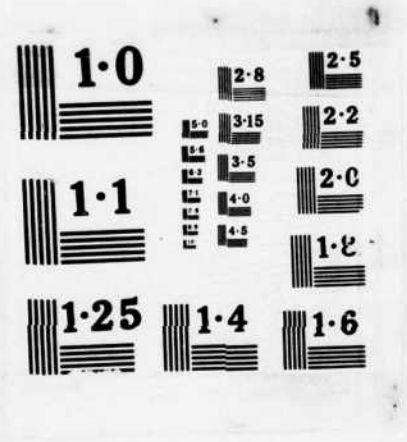
AD-A204 897 ADA COMPILER VALIDATION SUMMARY REPORT: CERTIFICATE NUMBER: 880708W109127.. (U) INFORMATION SYSTEMS AND TECHNOLOGY CENTER W-P AFB OH ADA VALI.. 14 JUL 88 1/1

UNCLASSIFIED AVF-VSR-205.1088 F/G 12/5 NL


1·0
1·1
1·25
1·4
1·6
2·8
3·15
3·5
4·0
4·5
2·5
2·2
2·0
1·8

(2)

AVF Control Number: AVF-VSR-205.1088
88-04-14-ICC

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 880708W1.09127
Irvine Compiler Corporation
ICC Ada, Release 5.0
HP 9000 Model 825

Completion of On-Site Testing:
14 July 1988

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

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

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: Irvine Computer Corporation, ICC Ada, Release 5.0, HP 9000 Model 825 (Host and Target). (880708 W1. 09127)	5. TYPE OF REPORT & PERIOD COVERED 14 July 1988 to 14 July 1989 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson Air Force Base, Dayton, Ohio, U.S.A.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson Air Force Base, Dayton, Ohio, U.S.A.	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 20301-3081	12. REPORT DATE 14 July 1988 13. NUMBER OF PAGES 38 p.
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* Wright-Patterson Air Force Base, Dayton, Ohio, U.S.A.	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)*

ICC Ada, Release 5.0, Irvine Computer Corporation, Wright-Patterson Air Force Base, HP 9000 Model 825 under HP-UX, Release 2.0 (Host and (Target), ACVC 1.9.

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)*

Ada Compiler Validation Summary Report:

Compiler Name: ICC Ada, Release 5.0

Certificate Number: 880708W1.09127

Host:	Target:
HP 9000 Model 825 under HP-UX, Release 2.0	HP 9000 Model 825 under HP-UX, Release 2.0

Testing Completed 14 July 1988 Using ACVC 1.9

This report has been reviewed and is approved.

Steven P. Wilson

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

John F. Kramer

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

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 the characteristics of particular operating systems, hardware, or implementation strategies. All 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.

INTRODUCTION

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 language constructs not supported by the compiler but 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 procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 14 July 1988 at Irvine Compiler Corporation, Irvine, California.

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/SCEL
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 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
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	An ACVC 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	An ACVC 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	An ACVC 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 a combination of 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	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. 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 compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. 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 REPORT and CHECK_FILE 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 this 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: ICC Ada, Release 5.0

ACVC Version: 1.9

Certificate Number: 880708W1.09127

Host Computer:

Machine:	HP 9000 Model 825
Operating System:	HP-UX, Release 2.0
Memory Size:	24 Megabytes

Target Computer:

Machine:	HP 9000 Model 825
Operating System:	HP-UX, Release 2.0
Memory Size:	24 Megabytes

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. The tests demonstrate the following characteristics:

- 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 processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, and TINY_INTEGER 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 during execution. (See test E24101A.)

- Expression evaluation.

  Apparently some default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

  Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Apparently NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

. Rounding.

The method used for rounding to integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

. 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. For this implementation:

Declaration of an array type or subtype with more than SYSTEM.MAX_INT components raises no exception. (See test C36003A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises STORAGE_ERROR when the array objects are declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (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 no exception. (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 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.)

- 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, index subtype checks appear to be made as choices are evaluated. (See tests C43207A and C43207B.)

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

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

- Representation clauses.

  An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

For this implementation:

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are not supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are not supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are not supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are not supported. (See tests A39005E and C87B62C.)

Record representation clauses are supported. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

- Pragmas.

The pragma INLINE is not supported for procedures and functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D and CE2102E.)

Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I, and CE2102J.)

RESET and DELETE are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2106A and CE2106B.)

Overwriting to a sequential file does not truncate the file. (See test CE2208B.)

An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and can be created in IN_FILE mode. (See test EE3102C.)

More than one internal file can be associated with each external file for text I/O for both reading and writing. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

More than one internal file can be associated with each external file for sequential I/O for both reading and writing. (See tests CE2107A..D (4 tests), CE2110B, and CE2111D.)

More than one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107F..I (5 tests), CE2110B, and CE2111H.)

An internal sequential access file and an internal direct access file can be associated with a single external file for writing. (See test CE2107E.)

An external file associated with more than one internal file can be deleted for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO. (See test CE2110B.)

Temporary sequential files and temporary direct files are given names. Temporary files given names are not deleted when they are closed. (See tests CE2108A and CE2108C.)

. Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 254 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for five tests were required to successfully demonstrate the test objective. (See section 3.6.)

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	105	1049	1614	17	12	44	2841
Inapplicable	5	2	239	0	6	2	254
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	190	484	540	244	166	98	141	327	131	36	234	3	247	2841
Inapplicable	14	88	134	4	0	0	2	0	6	0	0	0	6	254
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 27 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	C35904B	C37215E	C45332A	CC1311B
E28005C	C35A03E	C37215G	C45614C	BC3105A
C34004A	C35A03R	C37215H	A74106C	AD1A01A
C35502P	C37213H	C38102C	C85018B	CE2401H
A35902C	C37213J	C41402A	C87B04B	CE3208A
C35904A	C37215C			

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, 254 tests were inapplicable for the reasons indicated:

- C35702A uses SHORT_FLOAT which is not supported by this implementation.
- C35702B uses LONG_FLOAT which is not supported by this implementation.

- A39005C and C87B62B use length clauses with STORAGE_SIZE specifications for access types which are not supported by this implementation.

- A39005E and C87B62C use length clauses with SMALL specifications which are not supported by this implementation.

- C35502I..J (2 tests), C35502M..N (2 tests), C35507I..J (2 tests), C35507M..N (2 tests), C35508I..J (2 tests), C35508M..N (2 tests), A39005F, and C55B16A use enumeration representation clauses which are not supported by this compiler.

- The following 13 tests use LONG_INTEGER, which is not supported by this compiler:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45631C	C45632C
B52004D	C55B07A	B55B09C		

- C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.

- C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this compiler.

- CA3004E, EA3004C, and LA3004A use the INLINE pragma for procedures, which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use the INLINE pragma for functions, which is not supported by this compiler.

- AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

- AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

- The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for five Class B tests.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B24009A B49003A B49005A B59001A B59001E

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the ICC Ada, Release 5.0, 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 the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the ICC Ada, Release 5.0, compiler using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a HP 9000 Model 825 host/target operating under HP-UX, Release 2.0.

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 modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded onto a VAX-11/750 and then transferred using FTP over ETHERNET to the host computer. After the test files were loaded to disk, the full set of tests was compiled on the HP 9000 Model 825, and all executable tests were linked and run. Results were transferred using NFS to an Integrated Solutions computer for printing.

The compiler was tested using command scripts provided by Irvine Compiler Corporation and reviewed by the validation team. The compiler was tested using all default option settings except for the following:

Option	Effect
-STACK_CHECK	Enable stack checking.
-PA2F 5	Enable overflow checking.

Tests were compiled, linked, and executed (as appropriate) using a single host/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

Testing was conducted at Irvine Compiler Corporation and was completed on 14 July 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

Irvine Compiler Corporation has submitted the following Declaration of Conformance concerning the ICC Ada, Release 5.0, compiler.

DECLARATION OF CONFORMANCE

Compiler Implementor: Irvine Compiler Corporation
Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name: ICC Ada	Release: 5.0
Host Architecture ISA: HP 9000 Model 825	OS&VER#: HP-UX, Release 2.0
Target Architecture ISA: HP 9000 Model 825	OS&VER#: HP-UX, Release 2.0

Implementor's Declaration

I, the undersigned, representing Irvine Compiler 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 Irvine Compiler 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.

_____ Date: _____
Irvine Compiler Corporation
Dan Eilers, President

Owner's Declaration

I, the undersigned, representing Irvine Compiler 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 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 as measured by ACVC 1.9.

_____ Date: _____
Irvine Compiler Corporation
Dan Eilers, President

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 the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the ICC Ada, Release 5.0, compiler are described in the following sections, taken from Appendix F of the Ada Standard. Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -2147483648 .. 2147483647;
     type SHORT_INTEGER is range -32768 .. 32767;
     type TINY_INTEGER is range -128 .. 127;

     type FLOAT is digits 15
                   range -2.24711641857789E+307 .. 2.24711641857789E+307;

     type DURATION is delta 2.44140625E-04 range -524287.0 .. 524287.0;

     ...

end STANDARD;
```

Appendix F
ICC Ada Version 5.0
Hewlett-Packard, Model 9000/825
HP-UX, Version 2.0

Irvine Compiler Corporation
18021 Sky Park Circle, Suite L
Irvine, CA 92714
(714) 250-1366

July 13, 1988

1 ICC Ada Implementation

The Ada language definition allows certain differences between compilers. This document describes the implementation-dependent characteristics of ICC Ada.

2 Pragmas

The following predefined pragmas are implemented in ICC Ada as described by the Ada Reference Manual:

Elaborate This pragma allows the user to modify the elaboration order of compilation units.

List This pragma enables or disables writing to the output list file.

Pack Packing on arrays is implemented to the bit level. The current implementation packs records to the byte level. Fields of a record must start on byte boundaries.

Page This pragma ejects a new page in the output list file (if enabled).

Priority This pragma sets the priority of a task or main program. The range of the subtype priority is 0..254.

The following predefined pragmas have been extended by ICC:

Interface This pragma is allowed to designate variables in addition to subprograms. It is also allowed to have an optional third parameter which is a string designating the name for the linker to use to reference the variable or subprogram.

Suppress In addition to suppressing the standard checks, ICC also permits suppressing the following:

Exception_info Suppressing **exception_info** improves run-time performance by reducing the amount of information maintained for messages that appear when exceptions are propagated out of the main program or any task.

All_checks Suppressing **all_checks** suppresses all the standard checks as well as **exception_info**.

The following predefined pragmas are currently not implemented by ICC:

Controlled, Inline, Memory_size, Optimize

Shared, Storage_unit, System_name

The following additional pragmas have been defined by ICC:

Compress This pragma reduces the storage required for discrete subtypes in structures (arrays and records). Its single argument is the name of a discrete subtype. It specifies that the subtype should be represented as compactly as possible (regardless of the representation of the subtype's base type) when the subtype is used in a structured type. The storage requirement for variables and parameters is not affected. This pragma must appear prior to any reference to the named subtype.

Export This pragma is a complement to the predefined pragma **interface**. It enables subprograms written in Ada to be called from other languages. It takes 2 or 3 arguments. The first is the language to be called from, the second is the subprogram name, and the third is an optional string designating the actual subprogram name to be used by the linker. This pragma must appear prior to the body of the designated subprogram.

No_zero The single parameter to **no_zero** is the name of a record type. If the named record type has "holes" between fields that are normally initialized with zeroes, this pragma will suppress the clearing of the holes. If the named record type has no "holes" this pragma has no effect. When zeroing is disabled, comparisons (equality and non-equality) of the named type are disallowed. The use of this pragma can significantly reduce initialization time for record objects.

Put, Put_line These pragmas take any number of arguments and write their value to standard output at compile time when encountered by the compiler. The arguments may be expressions of any string, enumeration, or integer type, whose value is known at compile time. Pragma **put_line** adds a carriage return after printing all of its arguments. These pragmas are often useful in conjunction with conditional compilation. They may appear anywhere a pragma is allowed.

3 Preprocessor Directives

ICC Ada incorporates an integrated preprocessor whose directives begin with the keyword **pragma**. They are as follows:

If, Elsif, Else, End These preprocessor directives provide a conditional compilation mechanism. The directives **if** and **elsif** take a boolean static expression as their single argument. If the expression evaluates to **FALSE** then all text up to the next **end, elsif** or **else** directive is ignored. Otherwise, the text is compiled normally. The usage of these directives is identical to that of the similar Ada constructs. These directives may appear anywhere pragmas are allowed and can be nested to any depth.

Include This preprocessor directive provides a compile-time source file inclusion mechanism. It is integrated with the library management system, and the automatic recompilation facilities.

The results of the preprocessor pass, with the preprocessor directives deleted and the appropriate source code included, may be output to a file at compile-time.

4 Attributes

ICC Ada implements all of the predefined attributes, including the Representation Attributes described in section 13.7 of the Ada RM.
Limitations of the predefined attributes are:

Address This attribute cannot currently be used with a statement label or a task entry.

Storage_size Since ICC Ada does not allocate dynamic storage using "collections", this attribute always returns a constant value.

The implementation defined attributes for ICC Ada are:

Version, System, Target These attributes are used by ICC for conditional compilation. The prefix must be a discrete type. The values returned vary depending on the target architecture and operating system.

5 Input/Output Facilities

5.1

The implementation *dependent specifications* from TEXT_IO and DIRECT_IO are:

```
type COUNT is range 0 .. integer'last;
subtype FIELD is INTEGER range 0 .. integer'last;
```

5.2 FORM Parameter

ICC Ada implements the FORM parameter to the procedures OPEN and CREATE in DIRECT_IO, SEQUENTIAL_IO, and TEXT_IO to perform a variety of ancillary functions. The FORM parameter is a string literal containing parameters in the style of named parameter notation. In general the FORM parameter has the following format:

"*field*$_1$ => *value*$_1$ [, *field*$_n$ => *value*$_n$]"

where *field*$_i$ => *value*$_i$ can be

OPTION	=>	NORMAL
OPTION	=>	APPEND
PAGE_MARKERS	=>	TRUE
PAGE_MARKERS	=>	FALSE
READ_INCOMPLETE	=>	TRUE
READ_INCOMPLETE	=>	FALSE
MASK	=>	<9 character protection mask>

Each *field* is separated from its *value* with a "=>" and each *field/value* pair is separated by a comma. Spaces may be added anywhere between tokens and upper-case/lower-case is insignificant. For example:

```
create( f, out_file, "list.data",
    "option => append, PAGE_MARKERS => FALSE, Mask => rwxrwx---");
```

The interpretation of the fields and their values is presented below.

OPTION Files may be OPENed for appendage. This causes data to be appended directly onto the end of an existing file. The default is NORMAL which overwrites existing data. This field applies to OPEN in all three standard I/O packages. It has no effect if applied to procedure CREATE.

PAGE_MARKERS If FALSE then all TEXT_IO routines dealing with page terminators are disabled. They can be called, however they will not do anything. In addition the page terminator character (^L) is allowed to be read with GET and GET_LINE. The default is TRUE which leaves page terminators active. Disabling page terminators is particularly useful when using TEXT_IO with an interactive device.

READ_INCOMPLETE This field applies only to DIRECT_IO and SEQUENTIAL_IO and dictates what will be done with reads of incomplete records. Normally, if a READ is attempted and there is not enough data in the file for a complete record, then END_ERROR or DATA_ERROR will be raised. By setting READ_INCOMPLETE to TRUE, an incomplete record will be read successfully and the remaining bytes in the record will be zeroed. Attempting a read after the last incomplete record will raise END_ERROR. The SIZE function will reflect the fact that there is one more record when the last record is incomplete and READ_INCOMPLETE is TRUE.

MASK Set a protection mask to control access to a file. The mask is a standard nine character string notation used by Unix. The letters cannot be rearranged or deleted so that the string is always exactly nine characters long. This applies to CREATE in all three standard I/O packages. The default is determined at runtime by the user's environment settings.

If a syntax error is encountered within the FORM parameter then the exception USE_ERROR is raised at the OPEN or CREATE call. Also, the standard function TEXT_IO.FORM returns the current setting of the form fields, including default values, as a single string.

6 Package System

Package SYSTEM is:

```
Package SYSTEM is
   type name is (hp);

   -- Language Defined Constants

   system_name : constant name := hp;
   storage_unit: constant := 8;
   memory_size : constant := 24 * 1024 * 1024;  -- 24 Mb
   min_int     : constant := -2**31;
   max_int     : constant := 2**31-1;
   max_digits  : constant := 15;
   max_mantissa: constant := 31;
   fine_delta  : constant := 2.0**(-31);
   tick        : constant := 1.0/60.0;

   type address is range min_int..max_int;
   subtype priority is integer range 0..254;

   -- Constants for the HEAPS package

   bits_per_bmu : constant := 8;
   max_alignment: constant := 8;
   min_mem_block: constant := 1024;

   -- Constants for the HOST package

   host_clock_resolution: constant := 0;
   base_date_correction : constant := 25_202;
end SYSTEM;
```

7 Limits

Most data structures held within the ICC Ada compiler are dynamically allocated, and hence have no inherent limit (other than available memory). Some limitations are:

The maximum input line length is 254 characters.

The maximum number of tasks abortable by a single abort statement is 64.

Include files can be nested to a depth of 3.

The number of packages, subprograms, tasks, variables, aggregates, types or labels which can appear in a compilation unit is unlimited.

The number of compilation units which can appear in one file is unlimited.

The number of statements per subprogram or block is unlimited.

Packages, tasks, subprograms and blocks can be nested to any depth.

There is no maximum number of compilation units per library, nor any maximum number of libraries per library system.

8 Numeric Types

ICC Ada supports three predefined integer types:

TINY_INTEGER	-128..127	8 bits
SHORT_INTEGER	-32768..32767	16 bits
INTEGER	-2147483648..2147483647	32 bits

In addition, unsigned TINY and SHORT integer types can be defined by the user via the SIZE representation clause. Storage requirements for types can be reduced by using pragma pack and for subtypes by using the ICC pragma compress.

Type float is available.

Attribute	FLOAT value
size	64 bits
digits	15
first	$-2.24711641857789E+307$
last	$+2.24711641857789E+307$

9 Tasks

The type DURATION is defined with the following characteristics:

Attribute	DURATION value
delta	$2.44140625E-04$ sec
small	$2.44140625E-04$ sec
first	-524287.0 sec
last	524287.0 sec

The subtype SYSTEM.PRIORITY as defined provides the following range:

Attribute	PRIORITY value
first	0
last	254

Higher numbers correspond to higher priorities. If no priority is specified for a task, PRIORITY'FIRST is assigned during task creation.

10 Representation Clauses

Address clauses are implemented, but only for variables. Address clauses are currently not implemented for procedures, packages, tasks or statement labels.

The 'SIZE length clause is implemented. When applied to integer range types this representation clause can be used to define unsigned types.

The 'STORAGE_SIZE length clause for task types is implemented.

The 'STORAGE_SIZE length clause for access types is not implemented.

The 'SMALL length clause for fixed point types is not implemented.

Enumeration representation clauses are not implemented.

Record representation clauses are implemented. There are no implementation-generated names that can be used in record representation clauses.

Interfacing to Assembly, C, and Ada is supported via pragma **interface** and pragma **export**.

Unchecked_conversion is implemented without restriction.

Unchecked_deallocation is currently not implemented.

Machine code insertion is implemented. The package MACHINE_CODE is currently not provided.

11 Main Programs

Main programs may be procedures or functions and may have any number of parameters. Parameter and function return types can be either discrete types (including enumerations) or unconstrained arrays. Parameters may also include default values. If the main program is a function, then upon exit the returned value will be printed on the user's screen. If the program is invoked with the wrong number of parameters a usage error message is printed and execution is aborted. If an illegal value is passed to a parameter then CONSTRAINT_ERROR is raised.

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..253 => 'A', 254 => '1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..253 => 'A', 254 => '2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..126 => 'A', 127 => '3', 128..254 => 'A')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..126 => 'A', 127 => '4', 128..254 => '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..251 => '0', 252..254 => "298")

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..248 => '0', 249..254 => "69.0")
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1 => '"', 2..121 => 'A', 122 => '"')
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1 => '"', 2..134 => 'A', 135..136 => "1"")
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..234 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	2147483647
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	/xxx/xxx/xxx
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	/xxx/xxx/xxx*
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	524_287.5

Name and Meaning	Value
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	10_000_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	/xxx/xxx/xxx
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	/xxx/BAD-FILE-NAME
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2147483648
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-524_287.5
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-10_000_000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	254
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648

TEST PARAMETERS

Name and Meaning	Value
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2 => "2:", 3..251 => '0', 252..254 => "11:")
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..3 => "16:", 4..250 => '0', 251..254 => "F.E:")
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1 => '"', 2..127 => 'A', 128 => '3', 129..253 => 'A', 254 => '"')
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	TINY_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.	8#37777777776#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 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.

- B28003A: A basic declaration (line 36) incorrectly follows a later declaration.

- E28005C: This test requires that "PRAGMA LIST (ON);" not appear in a listing that has been suspended by a previous "PRAGMA LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the AJPO.

- C34004A: The expression in line 168 yields a value outside the range of the target type T, but there is no handler for CONSTRAINT_ERROR.

- C35502P: The equality operators in lines 62 and 69 should be inequality operators.

- A35902C: The assignment in line 17 of the nominal upper bound of a fixed-point type to an object raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

- C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

- C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may, in fact, raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

WITHDRAWN TESTS

- C35A03E and C35A03R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard does not support this assumption.

- C37213H: The subtype declaration of SCONS in line 100 is incorrectly expected to raise an exception when elaborated.

- C37213J: The aggregate in line 451 incorrectly raises CONSTRAINT_ERROR.

- C37215C, C37215E, C37215G, and C37215H: Various discriminant constraints are incorrectly expected to be incompatible with type CONS.

- C38102C: The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.

- C41402A: The attribute 'STORAGE_SIZE is incorrectly applied to an object of an access type.

- C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.

- C45614C: The function call of IDENT_INT in line 15 uses an argument of the wrong type.

- A74106C, C85018B, C87B04B, and CC1311B: A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur at lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively.

- BC3105A: Lines 159 through 168 expect error messages, but these lines are correct Ada.

- AD1A01A: The declaration of subtype SINT3 raises CONSTRAINT_ERROR for implementations which select INT'SIZE to be 16 or greater.

- CE2401H: The record aggregates in lines 105 and 117 contain the wrong values.

- CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.

DTIC

4-89

END

DATE

FILMED