER QUEUE USING XOQUE PACKAGE
IREMA2 Integer Remainder (test)
ISEQA2 TEST GENERIC SEQUENCE MANIPULATION PACKAGE, 50 INTEGERS
ISUBA1 Integer Subtraction (control)
ISUBA2 Integer Subtraction (test)
LAVRA1 1 Local Array Var. Ref. (control)
LAVRA2 1 Local Array Var. Ref. (test)
LAVRB1 10 Local Array Var. Ref. (control)
LAVRB2 10 Local Array Var. Ref. (test)
LFIRA1 Loop Fuse, Index Ref. (control)
LFIRA2 Loop Fuse, Index Ref. (test)
LFSRA1 Loop Fuse, Scalar Ref. (control)
LFSRA2 Loop Fuse, Scalar Ref. (test)
LOAEA1 Loop Optimization, Asst. Eval. (control)
LOAEA2 Loop Optimization, Asst. Eval. (test)
LOBCA1 Loop Optimization, Expr. Calc. (control)
LOBCA2 Loop Optimization, Expr. Calc. (test)
LOFCA1 Loop Optimization, Function Call (control)
LOFCA2 Loop Optimization, Function Call (test)
LONEA1 Loop Optimization, Nested Expr. comp. (control)
LONEA2 Loop Optimization, Nested Expr. comp. (test)
LOSCA1 Loop Optimization, Subscript Calc. (control)
LOSCA2 Loop Optimization, Subscript Calc. (test)
LOUIA1 Loop Optimization, Unroll Index ref. (control)
LOUIA2 Loop Optimization, Unroll Index ref. (test)
LOUSA1 Loop Optimization, Unroll Scalar ref. (control)
LOUSA2 Loop Optimization, Unroll Scalar ref. (test)
LRR1A1 First-level Local Record var. Ref. (control)
LRR1A2 First-level Local Record var. Ref. (test)
LRR2A1 Second-level Local Record var. Ref. (control)
LRR2A2 Second-level Local Record var. Ref. (test)
LRR3A1 Third-level Local Record var. Ref. (control)
LRR3A2 Third-level Local Record var. Ref. (test)
LVRAA1 1 Local Var. Ref., Access type (control)
LVRAA2 1 Local Var. Ref., Access type (test)
LVRB1 10 Local Var. Ref., Access type (control)
LVRB2 10 Local Var. Ref., Access type (test)
LVRNA1 1 Local Var. Ref., Non-access type (control)
LVRNA2 1 Local Var. Ref., Non-access type (test)
LVRNB1 10 Local Var. Ref., Non-access type (control)
LVRNB2 10 Local Var. Ref., Non-access type (test)
MINIA2 MINIMAL PROGRAM WITH 1 STMT, 1 DECLARATION
MTCQA2 TEST EMPTY CHARACTER QUEUE USING XOQUE PACKAGE
MTESA2 TEST EMPTY SET OF ENUMERATION TYPE USING XOSET PACKAGE
MTISA2 TEST EMPTY SET OF INTEGERS USING XOSET PACKAGE
MULTA1 10_000 floating pt. Multiplications (control)
MULTA2 10_000 floating pt. Multiplications (test)
MULTA3 10_000 floating pt. Multiplications (pragma suppress)
NLOOA1 Overhead for Nested Loops - NO loops (control)
NLO7A2 Overhead for 7 Nested Loops (test)
NL65A2 Overhead for 65 Nested Loops (test)
NPPCA1 No Parameter Procedure Call (control)
NPPCA2 No Parameter Procedure Call (test)
NRPCA1 Nested Recursive Procedure Call (control)
NRPCA2 Nested Recursive Procedure Call (test)
NULLA1 NULLPROCEDURE (CONTROL)
NULLA2 CALL TO NULL PROCEDURE
OPAEA1 Optimization Perf., Arith. Elim. (control)
OPAEA2 Optimization Perf., Arith. Elim. (test)
OPBFA1 Optimization Perf., Bool. Folding (control)
OPBFA2 Optimization Perf., Bool. Folding (test)
OPCEA1 Optimization Perf., Call Elim. (control)
OPCEA2 Optimization Perf., Call Elim. (test)
OPCFA1 Optimization Perf., Constant Folding (control)
OPCFA2 Optimization Perf., Constant Folding (test)
OPDSA1 Optimization Perf., Distributed Simp. (control)
OPDSA2 Optimization Perf., Distributed Simp. (test)
OPISA1 Optimization Perf., Identity Simp. (control)
OPISA2 Optimization Perf., Identity Simp. (test)
OPLEA1 Optimization Perf., Load Elim. (control)
OPLEA2 Optimization Perf., Load Elim. (test)
OPNFA1 Optimization Perf., Num. Folding (control)
OPNFA2 Optimization Perf., Num. Folding (test)
OPSCA1 Optimization Perf., Subscript Calc. (control)
OPSCA2 Optimization Perf., Subscript Calc. (test)
OPSEA1 Optimization Perf., Store Elim. (control)
OPSEA2 Optimization Perf., Store Elim. (test)
PGQUA2 TEST PUT_END AND GOT_END WITH AN ENUMERATED TYPE USING
XQQUE PKG
PIALA2 PI Algorithm (test)
PKGEA1 EACH PACKAGE BODY FOLLOWS DIRECTLY AFTER THE PKG SPEC
(CONTROL)
PKGEA2 EACH PACKAGE BODY FOLLOWS DIRECTLY AFTER THE PKG SPEC
PNCOA2 PRODUCER/CONSUMER PROBLEM
PRPCA1 Parallel Recursive Procedure Call (control)
PRPCA2 Parallel Recursive Procedure Call (test)
PRUAA1 Proc. Ref. to an Uplevel var., Access type (control)
PRUAA2 Proc. Ref. to an Uplevel var., Access type (test)
PRUNA2 Proc. Ref. to an Uplevel var., Non-access type (test)
PRUNA1 Proc. Ref. to an Uplevel var., Non-access type (control)
SIEVA1 Sieve of Eratosthenes (control)
PUZZA2 PUZZLE
PUZZA3 PUZZLE (PRAGMA SUPPRESS)
RANDA2 RANDOM NUMBER GENERATOR
RCDSA2 CHECKS 400 FIELD RECORDS
REND A1 SIMPLE RENDEZVOUS (CONTROL)
REND A2 SIMPLE RENDEZVOUS
SHARA2 READERS/WRITERS PROBLEM
SIEVA2  Sieve of Eratosthenes (test)
SORTA2  TEST INSERTION SORT USING XOSORT PACKAGE
SQ1OA2  PUT 10 INTEGERS IN SEQUENCE AND TEST IF EMPTY USING XOSEQ PACKAGE
SQPGA2  PUT AND GET 10 INTEGERS IN SEQUENCE USING XOSEQ PACKAGE
SRCRA1  Simple Record Component Ref. (control)
SRCRA2  Simple Record Component Ref. (test)
SRTEA1  Simple Record Type Elaboration (control)
SRTEA2  Simple Record Type Elaboration (test)
TAIPA1  Task Perf. w/1 element Array 'in' Param. (control)
TAIPB1  Task Perf. w/32 element Array 'in' Param. (control)
TAIPC1  Task Perf. w/64 element Array 'in' Param. (control)
TAIPD1  Task Perf. w/320 element Array 'in' Param. (control)
TAIPF1  Task Perf. w/3200 element Array 'in' Param. (control)
TAOPA1  Task Perf. w/1 element Array 'in out' Param. (control)
TAOPB1  Task Perf. w/32 element Array 'in out' Param. (control)
TAOPC1  Task Perf. w/64 element Array 'in out' Param. (control)
TAOPE1  Task Perf. w/640 element Array 'in out' Param. (control)
TAOPF1  Task Perf. w/3200 element Array 'in out' Param. (control)
TPGTA2  Task Perf., Guard Test, 2 guards (test)
TPGB2  Task Perf., Guard Test, 2 guards (test)
TPGTC2  Task Perf., Guard Test, 20 guards (test)
TPGTD2  Task Perf., Guard Test, 20 guards (test)
TPITA1  Task Performance w/1 Idle Task (control)
TPITA2  Task Performance w/1 Idle Task (test)
TPITB1  Task Performance w/5 Idle Tasks (control)
TPITB2  Task Performance w/5 Idle Tasks (test)
TPITC1  Task Performance w/10 Idle Tasks (control)
TPITC2  Task Performance w/10 Idle Tasks (test)
TPITD1  Task Performance w/20 Idle Tasks (control)
TPITD2  Task Performance w/20 Idle Tasks (test)
TPOTA2  Task Perf., Order Test (test)
TPOTB2  Task Perf., Order Test (test)
TPOTC2  Task Perf., Order Test (test)

A.6
TPSTA2 Task Perf., Select test (test)
TPSTB2 Task Perf., Select Test (test)
TPTCA2 Task Perf., Task Chain, length 1 (test)
TPTCB2 Task Perf., Task Chain, length 5 (test)
TPTCC2 Task Perf., Task Chain, length 10 (test)
TPTCD2 Task Perf., Task Chain, length 20 (test)
TPUTA2 Task Perf. Unknown Test (test)
TPUTB2 Task Perf., Unknown Test (test)
TPUTC2 Task Perf., Unknown Test (test)
TPUD2 Task Perf., Unknown Test (test)
TPUTE2 Task Perf., Unknown Test (test)
UAPAA1 Unconst. Array Param. Assoc. w/3 elems. (control)
UAPAA2 Unconst. Array Param. Assoc. w/3 elems. (test)
UAPAB1 Unconst. Array Param. Assoc. w/63 elems. (control)
UAPAB2 Unconst. Array Param. Assoc. w/63 elems. (test)
VFADA1 Vector Floating pt. Addition (control)
VFADA2 Vector Floating pt. Addition (test)
VIADA1 Vector Integer Addition (control)
VIADA2 Vector Integer Addition (test)
VPGSA2 TEST VARIOUS PUTS AND GETS IN SEQUENCE USING XOSEQ PACKAGE
WHETA2 WHETSTONE INSTRUCTIONS WITH FLOATS
WHETA3 WHETSTONE INSTRUCTIONS WITH FLOATS (Pragma SUPPRESS)
WHLPA1 EVALUATES THE EFFICIENCY OF A LOOP STATEMENT USING WHILE (CONTROL)
WHLPA2 EVALUATES THE EFFICIENCY OF A LOOP STATEMENT USING WHILE
The package specifications for all of the Ada packages in the ACEC are provided below.

**IO_PACKAGE.ADA specification:**

```ada
with TEXT_IO;
use TEXT_IO;

-- This package abstracts I/O operations from the Statistics files defined in LIST_STATISTICS. It is a consequence of Ada that such procedures must be defined for any structured types.
-- The package encapsulates details about file contents, layout, etc.

package IO_PACKAGE is

-- Constants

COLUMNS : constant := 120;
    -- must = SCHEMA.DESCRIPTION_LENGTH
UNIT_NAME_LENGTH : constant := 6;
    -- must = SCHEMA.NAME_LENGTH
MAX_FILE_LENGTH : constant := 22;

C_HEADER : constant STRING := "Name Elapsed/CPU Code/Data Comments";
R_HEADER : constant STRING := "Name Event Elapsed/CPU Code/Data Comments";
I_HEADER : constant STRING := "Name Event Elapsed/CPU Comments";

-- Basic types

type Choice_Type is (START_REC, COM_REC, STOP_REC);
subtype Name_Type is STRING(1..UNIT_NAME_LENGTH);
subtype Size_Type is NATURAL; -- integer bytes
subtype Comment_Length is NATURAL range 0..COLUMNS;
subtype File_Name_Type is STRING(1..MAX_FILE_LENGTH);

BLANK_UNIT_NAME : Name_Type := (others => ' ');
BLANK_FILE_NAME : File_Name_Type := (others => ' ');

-- Structured types

type File_Record_Type is record
    FILE_EXISTS : BOOLEAN := FALSE;

B.1
FILE_NAME : File_Name_Type := BLANK_FILE_NAME;
INTERNAL_NAME : TEXT_IO.File_Type;
end record;
type Compilation_Record_Type(LEN : Comment_Length := 0) is record
  TEST_NAME : Name_Type := BLANK_UNIT_NAME;
  TOTAL_ELAPSED_TIME : DURATION := 0.0;
  TOTAL_CPU_TIME : DURATION := 0.0;
  OBJECT_CODE_SIZE : Size_Type := 0;
  COMMENTS : STRING(1..LEN) := (others=>''');
end record;
type Run_Time_Record_Type(LEN : Comment_Length := 0) is record
  TEST_NAME : Name_Type := BLANK_UNIT_NAME;
  TOTAL_ELAPSED_TIME : DURATION := 0.0;
  TOTAL_CPU_TIME : DURATION := 0.0;
  MEMORY_CODE_SIZE : Size_Type := 0;
  MEMORY_DATA_SIZE : Size_Type := 0;
  COMMENTS : STRING(1..LEN) := (others=>''');
end record;
type Instrumentation_Record_Type(LEN : Comment_Length := 10) is record
  TEST_NAME : Name_Type := BLANK_UNIT_NAME;
  IDENT : Choice_Type := START_REC;
  ELAPSED_TIME : DURATION := 0.0;
  ELAPSED_CPU_TIME : DURATION := 0.0;
  COMMENTS : STRING(1..LEN) := (others=>''');
end record;

-- File I/O Operations
procedure Get_File_Name(FILE: in out File_Record_Type);
procedure Open_File(FILE: in out File_Record_Type);

B.2
MODE: in TEXT_IO.FILE_MODE := IN_FILE);

procedure Close_File( FILE: in out File_Record_Type);
procedure File_Status( FILE: in File_Record_Type);

-- Compilation I/O

procedure Get( FILE: in File_Type;
VALUE : out Compilation_Record_Type);

procedure Put( FILE: in File_Type:= CURRENT_OUTPUT;
VALUE : in Compilation_Record_Type);

-- Run Time I/O

procedure Get( FILE: in File_Type;
VALUE : out Run_Time_Record_Type);

procedure Put( FILE: in File_Type:= CURRENT_OUTPUT;
VALUE : in Run_Time_Record_Type);

-- Instrumentation I/O

procedure Get( FILE: in File_Type;
VALUE : out Instrumentation_Record_Type);

procedure Put( FILE: in File_Type:= CURRENT_OUTPUT;
VALUE : in Instrumentation_Record_Type);

end IO_PACKAGE;

LIST_PACKAGE.ADA specification:

generic
type List_Element is private;
package Singly_Linked_List is

-- Abstract : This package provides an abstraction for a singly linke

type List_Type is private;
function Empty (List : List_Type) return Boolean;
-- Indicates whether the list contains any elements.
function Null_Node (List : List_Type) return Boolean;
-- Indicates whether the "current pointer" references an element in
-- the list.
function Head_Node (List : List_Type) return Boolean;
-- Indicates whether the "current pointer" references the head of
-- the list.

B.3
function Tail_Node (List : List_Type) return Boolean;
-- Indicates whether the "current pointer" references the tail of
-- the list.
function Current_Element (List : List_Type) return List_Element;
-- Returns the value of the element referenced by the "current
-- pointer".
-- Raises End_Error if Null_Node(List) = True.
procedure First (List : in out List_Type);
-- Positions the "current pointer" at the head of the list
-- (even if the list is empty).
procedure Next (List : in out List_Type);
-- Positions the "current pointer" at the next element in the list.
-- After the last element in the list Null_Node(List) becomes True.
-- Raises End_Error if Null_Node(List) = True.
procedure Insert_After (List : in out List_Type;
  Element : List_Element);
-- Inserts an element after the "current pointer".
-- If Null_Node(List) = True the element is appended after the tail
-- element.
procedure Insert_Before (List : in out List_Type;
  Element : List_Element);
-- Inserts an element before the "current pointer".
-- If Null_Node(List) = True the element is prepended before the
-- head element.
procedure Delete_Element (List : in out List_Type);
-- Deletes the element referenced by the "current pointer" from
-- the list.
-- Upon deletion, the "current pointer" references the element
-- after the deleted element.
-- Raises End_Error if Null_Node(List) = True.
generic
  with procedure Transformation (Element : in out List_Element);
procedure Modify (List : List_Type);
-- Permits modification of the element referenced by the "current
-- pointer" where the modification doesn't require external values
-- (e.g. incrementing a field of the element).
-- Raises End_Error if Null_Node(List) = True.
generic
  type Update_Information is private;
  with procedure Transformation (Element : in out List_Element;
    Information : Update_Information);
procedure Update (List : List_Type;
  Information : Update_Information);
-- Permits modification of the element referenced by the "current
-- pointer" where the modification requires external values
-- (e.g. assigning a value to a field of the element).
-- Raises End_Error if Null_Node(List) = True.
pragma Inline (Empty, Null_Node, Head_Node, Tail_Node, Current_Element);
pragma Inline (Modify, Update);
End_Error : exception;

B.4
private
type Node:
type Node_Access is access Node:
type Node is
record
  Element : List_Element;
  Next : Node_Access;
end record;
type List_Type is
record
  Head : Node_Access;
  Tail : Node_Access;
  Previous : Node_Access;
  Current : Node_Access;
end record;
end Singly_Linked_List;

SCHEMA.ADA specification:

with SINGLY_LINKED_LIST;
package SCHEMA is
-- exports 7 basic types describing test units:
-- Architecture_Category_Type -- benchmark category of the test
-- Description_Type -- a string describing the test
-- E_and_V_Criterion_Type -- the E and V team category of
  the test. May be more than one,
  which is the reason for the
  List types
-- Language_Feature_Type -- the Ada language category of the
  test. May also be more than one.
-- Name_Type -- the short name (also file name).
-- Statistics_Type -- the principle kind of statistics
  measured by the test.
-- Version_Type -- the benchmark version of the
test.
-- the rest is there for implementation reasons

  type Architecture_Category_Type is (NORMATIVE_PERFORMANCE,
    NORMATIVE_CAPACITY, OPTIONAL_FEATURE,
    OPTIONAL_ALGORITHM, EVERY);

  DESCRIPTION_LENGTH : constant := 120; -- see User's Manual
  subtype Description_Type is STRING (1 .. DESCRIPTION_LENGTH);

  CRITERION_LENGTH : constant := 36;
  subtype E_and_V_Criterion_Type is STRING (1 .. CRITERION_LENGTH);
FEATURE_LENGTH : constant := 51;
subtype Language_Feature_Type is STRING (1 .. FEATURE_LENGTH);

NAME_LENGTH : constant := 6;
subtype Name_Type is STRING (1 .. NAME_LENGTH);

type Statistics_Type is (COMPILATION, EXECUTION, BOTH);

type Version_Type is (CONTROL, TEST, OPTIMIZE, SUPPRESS, OTHER, ALL_VERSIONS);

-- Lists of E and V Criteria

type E_and_V_Criterion_Abbreviation_Type is (EFFCY01, EFFCY06, EFFCY13, EFFCY18, EFFCY21, EFFCY22, EFFCY26, EFFCY29, EFFCY32);

package CRITERION_LISTS is
    new SINGLY_LINKED_LIST(E_and_V_Criterion_Abbreviation_Type);
use CRITERION_LISTS;
type E_and_V_Criteria_List is new CRITERION_LISTS.List_Type;
function EXPAND( KEY: E_and_V_Criterion_Abbreviation_Type ) return E_and_V_Criterion_Type;

-- Lists of Language Features

type Language_Feature_Abbreviation_Type is
    (IDENTIFIERS, LITERALS, DERIVED_TYPES, SCALAR_TYPES, ARRAY_TYPES, RECORD_TYPES, ACCESS_TYPES, LOCAL_NAMES, NON_LOCAL_NAMES, INDEXED_COMPS, SLICES, ATTRIBUTES, AGGREGATES, ARRAY_AGGS, EXPRESSIONS, RELATIONAL_OPERATORS, BINARY_ADDS, MULTIPLYING_OPS, HI_PRECEDENCE_OPS, QUALIFIED_EXPRESSIONS, ALLOCATORS, ASSIGNMENT, ARRAY_ASSIGN, CASE_STMTS, LOOP_STMTS, BLOCK_STMTS, EXIT_STMTS, RETURN_STMTS, SUBROUTINE_DECLS, SUBROUTINE_CALLS, DEFAULT_PRAMS, OVERLOADING, PACKAGE_BODIES, PRIVATE_TYPES, USE_CLAUSES, RENAMING_DECLS, TASK_EXECUTION, TASK_DEPENDENCE, DELAY_STMTS, SELECT_STMTS, CONDITIONAL_ENTRIES, TIMED_ENTRIES, ABORT_STMTS, CONTEXT_CLAUSES, EXCEPTION_DECLS, EXCEPTION_HANDLERS, EXCEPTION_PROPAGATION, GENERIC_DECLS, GENERIC_BODIES, GENERIC_INSTS, USES_OF_GENERIC_INSTS, LENGTH_CLAUSES, ENUM_REP_CLAUSES, RECORD_REP_CLAUSES, PARAMETER_ASSN, PACKAGE_SPECS_DECLS, REFERENCES_TO_OBJECTS, TASK_TYPES_OBJECTS, ENTRIES_ACCEPTS, SELECTIVE_WAITS, TASK_ENTRY_ATTRIBS, SUBUNITS, RAISE_STMTS, REPRESENTATION_CLAUSES, B.6);
package FEATURE_LISTS is
   new SINGLY_LINKED_LIST(Language_Feature_Abbreviation_Type);
use FEATURE_LISTS;
type Language_Features_List is new FEATURE_LISTS.List_Type;
function EXPAND( KEY: Language_Feature_Abbreviation_Type ) return
   Language_Feature_Type;
end FEATURE_LISTS;

-- Lists of Test Unit Names
package NAME_LISTS is new SINGLY_LINKED_LIST(Name_Type);
use NAME_LISTS;
type Name_List is new NAME_LISTS.List_Type;

end SCHEMA;

ATTRIBUTE.ADA specification:

with SCHEMA;
use SCHEMA;
package ATTRIBUTE_OPTIONS is
   -- LIST_BY_NAME means that attributes will be listed for a
   -- single instance of a test file name
   -- LIST_BY_CATEGORY means that attributes will be listed for
   -- all test files included in the specified attribute
   type Listing_Type is ( LIST_BY_NAME, LIST_BYCATEGORY );
   type Attribute_Type is ( SHORT, TEST_NAME, DESCRIPTION,
      ARCHITECTURE, E_AND_V, LANGUAGE_FEATURE,
      VERSION, STATISTICS );
   subtype Category_Type is Attribute_Type range ARCHITECTURE..STATISTICS;
   -- These subprograms set and observe the internal state maintained by
   -- the package body
   procedure SET_LIST( SWITCH : in Listing_Type := LIST_BY_NAME );
   function LISTING return Listing_Type;
   procedure SET( OPTION : in Attribute_Type );
   procedure RESET( OPTION : in Attribute_Type );
   function IS_SET( OPTION : in Attribute_Type ) return BOOLEAN;

B.7
procedure SET_QUERY( VALUE : 
in Architecture_Category_Type );
procedure SET_QUERY( VALUE : 
in E_and_V_Criterion_Abbreviation_Type );
procedure SET_QUERY( VALUE : 
in Language_Feature_Abbreviation_Type );
procedure SET_QUERY( VALUE : 
in Statistics_Type );
procedure SET_QUERY( VALUE : 
in Version_Type );

function CATEGORY return Category_Type;
function VALUE return Architecture_Category_Type;
function VALUE return E_and_V_Criterion_Abbreviation_Type;
function VALUE return Language_Feature_Abbreviation_Type;
function VALUE return Statistics_Type;
function VALUE return Version_Type;

end ATTRIBUTE_OPTIONS;

DATABASE.ADA specification:

with SCHEMA;
use SCHEMA;
package DATABASE_INTERFACE is
-- Each of the following functions will take as input a test name.
-- and return as output either a single object or a list of objects
-- as determined by the specific function called.

function GET_DESCRIPTION( INPUT_NAME : Name_Type) return Description_Type;
function GET_ARCH_CATEGORY( INPUT_NAME : Name_Type) return Architecture_Category_Type;
function GET_E_AND_V_CATEGORIES( INPUT_NAME : Name_Type) return E_and_V_Criteria_List;
function GET_FEATURES( INPUT_NAME : Name_Type) return Language_Features_List;
function GET_STATISTICS( INPUT_NAME : Name_Type) return Statistics_Type;
function GET_VERSION( INPUT_NAME : Name_Type) return Version_Type;

function NAMES(ATTRIBUTE: Architecture_Category_Type) return Name_List;
function NAMES(ATTRIBUTE: E_and_V_Criterion_Abbreviation_Type) return Name_List;

B.8
function NAMES(ATTRIBUTE: Statistics_Type) return Name_List;
function NAMES(ATTRIBUTE: Version_Type) return Name_List;
function NAMES(ATTRIBUTE: LanguageFeature_Abbreviation_Type) return Name_List;

NOT_FOUND, -- raised when unit not in database
CONSISTENCY_ERROR: exception; -- raised when database file
-- is corrupted

end DATABASE_INTERFACE;

CPU_TIME.ADA specification:

package CPU_TIME is
  function CPU_CLOCK return duration;
end CPU_TIME;

INQUIRY.ADA specification:

package INQUIRY_OPERATIONS is

  type Command_Type is (COLLECT_COMMAND, SELECT_COMMAND,
                        PRINT_COMMAND, HELP_COMMAND,
                        SAVE_COMMAND, LIST_COMMAND,
                        QUIT_COMMAND);

  -- ANSWER prompts the user for a Yes-No response, converting the
  -- result to type BOOLEAN
  -- REQUEST prompts with a menu of choices, converting the
  -- result to type Command_Type
  -- INITIALIZE prints greeting and initial help info.
  -- COLLECT could be called more than once, to change file name setup
  -- -- builds Current Files Record and opens Statistics files.
  -- SELECT could be called repeatedly, to change selections
  -- -- builds Current Options Record
  -- LIST could be called out-of-sequence, and repeatedly
  -- -- uses built records
  -- PRINT dumps the current contents of the statistics files
  -- SAVE prints (a less formatted) version of the Report dialog to
  -- a named file
  -- HELP re-displays the initial prompt.
  -- QUIT closes any open files and exits.

B.9
-- function REQUEST return Command_Type;
procedure INITIALIZE;
procedure COLLECT_FILES;
procedure SELECT_ATTRIBUTES;
procedure LIST_STATISTICS;
procedure PRINT_FILES;
procedure HELP_PROMPT;
procedure SAVE_STATISTICS;
procedure QUIT;

end INQUIRY_OPERATIONS;

--

INSTRUMENT.ADA specification:

package Instrument is
-- The Instrument routines.
    procedure START -- THIS ROUTINE MUST BE INVOKED AT THE
                   -- START OF A TEST, BEFORE ANY OF THE
                   -- OTHER REPORT ROUTINES ARE INVOKED.
                   -- IT SAVES THE TEST NAME AND OUTPUTS
                   -- THE NAME AND DESCRIPTION.
                   ( NAME : STRING;
                     DESCR : STRING
                   );
    procedure COMMENT -- OUTPUT A COMMENT MESSAGE.
                        -- THE MESSAGE.
                        ( DESCR : STRING
                        );
    procedure STOP: -- THIS ROUTINE MUST BE INVOKED AT THE
                    -- END OF A TEST. IT OUTPUTS A MESSAGE
                    -- INDICATING WHETHER THE TEST AS A
                    -- WHOLE HAS PASSED OR FAILED, OR IS
                    -- NOT-APPLICABLE.
                    --
                    -- THE DYNAMIC VALUE ROUTINES.
                    -- EVEN WITH STATIC ARGUMENTS, THESE FUNCTIONS WILL HAVE
                    -- DYNAMIC RESULTS.
    function IDENT_INT -- AN IDENTITY FUNCTION FOR TYPE
                        -- INTEGER.
                        ( X : INTEGER
                        ) return INTEGER; -- X.
    function IDENT_CHAR -- AN IDENTITY FUNCTION FOR TYPE
                        -- CHARACTER.
                        ( X : CHARACTER
                        ) return CHARACTER; -- X.

B.10
function IDENT_BOOL -- AN IDENTITY FUNCTION FOR TYPE BOOLEAN.
( X : BOOLEAN -- THE ARGUMENT.
 ) return BOOLEAN; -- X.

function IDENT_STR -- AN IDENTITY FUNCTION FOR TYPE STRING.
( X : STRING -- THE ARGUMENT.
 ) return STRING; -- X.

function EQUAL -- A RECURSIVE EQUALITY FUNCTION FOR TYPE INTEGER.
( X, Y : INTEGER -- THE ARGUMENTS.
 ) return BOOLEAN; -- X = Y.

generic
type GEN_TYPE is ('');
package PROCS is
type t is new gen_type;
type ref_t is access t;
global: t;
global_object: t;
GLOBAL_ACCESS: REF_T := new T;
INIT: constant T := T'FIRST;
function IDENT(X: in T) return T;
procedure LET(X: in out T; Y: T);
end PROCS;

generic
type gen_type is ('');
ar_size: integer;
package arr_procs is
subtype index is integer range 1..arr_size;
type t is array(integer range ( ) of gen_type;
init: t(index) := (others = gen_type'first);
global: t(index);
function ident(x: t) return t;
procedure let(x: in out t; y: t);
end arr_procs;
end Instrument;

REPORT_WRITER.ADA:

with INQUIRY_OPERATIONS; use INQUIRY_OPERATIONS;
-- This subprogram acts as the 'main' routine of the portion
-- of the Benchmark system that deals with information
-- retrieval and user interaction.
procedure REPORT_WRITER is
begin
 INITIALIZE; -- display greeting and helpful prompt

loop
  case REQUEST is
    when COLLECT_COMMAND = COLLECT_FILES;
      -- set up statistics files
    when SELECT_COMMAND = SELECT_ATTRIBUTES;
      -- customize display
    when LIST_COMMAND = LIST_STATISTICS;
      -- display statistics and database attributes
    when SAVE_COMMAND = SAVE_STATISTICS;
      -- display statistics and database attributes
      -- (to named file)
    when PRINT_COMMAND = PRINT_FILES;
      -- dump contents of statistics files to screen
    when HELP_COMMAND = HELP_PROMPT;
      -- display helpful prompt
    when QUIT_COMMAND = QUIT; exit;
      -- close files
  end case:
  end loop;
end REPORT_WRITER;
APPENDIX C: Report Writer’s Guide

I. INTRODUCTION

The report writer is used by the Prototype ACEC system to report statistics generated by the host/target system and data supplied with the ACEC. The user carries on an interactive dialog with the report writer via a menu. From this menu, the user can 1) indicate which files are to be used as statistics files, 2) specify both the test attributes to be displayed and the type of database query, and 3) choose to list information either to the terminal or to a file. The remainder of this guide contains specific information on the menu (Section II) and the menu options (Section III). Section IV is an example run of the report writer.

II. MENU

There are seven options present in the report writer’s menu. The first option, “C”, collects the names of the files to be used as statistics files. The second option, “S”, provides for setting the test attributes to be displayed and for changing the database query parameter. Choosing the list option, “L”, queries the database using the attributes and parameter set by the second option and lists the desired information to the terminal. Option “P” displays all of the contents of all of the collected statistics files. Not only can the database information be displayed on the terminal, but it can also be put into a data file. This file is specified by using the fifth option, “F”. This option acts exactly like the list option except the output is sent to the given file instead of to the screen. If, at any time when the menu is displayed, you need some help, enter “H” to view the short help prompt. Finally, to leave the report writer, select the option “Q”, for “quit”. When quitting, all opened statistics files are closed and control is returned to the host system.

When the report writer is first executed, and after completing an option (except the QUIT option), the menu is displayed. Only the responses given in the menu are valid. Entering an invalid response will simply cause a reprompt. The responses can be given either in upper- or lower-case characters. The menu is as shown on the following page:
Valid choices are:
C     Collect names of statistics files.
S     Select attributes for reporting.
L     List selected attributes from files.
P     Print contents of current files to screen.
F     Save output to named file.
H     Re-display help prompt.
Q     Quit processing.

Please enter your choice:

III. OPTIONS

Whenever the menu is displayed, entering a valid response will result in a confirmation statement, such as "COLLECT command accepted" for the "C" response. Any values set in an option can be changed and rechanged as many times as necessary while running the report writer.

III.a. COLLECT, "C"

The COLLECT option collects the names of the files to be used as the statistics files. These statistics files are the compilation file, the run time file, and the instrumentation file. At the beginning of a session with the report writer, there are no file names associated with the files; they are "not defined." When this command is invoked, the current status of the statistics files is given. The user can then decide either not to change the status of these files, or to selectively change the status of individual files. After completion of this command, the new file status is given. Below is an example of using this option, where the user wants to change the compilation file to 'c.data', the run time file to 'r.data', and wants to leave the instrumentation file unchanged:

Please enter your choice: c
COLLECT command accepted.

COMPILATION file is not defined.
RUN_TIME file is not defined.
INSTRUMENTATION file is not defined.
-- the current files' status

C.2
Do you wish to change file(s)? [y/n]: y
COMPILATION file is not defined. & Do you wish to change? [y/n]: y
File name?: c.data
c.data open.
-- compilation file is now c.data
RUN_TIME file is not defined. & Do you wish to change? [y/n]: y
File name?: r.data
r.data open.
-- run time file is now r.data
INSTRUMENTATION file is not defined. & Do you wish to change? [y/n]: n
COMPILATION file is c.data
RUN_TIME file is r.data -- the new status
INSTRUMENTATION file is not defined

The file names given must have six characters. The file must also exist in order to be used by the report writer. However, entering a nonexistant file will not cause a problem:

COMPILATION file is not defined. & Do you wish to change? [y/n]: y
File name?: nofile
nofile *** File not found

The files collected can be changed or closed by the COLLECT command. Whenever a yes response ("y") is received to the prompt "Do you wish to change? [y/n]", the currently opened statistics file is closed and the user is asked for a file name. To keep that statistics file closed ("not defined"), simply enter a carriage return as the file name. Entering a six character file name will open that file. The file opened will be used as the statistics file. For example, given the three files are currently defined as 'c.data', 'r.data', and 'i.data' respectively, and we want to change the compilation file to 'cl.dat', leave the run time file as is, and close the instrumentation file, the example discourse with the report writer follows:

Please enter your choice: c
COLLECT command accepted.
COMPILATION file is c.data
RUN_TIME file is r.data

C.3
III.b. SELECT, "S"

The SELECT prompt is used to change the database query and to specify which attributes of the tests are to be displayed. There are two basic database queries. The first is LIST_BY_NAME, and the other is LIST_BYCATEGORY. The LIST_BY_NAME query is used to gain information about single specific tests. The LIST_BYCATEGORY query finds all tests that satisfy the category parameter specified. The possible values for the category query are Architecture, E and V Criteria, Language Feature, Version, and Statistic. Under each of these categories are the specific values: Normative performance, Normative capacity, Optional feature, and Optional algorithm are Architecture values. Below is an example of using the SELECT option to change only the database query, with the default list setting and category setting marked:

SELECT command accepted.
Do you wish to change query? [y\n]: y
Listing setting: LIST_BY_NAME -- default list setting
Do you wish to change? [y\n]: y
Listing setting: LIST_BYCATEGORY
Category is: ARCHITECTURE Current Value is EVERY -- default for category
Do you wish to change both Category and Value? [y\n]: y
The choice is one of the following:
ARCHITECTURE
E_AND_V
LANGUAGE_FEATURE
VERSION
STATISTICS

C.4
Select the new category as it comes by...

ARCHITECTURE \[y\|n\]: n
E_AND_V \[y\|n\]: y -- select the E_AND_V category
Now, select a value as it comes by ...
EFFCY01 \[y\|n\]: n
EFFCY06 \[y\|n\]: y -- select the EFFCY06 value
Category is: E_AND_V Current Value is EFFCY06
Do you wish to change display? \[y\|n\]: n

After specifying the desired query, the user can change the display. If the user requires changes to be made, the first display prompt concerns whether the information for display must be in its long or short form. ‘EFFCY01’ is the short form of efficiency criteria 01, whereas "Speed of object code generation" is the long form. With the short form, short language feature names and only one header for the various statistics file data are printed. The other questions concern which attributes are to be printed. For all of the attributes, the current setting of the option is given and the user is asked whether to change the current setting or to leave this value unchanged. After all the attributes have been set, the new values of the options are shown. Below is an example, with the default display options marked:

SELECT command accepted.
Do you wish to change query? \[y\|n\]: n
Do you wish to change display? \[y\|n\]: y
Display options are:

Attribute: SHORT
Current value is: FALSE -- default
Do you wish to change? \[y\|n\]: n

Attribute: TEST_NAME
Current value is: TRUE -- default
Do you wish to change? \[y\|n\]: n

Attribute: DESCRIPTION
Current value is: TRUE -- default
Do you wish to change? \[y\|n\]: n

Attribute: ARCHITECTURE
Current value is: TRUE -- default
Do you wish to change? \[y\|n\]: n

Attribute: E_AND_V
Current value is: TRUE -- default
Do you wish to change? \[y\|n\]: y... changed.

C.5
III.c. LIST, "L"

The LIST option is used to query the database. If the query is LIST_BY_NAME, then a 'test unit name' will be asked for. After the test name is received, information about that test is displayed according to the display settings made by the user (or the defaults). For the LIST_BYCATEGORY query, all entries in the database that satisfy the query will be displayed according to the display settings.

Below are examples of both LIST_BY_NAME and LIST_BYCATEGORY sessions:

LIST_BY_NAME with display options DESCRIPTION, and STATISTICS and only the compilation statistics file opened.

Please enter your choice: 1
LIST command accepted.
What is the test unit name? piala2
Description: PI Algorithm (test)
Statistics: BOTH
COMPIlation
<table>
<thead>
<tr>
<th>Name</th>
<th>Elapsed/CPU</th>
<th>Code/Data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIALA2</td>
<td>14.29</td>
<td>2.10</td>
<td>1536</td>
</tr>
</tbody>
</table>

Do you want to list another unit? [y/n]: y

What is the test unit name? tpitc2

Description: Task Performance w/10 Idle Tasks (test)

COMPILATION

<table>
<thead>
<tr>
<th>Name</th>
<th>Elapsed/CPU</th>
<th>Code/Data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPITC2</td>
<td>18.79</td>
<td>4.56</td>
<td>11776</td>
</tr>
</tbody>
</table>

Do you want to list another unit? [y/n]: y

What is the test unit name? notest

Description: *** Sorry, NOTEST does not have a database entry

Do you want to list another unit? [y/n]: n

---

LIST_BYCATEGORY with category E_AND_V value EFFCY26 and display options SHORT, TEST_NAME, E_AND_V, and LANGUAGE_FEATURE:

Please enter your choice: 1

LIST command accepted.

Test Name: ADDSA1
E and V Criteria: EFFCY21 EFFCY22 EFFCY01 EFFCY26
Language Feature(s): BINARY_ADDS

Test Name: ADDSA2
E and V Criteria: EFFCY21 EFFCY22 EFFCY01 EFFCY26
Language Feature(s): BINARY_ADDS

---

III.d. PRINT. "P"

The PRINT option dumps all of the collected statistics files to the screen. If there are no statistics files collected then nothing will be printed to the screen.

III.e. SAVE or FILE, "F"

The "F" option is exactly like the LIST option except that the information extracted from the database is put into a file specified by the user. The file name asked for must be six (6)
characters long. At the completion of this command, the specified file will be closed. The following example is just like the LIST example above, where the category was LIST_BY_NAME:

```
Please enter your choice: f
SAVE command accepted.
File name?: listed
listed ... open.
What is the test unit name? piala2
Do you want to list another unit? [y/n]: y
What is the test unit name? tpitc2
Do you want to list another unit? [y/n]: n
listed ... closed.

Here is the file 'listed':
```

```
Description: PI Algorithm (test)
Statistics: BOTH
COMPILATION
Name       Elapsed/CPU Code/Data Comments
PIALA2     14.29  2.10  1536

Description: Task Performance w/10 Idle Tasks (test)
Statistics: BOTH
COMPILATION
Name       Elapsed/CPU Code/Data Comments
TPITC2     18.79  4.56  11776
```

III.f. HELP, "H"

The HELP option re-displays the report writer header.

III.g. QUIT, "Q"

To quit from the report writer, enter the QUIT response to the menu prompt. When quitting, all opened files are closed.

C.8
IV. EXAMPLE

Below is an example of running the report writer from beginning to end:

-------------------

Initializing Inquiry_Operations.

There are 3 parameters you can set/change:

1. COLLECT the names of files which contain Statistics
   Then SELECT the Attributes you wish to display.
2. The Query parameter tells the Report Writer how to search the
   Database and Statistics files.
   You can query by Name or by Category.
3. The Display parameter tells what test unit Attributes to display.
   The LIST command produces the Report you’ve defined.

Use the PRINT command to see what’s in the selected Statistics files.
SAVE is just like List, but lets you put the report in a named file for
later processing in the Host environment.

You can change Statistics files and Attributes as often as you wish.

See the User’s Manual for details.

Ready to continue? [y:n]: y
Valid choices are:
   C Collect names of statistics files.
   S Select attributes for reporting.
   L List selected attributes from files.
   P Print contents of current files to screen.
   F Save output to named file.
   H Re-display help prompt.
   Q Quit processing.

Please enter your choice: 1
LIST command accepted.
What is the test unit name? akera2
Test Name: AKERA2
Description: Ackermann function (test)
Version: TEST
Architecture Category: OPTIONAL_ALGORITHM
E and V Criteria: Object code size
   Execution time
   Speed of object code generation

C.9
Language Feature(s): Subprogram Calls
Scalar Types, Declarations, and Object & Declarations

Statistics: BOTH

Do you want to list another unit? [y:n]: y
What is the test unit name? sieval
Test Name: SIEVAL
Description: Sieve of Eratosthenes (control)
Version: CONTROL
Architecture Category: OPTIONAL_ALGORITHM
E and V Criteria: Object code size
Execution time
Speed of object code generation
Execution time, arith. & logic opers
Language Feature(s): Loop Statements
Relational Operators and Membership Tests
Binary Adding Operators

Statistics: BOTH

Do you want to list another unit? [y:n]: y
What is the test unit name? notest
Test Name: NOTEST
Description: *** Sorry, NOTEST does not have a Database entry.

Do you want to list another unit? [y:n]: n
Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.

Please enter your choice: j
... please try again.

h
HELP command accepted.
There are 3 parameters you can set/change:

1. COLLECT the names of files which contain Statistics
   Then SELECT the Attributes you wish to display.
2. The Query parameter tells the Report Writer how to search the
   Database and Statistics files.
   You can query by Name or by Category.
3. The Display parameter tells what test unit Attributes to display.
   The LIST command produces the Report you’ve defined.

Use the PRINT command to see what’s in the selected & Statistics files.

C.10
SAVE is just like List, but lets you put the report in a named file for later processing in the Host environment.

You can change Statistics files and Attributes as often as you wish.

See the User’s Manual for details.

Ready to continue? [y/n]: y
Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.
Please enter your choice: c
COLLECT command accepted.

COMPILATION file is not defined.
RUNTIME file is not defined.
INSTRUMENTATION file is not defined.
Do you wish to change file(s)? [y/n]: y
COMPILATION file is not defined. Do you wish to change? [y/n]: y
File name?: nofile
 *** File not found.

RUNTIME file is not defined. Do you wish to change? [y/n]: n
INSTRUMENTATION file is not defined. Do you wish to change? [y/n]: n

COMPILATION file is not defined.
RUNTIME file is not defined.
INSTRUMENTATION file is not defined.

Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.

Please enter your choice: c
COLLECT command accepted.

COMPILATION file is not defined.
RUNTIME file is not defined.
INSTRUMENTATION file is not defined.
Do you wish to change file(s)? [y/n]: y

C.11
COMPILATION file is not defined. Do you wish to change? [y/n]: y
File name?: c.data
c.data ... open.
RUN_TIME file is not defined. Do you wish to change? [y/n]: y
File name?: r.data
r.data ... open.
INSTRUMENTATION file is not defined. y
Do you wish to change? [y/n]: y
File name?: i.data
i.data ... open.
COMPILATION file is c.data
RUN_TIME file is r.data
INSTRUMENTATION file is i.data

Valid choices are:
C Collect names of statisti's files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.

Please enter your choice: 1
LIST command accepted.
What is the test unit name? piala2
Test Name: PIALA2
Description: PI Algorithm (test)
Version: TEST
Architecture Category: OPTIONAL_ALGORITHM
E and V Criteria: Speed of object code generation
Object code size
Execution time
Execution time, arith. & logic opers
Language Feature(s): Binary Adding Operators
Multiplying Operators
Highest Precedence Operators
Statistics: BOTH

COMPILATION
Name    Elapsed/CPU Code/Data Comments
PIALA2  14.29  2.10  1536

RUN_TIME
Name    Event Elapsed/CPU Code/Data Comments
PIALA2  STOP_REC  0.00  0.00

Do you want to list another unit? [y/n]: y
What is the test unit name? tpitc2

Test Name: TPITC2
Description: Task Performance w/10 Idle Tasks (test)
Version: TEST
Architecture Category: NORMATIVE_PERFORMANCE
E and V Criteria: Speed of object code generation
Idle task effect on performance
Object code size
Execution time
Language Feature(s): Task Types and Task Objects
Task Execution - Task Activation
Statistics: BOTH

COMPILATION
Name Elapsed/CPU Code/Data Comments
TPITC2 18.79 4.56 11776

RUN_TIME
Name Event Elapsed/CPU Code/Data Comments
TPITC2 11.77 1.30 512 512

INSTRUMENTATION
Name Event Elapsed/CPU Comments
TPITC2 STOP_REC 1.36 1.36

Do you want to list another unit? [y/n]: n
Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.

Please enter your choice: s
SELECT command accepted.
Do you wish to change query? [y/n]: y
Listing setting: LIST_BY_NAME
Do you wish to change? [y/n]: y
Listing setting: LIST_BY_CATEGORY
Category is: ARCHITECTURE Current Value is EVERY
Do you wish to change both Category and Value? [y/n]: y
The choice is one of the following:
ARCHITECTURE
E_AND_V
LANGUAGE_FEATURE
VERSION
STATISTICS
Select the new category as it comes by...

ARCHITECTURE [y/n]: n
E_AND_V [y/n]: y

C.13
Now, select a value as it comes by ...

EFFCY01 [y\n]: n
EFFCY06 [y\n]: n
EFFCY13 [y\n]: n
EFFCY18 [y\n]: n
EFFCY21 [y\n]: n
EFFCY22 [y\n]: n
EFFCY26 [y\n]: y

Category is: E_AND_V Current Value is EFFCY26
Do you wish to change display? [y\n]: y

Display options are:

Attribute: SHORT Current value is: FALSE
Do you wish to change? [y\n]: y
... changed.

Attribute: TEST_NAME Current value is: TRUE
Do you wish to change? [y\n]: y
... changed.

Attribute: DESCRIPTION Current value is: TRUE
Do you wish to change? [y\n]: y
... changed.

Attribute: ARCHITECTURE Current value is: TRUE
Do you wish to change? [y\n]: n

Attribute: E_AND_V Current value is: TRUE
Do you wish to change? [y\n]: n

Attribute: LANGUAGEFEATURE Current value is: TRUE
Do you wish to change? [y\n]: n

Attribute: VERSION Current value is: TRUE
Do you wish to change? [y\n]: n

Attribute: STATISTICS Current value is: TRUE
Do you wish to change? [y\n]: n

Display options are:

Attribute: SHORT Current value is: TRUE
Attribute: TEST_NAME Current value is: FALSE
Attribute: DESCRIPTION Current value is: FALSE
Attribute: ARCHITECTURE Current value is: TRUE
Attribute: E_AND_V Current value is: TRUE
Attribute: LANGUAGE_FEATURE Current value is: TRUE
Attribute: VERSION Current value is: FALSE
Attribute: STATISTICS Current value is: TRUE

Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.

C.14
Re-display help prompt.
Q Quit processing.

Please enter your choice: 1

LIST command accepted.

<table>
<thead>
<tr>
<th>Architecture Category: NORMATIVE_PERFORMANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E and V Criteria: EFFCY21 EFFCY22 EFFCY01 EFFCY26</td>
</tr>
<tr>
<td>Language Feature(s): BINARY_ADDS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Event</th>
<th>Elapsed/CPU</th>
<th>Code/Data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDSA1</td>
<td></td>
<td>13.74</td>
<td>2.30</td>
<td>1536</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.23</td>
<td>0.30</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.16</td>
<td>0.16</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>Event</th>
<th>Elapsed/CPU</th>
<th>Code/Data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDSA2</td>
<td></td>
<td>28.50</td>
<td>1.95</td>
<td>1536</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.71</td>
<td>0.28</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.23</td>
<td>0.23</td>
<td></td>
</tr>
</tbody>
</table>

Architecture Category: NORMATIVE_PERFORMANCE
E and V Criteria: EFFCY21 EFFCY22 EFFCY01 EFFCY26 EFFCY06 EFFCY29 & EFFCY26
Language Feature(s): BINARY_ADDS PRAGMA_SUPPRESS

<table>
<thead>
<tr>
<th>Name</th>
<th>Event</th>
<th>Elapsed/CPU</th>
<th>Code/Data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDSA3</td>
<td></td>
<td>28.99</td>
<td>2.10</td>
<td>1536</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.89</td>
<td>0.30</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.11</td>
<td>0.11</td>
<td></td>
</tr>
</tbody>
</table>

Architecture Category: NORMATIVE_PERFORMANCE
E and V Criteria: EFFCY21 EFFCY22 EFFCY01 EFFCY26
Language Feature(s): MULTIPLYING_OPS

<table>
<thead>
<tr>
<th>Name</th>
<th>Event</th>
<th>Elapsed/CPU</th>
<th>Code/Data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTA1</td>
<td></td>
<td>15.00</td>
<td>1.85</td>
<td>1536</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.20</td>
<td>0.28</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.08</td>
<td>0.08</td>
<td></td>
</tr>
</tbody>
</table>

C.15
many database responses have been deleted from this example.
There were a lot of them.

Valid choices are:
C  Collect names of statistics files.
S  Select attributes for reporting.
L  List selected attributes from files.
P  Print contents of current files to screen.
F  Save output to named file.
H  Re-display help prompt.
Q  Quit processing.

Please enter your choice:  c
COLLECT command accepted.

COMPILATION file is c.data
RUN_TIME file is r.data
INSTRUMENTATION file is i.data
Do you wish to change file(s)? [y\n]: y
COMPILATION file is c.data
Do you wish to change? [y\n]: y
  c.data  ... closed. File name?:
RUN_TIME file is r.data
Do you wish to change? [y\n]: y
  r.data  ... closed. File name?:
INSTRUMENTATION file is i.data
Do you wish to change? [y\n]: n

COMPILATION file is not defined.
RUN_TIME file is not defined.
INSTRUMENTATION file is i.data

Valid choices are:
C  Collect names of statistics files.
S  Select attributes for reporting.
L  List selected attributes from files.
P  Print contents of current files to screen.
F  Save output to named file.
H  Re-display help prompt.
Q  Quit processing.

Please enter your choice:  s
SELECT command accepted.
Do you wish to change query? [y\n]: y
Listing setting: LIST_BY_CATEGORY
Do you wish to change? [y\n]: y
Listing setting: LIST_BY_NAME
Do you wish to change display? [y\n]: y
Display options are:

Attribute: SHORT  Current value is: TRUE
Do you wish to change? [y\n]: n

C.16
Attribute: TEST_NAME
Current value is: FALSE
Do you wish to change? [y\n]: n

Attribute: DESCRIPTION
Current value is: FALSE
Do you wish to change? [y\n]: n

Attribute: ARCHITECTURE
Current value is: TRUE
Do you wish to change? [y\n]: y
... changed.

Attribute: E_AND_V
Current value is: TRUE
Do you wish to change? [y\n]: y
... changed.

Attribute: LANGUAGE_FEATURE
Current value is: TRUE
Do you wish to change? [y\n]: y
... changed.

Attribute: VERSION
Current value is: FALSE
Do you wish to change? [y\n]: n

Attribute: STATISTICS
Current value is: TRUE
Do you wish to change? [y\n]: n

Display options are:

Attribute: SHORT
Current value is: TRUE
Attribute: TEST_NAME
Current value is: TRUE
Attribute: DESCRIPTION
Current value is: FALSE
Attribute: ARCHITECTURE
Current value is: FALSE
Attribute: E_AND_V
Current value is: FALSE
Attribute: LANGUAGE_FEATURE
Current value is: FALSE
Attribute: VERSION
Current value is: FALSE
Attribute: STATISTICS
Current value is: TRUE

Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.

Please enter your choice: 1
LIST command accepted.
What is the test unit name? addsal
Name Event Elapsed/CPU Code/Data Comments
ADDSA1 STOP_REC 0.16 0.16

Do you want to list another unit? [y\n]: y
What is the test unit name? addsa2
Name Event Elapsed/CPU Code/Data Comments
ADDSA2 STOP_REC 0.23 0.23
Do you want to list another unit? [y/n]: n
Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.

Please enter your choice: f
SAVE command accepted.
File name?: listed
listed ... open.
What is the test unit name? addsa1
Do you want to list another unit? [y/n]: y
What is the test unit name? addsa2
Do you want to list another unit? [y/n]: n
listed ... closed.
Valid choices are:
C Collect names of statistics files.
S Select attributes for reporting.
L List selected attributes from files.
P Print contents of current files to screen.
F Save output to named file.
H Re-display help prompt.
Q Quit processing.

Please enter your choice: q
QUIT command accepted.
i.data ... closed.

Below is the file 'listed'.

Name Event   Elapsed/CPU Code/Data Comments
ADDSAl STOP_REC 0.16 0.16

Name Event   Elapsed/CPU Code/Data Comments
ADDSA2 STOP_REC 0.23 0.23

---
APPENDIX D : EXECUTION EXAMPLES

DATA GENERAL MV 10000:

Below is the test harness. Examples of the input and output files will follow. In the following command procedures,
 & is a line continuation character.
 %n% is a symbol for the n-th parameter passed to the command.
 [file] is an expansion symbol, so that this symbol is expanded to the contents of the named file, and
 :udd:f.n is a sample pathname for a file.

Note that naming a file as a command requests that the file name requested, plus a ".cli" on the end, is used as further input to the command line interpreter at that point in the command file.
The following is ":udd:benchmarks:harness_many.cli", the main command file. This command file creates the compilation statistics and run-time statistics files, in that order, and batches a job to generate the statistics. The batch command runs onto a second line, as indicated by the ":&".

create harness.out
create instr.dat
qbatch/qpri=101/qoutput=%1%.log/notify &
:udd:benchmarks:harness_a %1

The following is :udd:benchmarks:harness_a.cli, called from the file above. This file is useful only to insure that only one Ada compilation or execution is being created by this set of command files at a time.

:udd:benchmarks:harness [%1]

The following is :udd:benchmarks:harness.cli, which contains all the commands for a single test job. Harness first records the name of the test on the temporary statistics file (%1%.stat). Next it records the current (elapsed) time and the amount of CPU time used so far by this process. After compiling and linking the test file, the current time and amount of CPU time used are recorded again; the execution time statistics will
be derived from this information. After a fifteen second pause, the size of the produced load image is recorded on the temporary statistics file (by a little program called "HFORMAT", designed for that purpose). The collected statistics are then appended to the enduring statistics file, and the temporary compilation statistics file is deleted. Lastly, the test program is run, the statistics produced by the instrumentation package are appended to the enduring instrumentation statistics file, and the temporary statistics file is deleted.

```
write/l-%l%.stat %1%
runtime/l-%l%.stat
ada/main remakes:%1-%
adalink %1%
runtime/l-%l%.stat
pause 15
fi/length/nheader/l-%l%.stat %1%.pr
x :udd:benchmarks:hformat %1%.stat %1%.stats
copy/a harness.out %1%.stats
delete %1%.stats
x %1%
copy/a/1=warning/2=warning instr.dat instr
delete instr
```

Here is a sample input file to the test harness for the AOS/VS system. The test harness would be executed with the command "harness_many input.file" where the file "input.file" is shown below. The parentheses and ampersands are part of the AOS/VS command line syntax, which allow the processing of each listed test in turn.

```
(ADDSA1 &
ADDSA2 &
ADDSA3)
```

Below is the generated file "harness.out", which is the compilation statistics file for the input listed above.
Below is the generated file "instr.dat", which is the file containing the instrumentation statistics collected during the run for the input above. Note that the lines have been broken to fit in this manual; the "&" is used as a continuation character.

| ADDSA1 | START_REC | 0.000 | 0.000 &
|        | ADD PROGRAM, CONTROL VERSION - 10_000 ADDS |
| ADDSA1 | STOP_REC  | 0.301 | 0.301 |
| ADDSA2 | START_REC | 0.000 | 0.000 &
|        | ADD PROGRAM, W/O PRAGMAS - 10_000 ADDS |
| ADDSA2 | STOP_REC  | 0.400 | 0.400 |
| ADDSA3 | START_REC | 0.000 | 0.000 &
|        | ADD PROGRAM, WITH PRAGMAS - 10_000 ADDS |
| ADDSA3 | STOP_REC  | 0.201 | 0.201 |

D.3
VAX/VMS:

The following is 'USW:[BENCHMARK.WORK]HARNESS_MANY.COM', the main command file:

$!

This VAX/VMS command file loops through a file containing ADA source benchmark test file names and submits them to the test harness for the collection of the various statistics. For this implementation, this COM file must be submitted as a batch job.

The name of the file containing the test names is given as the first parameter to this command procedure.

The second parameter is the directory in which these tests must reside.

Set the default ADA library

acs set lib usw:[benchmark.mike.adalib]

Set the default directory to usw:[benchmark.work]

set def usw:[benchmark.work]

Create the three statistic files

create comp.dat
create instr.dat
create run.dat

Open the file with the test names

open/read in_file 'pl'

Loop through the file of tests, submitting each test to the harness for the collection of the various data.

loop:
read/end_of_file-done in_file test
@harness 'test' 'p2'
goto loop

At the end of the input file, close the file and terminate this command procedure.

done:
close in_file
write sys$output "All tests have been submitted for testing"

D.4
Below is the command file that is executed by 'HARNESS_MANY.COM'. This file's name is 'USW:[BENCHMARK.WORK]HARNESS.COM'.

This VAX/VMS command file performs functions necessary to collect various data about ADA source test files.

These data are put into the files 'comp.dat' (compilation statistics), 'instr.dat' (instrumentation statistics) and 'run.dat' (run-time statistics).

Record the current elapsed and cpu times (before compilation)

beg_cpu_time = f$getjpi(""."cputim")
beg_time = f$time()

Compile and link the test

ada/nocopy_source 'p2''p1'
a cs link 'p1'

Record the current elapsed and cpu times (after compilation)

end_cpu_time = f$getjpi(""."cputim")
end_time = f$time()

'file' => file_spec of the object file created by the compilation

file = "usw:[benchmark.mike.adalib]" + pl + ".obj"

Calculate the number of bytes in the object file

blocks_used = f$file_attributes(file,"eof")
block_size = f$file_attributes(file,"bls")
file_size = blocks_used * block_size

Calculate elapsed cpu time (in hundredths of seconds)

cpu_time = (end_cpu_time - beg_cpu_time)

Divide the elapsed cpu time into seconds and hundredths-seconds

cpu_time_secs = cpu_time / 100
cpu_time_hundsecs = cpu_time - 100 * cpu_time_secs
Calculate elapsed time

The CON file 'calc_elapsed_time' takes the 'begin_time' and 'end_time' and returns the elapsed time in its seconds and hundredths seconds parts. These values are placed in the global symbol table and have symbol names 'elapsed_time_secs' and 'elapsed_time_hundsecs'.

@calc_elapsed_time "'begin_time'" "'end_time'"

Put the compilation statistics in one output line

out_line = "'pl' 'elapsed_time_secs'" + - "'elapsed_time_hundsecs'" + - "'cpu_time_secs' 'cpu_time_hundsecs' 'file_size'"

Append the output line onto the file

open/append comp comp.dat
write comp out_line
close comp

Record the current elapsed and cpu times (before execution)

beg_cpu_time = f$getjpi("","cputim")
beg_time = f$time()

Run the executable file

run 'pl'.exe

Record the current elapsed and cpu times (after execution)

end_cpu_time = f$getjpi("","cputim")
end_time = f$time()

Append the instrumentation statistics to instr.dat

append instr.; instr.dat

Calculate elapsed cpu time (in hundredths of seconds)

cpu_time = (end_cpu_time - beg_cpu_time)

divide the elapsed cpu time into seconds and hundredths-seconds

cpu_time_secs = cpu_time / 100

cpu_time_hundsecs = cpu_time - 100 * cpu_time_secs
Calculate elapsed time

@calc_elapsed_time "'beg_time'" "'end_time'"

Put the available execution statistics in one output line

out_line="'pl' 'elapsed_time_secs'." + ~ 
"'elapsed_time_hundsecs'" + ~ 
"'cpu_time_secs'.'cpu_time_hundsecs' 512 512"

Append the output line onto the file

open/append run run.dat
write run out-line
close run

Delete unnecessary files

del instr.;
delete 'pl'.exe.*
acs delete unit 'pl'

Here is the file 'USW:[BENCHMARK.WORK]CALC_ELAPSED_TIME.COM':

Calculate the elapsed time between the beginning time and the ending time (the first and second parameters). The times are given in the VAX/VMS format. The two times are assumed to be less than 24 hours apart!

Dissect the beginning time (first parameter) into time units

beg_hs = f$extract(21,2,pl) ! hundredths of seconds
beg_sec = f$extract(18,2,pl) ! seconds
beg_min = f$extract(15,2,pl) ! minutes
beg_hr = f$extract(12,2,pl) ! hours

Dissect the ending time (second parameter) into time units

end_hs = f$extract(21,2,p2) ! hundredths of seconds
end_sec = f$extract(18,2,p2) ! seconds
end_min = f$extract(15,2,p2) ! minutes
end_hr = f$extract(12,2,p2) ! hours

Convert each of the times to hundredths of seconds

D.7
$ beg_time = -
  (((beg_hr * 60) + beg_min) * 60 + beg_sec) * 100 + beg_hs
$ end_time = -
  (((end_hr * 60) + end_min) * 60 + end_sec) * 100 + end_hs
$

$! Get the total elapsed time ( in hundredths of seconds )
$!
$ total_time = end_time - beg_time
$!
$! If the total time is negative then the beginning and end
$! times were on different days.
$!
$! if total_time .lt. 0 then -
total_time = (((24 * 60) * 60) * 100) + end_time - beg_time
$!
$! Separate the elapsed time into seconds and hundredths of
$! seconds
$!
$ total_time_secs = total_time / 100
$ total_time_hundsecs = total_time - total_time_secs * 100
$!
$! Put the elapsed time in the global symbol table, so that
$! it can be accessed.
$!
$ elapsed_time_secs = total_time_secs
$ elapsed_time_hundsecs = total_time_hundsecs
$!
$ exit

---

Here is an example of an input file to the 'HARNESS_MANY' COM file:

----
CAPAA1
CAPAA2
NRPCA1
NRPCA2
----

The format of this file must be as given above in order to
use these COM files. No file type is given on the file names;
however, they must be of type '.ADA' in their resident directory.

The main COM file would be executed with the command

"$ submit HARNESS_MANY/parameters=(file_spec, dir_spec) -
/queue-big"

where 'file_spec' is the VAX/VMS file specification of the file
containing the names of the tests and 'dir_spec' is the directory specification of the directory containing the tests.

Here is an example of executing the main COM file, and the system messages received:

```
$ submit harness_many/parameters=(new.1st,usw:{benchmark.new}) -
      /queue=big

Job HARNESS_MANY (queue BIG entry 355) started on BIG

Job HARNESS_MANY (queue BIG entry 355) completed
```

Below is the file 'comp.dat', which is the compilation statistics file generated for the input file above.

```
CAPAA1 22.76  3.64 3584
CAPAA2 19.6  3.85 3584
NRPCA1 20.26  3.73 4096
NRPCA2 19.28  3.79 4096
```

Below is the generated file 'run.dat' with the run-time statistics for the input file given above. NOTE: The last two numbers do not mean anything; they are there simply to make the format of the execution statistics file what is expected by the Report Writer.

```
CAPAA1 11.95  0.21 512  512
CAPAA2 12.23  0.24 512  512
NRPCA1 11.15  0.24 512  512
NRPCA2 10.93  0.20 512  512
```

Below is the file 'instr.dat', containing instrumentation statistics generated for the input file above. Lines too long to fit in the manual are denoted by the '继续', and continued on the next line.
CAPAA1 START_REC 0.0000 0.0000 &
Constrained Array Param. Assoc. w/3 elements (Control)
CAPAA1 STOP_REC 0.0300 0.0300
CAPAA2 START_REC 0.0000 0.0000 &
Constrained Array Param. Assoc. w/3 elements (test)
CAPAA2 STOP_REC 0.0300 0.0300
NRPCA1 START_REC 0.0000 0.0000 &
Nested Recursive Procedure Call (Control)
NRPCA1 STOP_REC 0.0500 0.0500
NRPCA2 START_REC 0.0000 0.0000 &
Nested Recursive Procedure Call (Test)
Distribution List for IDA Paper P-1879

Sponsors
Ms. Virginia Castor
Director
Ada Joint Program Office
1211 Fern St., C-107
Arlington, VA 22202

LCDR Phil Meyers
Navy Deputy Director
Ada Joint Program Office
1211 Fern St., C-107
Arlington, VA 22202

Ray Szymanoski
AFWAL/AAAF-2
Wright-Patterson AFB, OH 45433-6503

Others
Mr. Michael Vilot
General Systems Group
51 Main St.
Salem, NH 03079

Dr. Gregory Riccardi
Room 206, Lone Building
Department of Computer Science
Florida State University
Tallahassee, FL 32306

Defense Technical Information Center (2 copies)
Cameron Station
Alexandria, VA 22314

CSED Review Panel
Dr. Dan Alpert, Director
Center for Advanced Study
University of Illinois
912 W. Illinois Street
Urbana, Illinois 61801

Dr. Barry W. Boehm
TRW Defense Systems Group
MS 2-2304
One Space Park
Redondo Beach, CA 90278
Dr. Ruth Davis
The Pymatuning Group, Inc.
2000 N. 15th Street, Suite 707
Arlington, VA 22201

Dr. Larry E. Druffel
Rational Machines
1501 Salado Drive
Mountain View, CA 94043

Mr. Neil S. Eastman, Manager
Software Engineering & Technology
IBM Federal Systems Division
6600 Rockledge Drive
Bethesda, MA 20817

Admiral Noel Gayler, USN, Retired
1250 S. Washington St.
Alexandria, VA 22314

Dr. C.E. Hutchinson, Dean
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755

Mr. Oliver Selfridge
45 Percy Road
Lexington, MA 02173

Dr. Harrison Shull, Chancellor
University of Colorado
Campus Box B-17
301 Regent Administration Center
Boulder, CO 80309

Dr. Robert L. Sproull
President Emeritus
University of Rochester
Rochester, NY 14627

IDA

Mr. Seymour Deitchman, HQ
Mr. Robin Pirie, HQ
Dr. Thomas H. Probert, CSED
Dr. Jack Kramer, CSED
Dr. John Salasin, CSED
Ms. Audrey A. Hook, CSED (2 copies)
Mr. Stephen Welke, CSED
Ms. Katydean Price (2 copies)
IDA Control & Distribution Vault (25 copies)