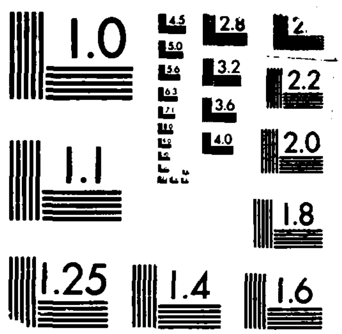
AD-A193 506 ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT: 1/1
ROCKWELL INTERNATION..(U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI.. 23 JUN 87
UNCLASSIFIED AVF-VSR-112.0787 F/G 12/5 NL



MICROCOPY RESOLUTION TEST CHART
UREAU OF STANDARDS-1963-A

(2)

AVF Control Number: AVF-VSR-112.0787
87-01-21-RWL

Ada® COMPILER
VALIDATION SUMMARY REPORT:
Rockwell International
DDC-Based Ada/CAPS Compiler, 1.0
VAX-11/8650 host and CAPS/AAMP target

Completion of On-Site Testing:
23 June 1987

Prepared By:
Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: Rockwell International DDC-Based Ada/CAPS Compiler, 1.0 VAX-11/8650 host and CAPS/AAMP target	5. TYPE OF REPORT & PERIOD COVERED 23 June'87 to 23 June'88
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson AFB OH 45433-6503	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson AFB OH 45433-6503	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081ASD/SIOL	12. REPORT DATE 23 June 1987
	13. NUMBER OF PAGES 37p.
14. MONITORING AGENCY NAME & ADDRESS *(If different from Controlling Office)* Wright-Patterson AFB OH 45433-6503.	15. SECURITY CLASS *(of this report)* UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See Attached.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

```
+++++++++++++++++++++++
+                     +
+ Place NTIS form here +
+                     +
+++++++++++++++++++++++
```

Ada® Compiler Validation Summary Report:

Compiler Name: DDC-Based Ada/CAPS Compiler, 1.0

Host:	Target:
VAX-11/8650 under VMS, Version 4.5	CAPS/AAMP (bare machine)

Testing Completed 23 June 1987 Using ACVC 1.8

This report has been reviewed and is approved.

Ada Validation Facility
Steven P. Wilson
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the DDC-Based Ada/CAPS Compiler, 1.0, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The DDC-Based Ada/CAPS Compiler is hosted on a VAX-11/8650 operating under VMS, Version 4.5. Programs processed by this compiler may be executed on a CAPS/AAMP (bare machine).

On-site testing was performed 22 June 1987 through 23 June 1987 at 400 Collins Road NE, Cedar Rapids Iowa, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2138 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 242 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2138 tests were processed, results for Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 179 of the processed tests determined to be inapplicable. The remaining 1959 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER 2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	96	223	296	246	161	97	136	261	128	32	218	65	1959
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	20	102	124	1	0	0	3	1	2	0	0	168	421
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any unsupported language constructs required by the Ada Standard

- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 22 June 1987 through 23 June 1987 at 400 Collins Road NE, Cedar Rapids Iowa.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler Versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983.

2. Ada Validation Organization: Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable test — A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test — A test for which a compiler generates the expected result.

Target — The computer for which a compiler generates code.

Test — A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn test — A test found to be incorrect and not used to check conformity to the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation.

Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: DDC-Based Ada/CAPS Compiler, 1.0

ACVC Version: 1.8

Certificate Number: 870601W1.08061

Host Computer:

Machine:	VAX-11/8650
Operating System:	VMS, Version 4.5
Memory Size:	16 megabytes

Target Computer:

Machine:	CAPS/AAMP (bare machine)
Memory Size:	256K words

Communications Network: Ethernet

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation rejects such calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, LONG_INTEGER, and LONG_FLOAT in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR. (See test E24101A.)

- Array types.

  An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

  A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST CONSTRAINT_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises CONSTRAINT_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are not evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Functions.

  An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declaration, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declaration. (See test E66001D.)

- Representation clauses.

  The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation accepts 'STORAGE_SIZE for collections or tasks; it rejects 'SIZE and 'SMALL clauses. Enumeration representation clauses appear not to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

- Pragmas.

  The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests CA3004E and CA3004F.)

- Input/output.

  The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. However, the target has no file system. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

- Generics.

  Generic subprogram declarations and bodies cannot be compiled in separate compilations. (See test CA2009F.)

  Generic package declarations and bodies cannot be compiled in separate compilations. (See tests CA2009C and BC3205D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of the DDC-Based Ada/CAPS Compiler was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 421 tests were inapplicable to this implementation, and that the 1959 applicable tests were passed by the implementation.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	866	952	15	12	46	1959
Failed	0	0	0	0	0	0	0
Inapplicable	1	1	416	2	1	0	421
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER 2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	96	223	296	246	161	97	136	261	128	32	218	65	1959
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	20	102	124	1	0	0	3	1	2	0	0	168	421
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 426 tests were inapplicable for the reasons indicated:

- C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.

- D4A002B and D4A004B use 64-bit integer calculations which are not supported by this compiler.

- C55B16A makes use of an enumeration representation clause containing noncontiguous values which is not supported by this compiler.

- B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

- C87B62A and C87B62C use length clauses which are not supported by this compiler. The length clause is rejected during compilation.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

- CA2009C and CA2009F compile the body and subunits of a generic unit in separate compilation files. Separate compilation of generic specifications and bodies is not supported by this compiler.

- The following 168 tests are inapplicable because DIRECT_IO, SEQUENTIAL_IO, and TEXT_IO are not supported by this implementation:

CE2102C	CE3108A..B (2 tests)	CE3411C
CE2102G	CE3109A	CE3412A
CE2104A..D (4 tests)	CE3110A	CE3412C
CE2105A	CE3111A..E (5 tests)	CE3413A
CE2106A	CE3112A..B (2 tests)	CE3413C
CE2107A..F (6 tests)	CE3114A..B (2 tests)	CE3602A..D (4 tests)
CE2108A..D (4 tests)	CE3115A	CE3603A
CE2109A	CE3201A	CE3604A
CE2110A..C (3 tests)	CE3202A	CE3605A..E (5 tests)
CE2111A..E (5 tests)	CE3203A	CE3606A..B (2 tests)
CE2111G..H (2 tests)	CE3208A	CE3704A..B (2 tests)
CE2201A..F (6 tests)	CE3301A..C (3 tests)	CE3704D..F (3 tests)
CE2204A..B (2 tests)	CE3302A	CE3704M..O (3 tests)
CE2210A	CE3305A	CE3706D
CE2401A..F (6 tests)	CE3402A..D (4 tests)	CE3706F
CE2404A	CE3403A..C (3 tests)	CE3804A..E (5 tests)
CE2405B	CE3403E..F (2 tests)	CE3804G
CE2406A	CE3404A..C (3 tests)	CE3804I
CE2407A	CE3405A..D (4 tests)	CE3804K
CE2408A	CE3406A..D (4 tests)	CE3804M
CE2409A	CE3407A..C (3 tests)	CE3805A..B (2 tests)
CE2410A	CE3408A..C (3 tests)	CE3806A
AE3101A	CE3409A	CE3806D..E (2 tests)
CE3102B	CE3409C..F (4 tests)	CE3905A..C (3 tests)
EE3102C	CE3410A	CE3905L
CE3103A	CE3410C..F (4 tests)	CE3906A..C (3 tests)
CE3104A	CE3411A	CE3906E..F (2 tests)
CE3107A		

- The following 242 tests require a floating-point accuracy that exceeds the maximum of 9 supported by the implementation:

C24113F..Y (20 tests)	C35705F..Y (20 tests)
C35706F..Y (20 tests)	C35707F..Y (20 tests)
C35708F..Y (20 tests)	C35802F..Y (20 tests)
C45241F..Y (20 tests)	C45321F..Y (20 tests)
C45421F..Y (20 tests)	C45424F..Y (20 tests)
C45521F..Z (21 tests)	C45621F..Z (21 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for five Class B tests:

B33301A	B55A01A	B67001A
B67001C	B67001D	

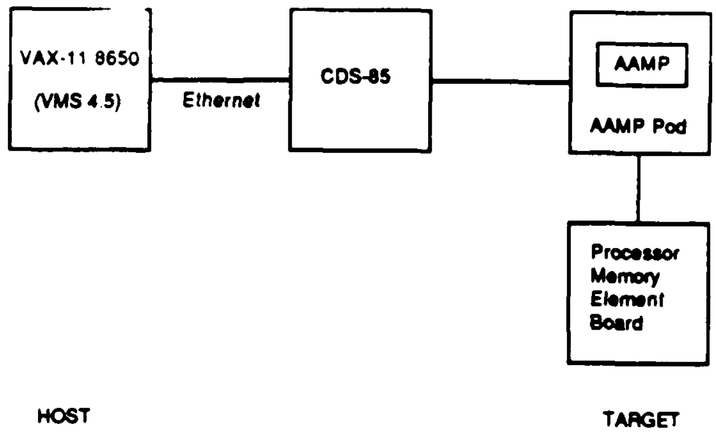
3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by the DDC-Based Ada/CAPS Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and that the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the DDC-Based Ada/CAPS Compiler using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX-11/8650 host operating under VMS, Version 4.5, and an Advanced Architecture Microprocessor (AAMP) target (see figure below). A CDS-85 Computer Development Station was used to facilitate running the executable tests. An executable image for each test was downloaded from the VAX to the CDS-85 using Ethernet. An Ada Symbolic Debugger, Version 3.3, was used to load each image into CDS-85 memory from which the program was executed by the AAMP. The Processor Memory Element Board provided the clock used by the AAMP. Test output was captured by the CDS-85.



A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring splits during the prevalidation testing were included in their split form on the magnetic tape.

The body of package REPORT was modified to use a package SIMPLE_IO instead of TEXT_IO because package TEXT_IO is implemented in such that a way an exception is raised for all file operations. A set of executable tests was run to verify that the modified body of package REPORT operated correctly.

The contents of the magnetic tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled on the VAX-11/8650, and all executable tests were run on the CAPS/AAMP. Object files were linked on the host computer, and executable images were transferred to the target computer. The transferred executable images did not include those portions of the run-time system that are identical for every test. The run-time system was loaded once for each chapter and used by each test. This had the effect of significantly reducing the time needed for downloading the tests. Results were printed from the host computer, with results being transferred to the host computer via Ethernet.

TEST INFORMATION

The compiler was tested using command scripts provided by Rockwell International and reviewed by the validation team. Default options were in effect for testing, except for the following:

Option	Effect
/LIST	Generate a source listing
/NODEBUG	Suppress debugger information
/NOOBJECT	Suppress object code generation (for Class B and L tests)

Tests were compiled, linked, and executed (as appropriate) using 1 host computer and 1 target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

The validation team arrived at 400 Collins Road NE, Cedar Rapids Iowa on 22 June 1987, and departed after testing was completed on 23 June 1987.

APPENDIX A

DECLARATION OF CONFORMANCE

Rockwell International has submitted the following declaration of conformance concerning the DDC-Based Ada/CAPS Compiler.

DECLARATION OF CONFORMANCE

Compiler Implementor: Rockwell International Corporation
Ada® Validation Facility: ASD/SCOL, Wright-Patterson AFB, OH
Ada Compiler Validation Capability (ACVC) Version: 1.8

Base Configuration

Base Compiler Name: DDC-Based Ada/CAPS Compiler Version: 1.0
Host Architecture ISA: VAX-11/8650 OS&VER #: VMS, VERSION 4.5
Target Architecture ISA: CAPS/AAMP (bare machine)

Implementor's Declaration

I, the undersigned, representing Rockwell International Corporation, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that Rockwell International Corporation is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

D. R. Stover Date: June 23, 1987

Rockwell International Corporation
Don R. Stover, Manager
Computer Support Systems Section

Owner's Declaration

I, the undersigned, representing Rockwell International Corporation, take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

D. R. Stover Date: June 23, 1987

Rockwell International Corporation
Don R. Stover, Manager
Computer Support Systems Section

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the DDC-Based Ada/CAPS Compiler, 1.0, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -32 768 .. 32 767;
     type SHORT_INTEGER is range -128 .. 127;
     type LONG_INTEGER is range -2 147 483 648 .. 2 147 483_647;

     type FLOAT is digits 6 range -1.701411E38 .. 1.701411E38;
     type LONG_FLOAT is digits 9
              range -1.701411834E38 .. 1.701411834E38;

     type DURATION is delta 0.0001 range -131 072.0 .. 131 071.0;

     ...

end STANDARD;
```

APPENDIX F

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

This appendix describes the implementation-dependent characteristics of the DDC-Based Ada/CAPS Compiler.

F.1 Implementation-Dependent Pragmas.

No implementation-dependent pragmas are supported.

F.2 Implementation-Dependent Attributes.

No implementation-dependent attributes are supported.

F.3 Specification Of The Package SYSTEM.

```
package SYSTEM is

  type ADDRESS        is range 0..16#FF_FFFF#    -- 24 bit address
  subtype PRIORITY    is INTEGER range 0 .. 254;
  type NAME           is (VAX11,   AAMP,  CAPS6,  CAPS7,
                            CAPS8, CAPS10,  ACAPS);
  SYSTEM_NAME:        constant NAME    := AAMP;
  STORAGE_UNIT:       constant         := 16;
  MEMORY_SIZE:        constant         := 16_384 * 1024;
  MIN_INT:            constant         := -2_147_483_647-1;
  MAX_INT:            constant         := 2_147_483_647;
  MAX_DIGITS:         constant         := 9;
  MAX_MANTISSA:       constant         := 31;
  FINE_DELTA:         constant         := 2#1.0#E-30;
  TICK:               constant         := 0.000_1;

end SYSTEM;
```

F.4 Representation Clause Restrictions.

F.4.1 Representation Clauses.

In general, no representation clauses may be given for a derived type. The representation clauses that are allowed for non-derived types are described in the following sections.

F.4.2 Length Clauses.

The compiler accepts only length clauses that specify the number of storage units reserved for a collection or a task activation.

F.4.3 Enumeration Representation Clauses.

Enumeration representation clauses are not supported.

F.5 Implementation-Generated Names.

Implementation-generated names for implementation-dependent components are not supported.

F.6 Address Clause Expressions.

All address values are treated as the address of a 16 bit word of memory, even for code addresses which are normally thought of as 8 bit (byte) addresses. All subprogram and task entry addresses are word aligned by the compiler. The address clause for an interrupt entry is not supported.

F.7 Unchecked Conversion Restrictions.

Unchecked conversion is only allowed between values of the same size.

F.8 I/O Package Implementation-Dependent Characteristics.

The target environment does not support a file system; therefore I/O procedure or function calls involving files will raise predefined exceptions.The I/O exceptions raised will be as follows for the subprograms in the packages TEXT_IO, SEQUENTIAL_IO, and DIRECT_IO:

Subprogram	Exception
CREATE	USE_ERROR
OPEN	USE_ERROR
IS_OPEN	none - always returns FALSE

All other subprograms will raise STATUS_ERROR.

F.8.1 Package LOW_LEVEL_IO.

Package LOW_LEVEL_IO is not provided.

F.9 Other Implementation-Dependent Features.

F.9.1 Predefined Types.

This section describes the implementation-dependent predefined types declared in the predefined package STANDARD, and the relevant attributes of these types.

F.9.1.1 Integer Types.

Three predefined integer types are implemented, SHORT_INTEGER, INTEGER, and LONG_INTEGER. They have the following attributes:

```
SHORT_INTEGER'FIRST     = -128
SHORT_INTEGER'LAST      =  127
SHORT_INTEGER'SIZE      =   16

INTEGER'FIRST           = -32768
INTEGER'LAST            =  32767
INTEGER'SIZE            =     16

LONG_INTEGER'FIRST      = -2_147_483_648
LONG_INTEGER'LAST       =  2_147_483_647
LONG_INTEGER'SIZE       =             32
```

F.9.1.2 Floating Point Types.

Two predefined floating point types are implemented, FLOAT and LONG_FLOAT. They have the following attributes:

```
FLOAT'DIGITS              =  6
FLOAT'EMAX                =  84
FLOAT'EPSILON             =  16#0.1000_000#E-04
                         ~=  9.53674E-07
FLOAT'FIRST               = -16#0.7FFF_FF8#E+32
                         ~= -1.70141E+38
FLOAT'LARGE               =  16#0.FFFF_F80#E+21
                         ~=  1.93428E+25
FLOAT'LAST                =  16#0.7FFF_FF8#E+32
                         ~=  1.70141E+38
FLOAT'MACHINE_EMAX        =  127
FLOAT'MACHINE_EMIN        = -127
FLOAT'MACHINE_MANTISSA    =  24
FLOAT'MACHINE_OVERFLOWS   =  TRUE
FLOAT'MACHINE_RADIX       =  2
FLOAT'MACHINE_ROUNDS      =  TRUE
FLOAT'MANTISSA            =  21
FLOAT'SAFE_EMAX           =  127
FLOAT'SAFE_LARGE          =  16#0.7FFF_FC0#E+32
                         ~=  1.70141E+38
FLOAT'SAFE_SMALL          =  16#0.1000_000#E-31
                         ~=  2.93874E-39
FLOAT'SIZE                =  32
FLOAT'SMALL               =  16#0.8000_000#E-21
                         ~=  2.58494E-26


LONG_FLOAT'DIGITS              =  9
LONG_FLOAT'EMAX                =  124
LONG_FLOAT'EPSILON             =  16#0.4000_0000_000#E-7
                              ~=  9.31322575E-10
LONG_FLOAT'FIRST               = -16#0.7FFF_FFFF_FF8#E+32
                              ~= -1.70141183E+38
LONG_FLOAT'LARGE               =  16#0.FFFF_FFFE_000#E+31
                              ~=  2.12676479E+37
LONG_FLOAT'LAST                =  16#0.7FFF_FFFF_FF8#E+32
                              ~=  1.70141183E+38
LONG_FLOAT'MACHINE_EMAX        =  127
LONG_FLOAT'MACHINE_EMIN        = -127
LONG_FLOAT'MACHINE_MANTISSA    =  40
LONG_FLOAT'MACHINE_OVERFLOWS   =  TRUE
LONG_FLOAT'MACHINE_RADIX       =  2
LONG_FLOAT'MACHINE_ROUNDS      =  TRUE
```

LONG_FLOAT'MANTISSA	=	31
LONG_FLOAT'SAFE_EMAX	=	127
LONG_FLOAT'SAFE_LARGE	=	16#0.7FFF_FFFF_000#E+32
	~=	1.70141183E+38
LONG_FLOAT'SAFE_SMALL	=	16#0.1000_0000_000#E-31
	~=	2.93873588E-39
LONG_FLOAT'SIZE	=	48
LONG_FLOAT'SMALL	=	16#0.8000_0000_000#E-31
	~=	2.35098870E-38

F.9.1.3 Fixed Point Types.

To implement fixed point numbers, Ada/CAPS uses two sets of anonymous, predefined, fixed point types, here named FIXED and LONG_FIXED. These names are not defined in package STANDARD, but are only used here for reference.

These types are of the following form:

```
type FIXED_TYPE is delta SMALL range -M*SMALL .. (M-1)*SMALL;

where SMALL = 2**n for -127 <= n <= 127,
and M = 2**15 for FIXED, or M = 2**31 for LONG_FIXED.
```

For each of FIXED and LONG_FIXED there exists a virtual predefined type for each possible value of SMALL (cf. RM 3.5.9).

A user defined fixed point type is represented as that predefined FIXED or LONG_FIXED type which has the largest value of SMALL not greater than the user-specified DELTA, and which has the smallest range that includes the user-specified range.

As the value of SMALL increases, the range increases. In other words, the greater the allowable error (the value of SMALL), the larger the allowable range.

Example 1:

For a FIXED type, to get the smallest amount of error possible requires SMALL = 2**-127, but the range is constrained to:
    -2**-122 .. ((2**-122) - (2**-127)).

Example 2:

For a FIXED type, to get the largest range possible requires SMALL = 2**127, i.e., the error may be as large as 2**127. The range is then:
    -2**132 .. ((2**132) - (2**127)).

For any FIXED or LONG_FIXED type T:

T'MACHINE_OVERFLOWS	=	TRUE
T'MACHINE_ROUNDS	=	FALSE

F.9.1.4 The Type DURATION.

The predefined fixed point type DURATION has the following attributes:

```
DURATION'AFT             =  4
DURATION'DELTA           =  0.0001
DURATION'FIRST           = -131_072.0000
DURATION'FORE            =  7
DURATION'LARGE           =  131_071.999938965
                         =  2#1.0#E+17 - 2#1.0E-14
DURATION'LAST            =  DURATION'LARGE
DURATION'MANTISSA        =  31
DURATION'SAFE_LARGE      =  DURATION'LARGE
DURATION'SAFE_SMALL      =  DURATION'SMALL
DURATION'SIZE            =  32
DURATION'SMALL           =  6.103515625E-5
                         =  2#1.0#E-14
```

F.9.2 Uninitialized Variables.

There is no check on the use of uninitialized variables. The effect of a program that uses the value of such a variable is undefined.

F.9.3 Package MACHINE_CODE.

Machine code insertions (cf. RM 13.8) are supported by the Ada/CAPS compiler via the use of the predefined package MACHINE_CODE.

```
package MACHINE_CODE is

   type CODE is record
      INSTR: STRING (1 .. 80);
   end record;

end MACHINE_CODE;
```

Machine code insertions may be used only in a procedure body, and only if the body contains nothing but code statements, as in the following example:

```
with MACHINE_CODE;   -- Must apply to the compilation unit
                     -- containing DOUBLE.

procedure DOUBLE (VALUE: in INTEGER; DOUBLE_VALUE: out INTEGER);

procedure DOUBLE (VALUE: in INTEGER; DOUBLE_VALUE: out INTEGER) is

begin

  MACHINE_CODE.CODE' (INSTR => "REFSL 1");  -- Get VALUE.
```

```
  MACHINE_CODE.CODE' (INSTR => "DUP");       -- Make copy of VALUE.
  MACHINE_CODE.CODE' (INSTR => "LOCX");      -- Add copies together.
  MACHINE_CODE.CODE' (INSTR => "ASNSL 0");   -- Store result in DOUBLE_VALUE

end DOUBLE;
```

The string value assigned to INSTR may be a CAPS assembly language instruction or macro.

F.9.4 Compiler Limitations.

The following limitations apply to Ada programs in the DDC-Based Ada/CAPS Compiler System:

- A program (sum of all compilation units) may not contain more than 64K words of static data and stacks. Static data is allocated for variables declared in the specification or body of a package. A stack is allocated for each task including the main program. Some of the 64K maximum is used by the runtime system. Static data requirements exceeding the 64K word maximum may be permanently allocated to the heap at the cost of an additional indirect memory access.

- A compilation unit may not contain more than 64K bytes (32K words) of code.

- A compilation unit may not contain more than 32K words of data.

- A compilation unit may not contain more than 32K words of constants.

- It follows that any single object may be no larger than 32K words.

- No more than 500 subprograms may be defined in a single compilation unit, including any implicitly allocated by the compiler.

- The maximum nesting level for blocks is 100.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1 .. 125 => 'A', 126 => '1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1 .. 125 => 'A', 126 => '2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1 .. 62 => 'A', 63 => '3', 64 .. 126 => 'A')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1 .. 62 => 'A', 63 => '4', 64 .. 126 => 'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1 .. 123 => '0', 124 .. 126 => '298')

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be the size of the maximum line length.	(1 .. 120 => '0', 121 .. 126 => '69.0E1')
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1 .. 106 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	35
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters, or is too long if no invalid characters exist.	X}]!@#$^&~Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character, or is too long if no wild card character exists.	XYZ*
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in the range of DURATION.	76_536.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	10_000_000.0

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	BAD-CHARACTER*^
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	MUCH-TOO-LONG-NAME-FOR-A-FILE
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2_147_483_647 - 1
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2_147_483_647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-76_536.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-10_000_000.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	9
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	126
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647

TEST PARAMETERS

Name and Meaning	Value
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	LONG_LONG_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- C32114A: An unterminated string literal occurs at line 62.

- B33203C: The reserved word "IS" is misspelled at line 45.

- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conVersions.

- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.

- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.

- C41404A: The values of 'LAST and 'LENGTH are incorrect in the <u>if</u> statements from line 74 to the end of the test.

- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.

- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.

- B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and <u>end</u> <u>case</u>; is missing from line 42.

- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

- B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

- C87B50A: The call of "/=" at line 31 requires a use clause for package A.

- C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

- C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- BC3204C: The body of BC3204C0 is missing.

END
DATE
FILMED
7-88
DTIC