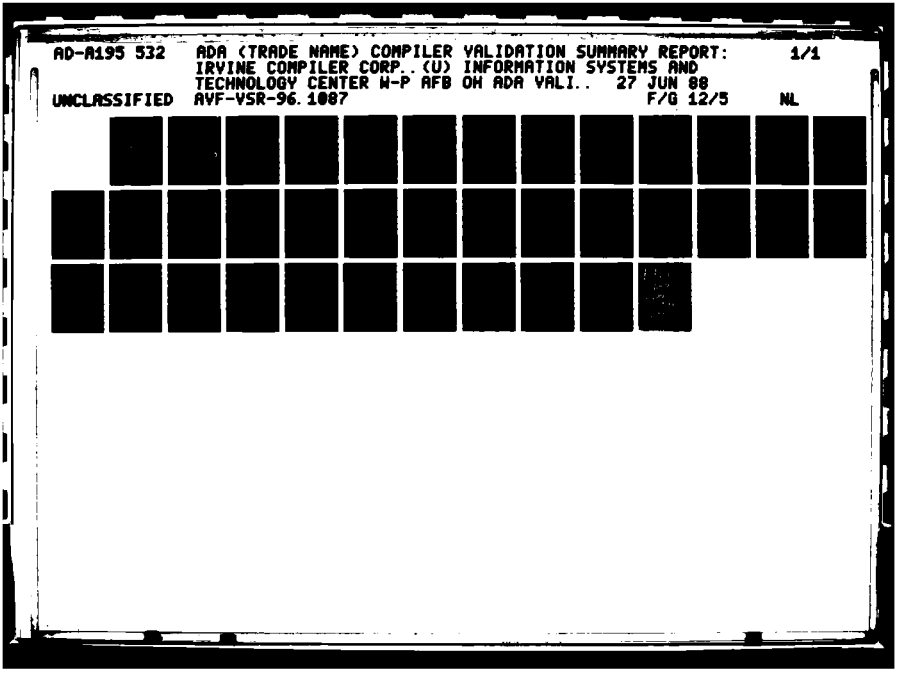
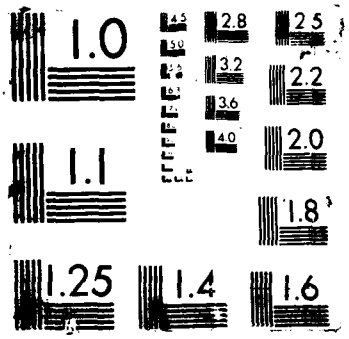

AD-A195 532
ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT:
IRVINE COMPILER CORP.. (U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI..
27 JUN 88
1/1
UNCLASSIFIED
AVF-VSR-96.1087
F/G 12/5
NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

(When Data Entered)

AD-A195 532

ENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

12. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: Irvine Compiler Corporation, ICC Ada Compiler, Release 4.0, VAX-11/750	5. TYPE OF REPORT & PERIOD COVERED 27 June 1987 to 27 June 1988 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 20301-3081	12. REPORT DATE 27 June 1987 13. NUMBER OF PAGES 36 p.
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)

ICC Ada Compiler, Release 4.0. Irvine Compiler Corporation, Wright-Patterson AFB, VAX-11/750 uncer VMS, Version 4.5 (host and target). ACVC 1.8.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE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

AVF Control Number: AVF-VSR-96.1087
87-04-07-ICC

Ada® COMPILER
VALIDATION SUMMARY REPORT:
Irvine Compiler Corporation
ICC Ada Compiler, Release 4.0
VAX-11/750

Completion of On-Site Testing:
27 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, DC

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

Ada® Compiler Validation Summary Report:

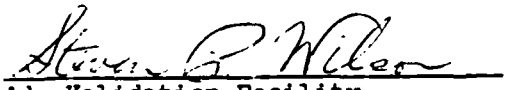
Compiler Name: ICC Ada Compiler, Release 4.0

Host: VAX-11/750 under VMS, Version 4.5

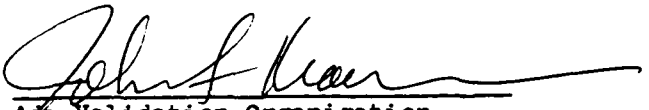
Target: VAX-11/750 under VMS, Version 4.5

Testing Completed 27 June 1987 Using ACVC 1.8

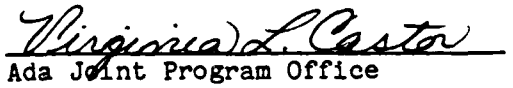
This report has been reviewed and is approved.



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

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

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the ICC Ada Compiler, Release 4.0, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The ICC Ada Compiler is hosted on a VAX-11/750 operating under VMS, Version 4.5. Programs processed by this compiler may be executed on a VAX-11/750 operating under VMS, Version 4.5.

On-site testing was performed 21 June 1987 through 27 June 1987 at Irvine Compiler Corporation in Irvine CA, 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 34 of the processed tests determined to be inapplicable. The remaining 2104 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	96	220	298	244	161	97	137	261	124	32	218	216	2104
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	20	105	122	3	0	0	2	1	6	0	0	17	276
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.

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 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 21 June 1987 through 27 June 1987 at Irvine Compiler Corporation in Irvine CA.

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 follows 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: ICC Ada Compiler, Release 4.0

ACVC Version: 1.8

Certificate Number: 870622W1.08101

Host Computer:

Machine:	VAX-11/750
Operating System:	VMS, Version 4.5
Memory Size:	4 megabytes

Target Computer:

Machine:	VAX-11/750
Operating System:	VMS, Version 4.5
Memory Size:	4 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. 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 does not reject such calculations and processes them correctly. (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.)

- 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 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 does not raise an 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, all choices are not evaluated before being checked for identical bounds. (See test E43212B.)

  All choices are not 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.)

CONFIGURATION INFORMATION

. 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 'SIZE. It rejects 'STORAGE_SIZE for collections and 'SMALL clauses. Enumeration representation clauses, including those that specify noncontiguous values, appear to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is not supported for procedures or functions. (See tests CA3004E and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants. The package DIRECT IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

An existing text file can be opened in OUT_FILE mode and can be created in both OUT_FILE and IN_FILE modes. (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).)

More than one internal file can be associated with each external file for sequential I/O for both reading and writing. (See tests CE2107A..F (6 tests).)

More than one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107A..F (6 tests).)

An external file associated with more than one internal file can be deleted. (See test CE2110B.)

Temporary sequential files are given a name. Temporary direct files are given a name. Temporary files given names are deleted when they are closed. (See tests CE2108A and CE2108C.)

. Generics.

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

Generic package declarations and bodies can 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 ICC Ada Compiler was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 276 tests were inapplicable to this implementation, and that the 2104 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	67	865	1100	17	11	44	2104
Failed	0	0	0	0	0	0	0
Inapplicable	2	2	268	0	2	2	276
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												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	96	220	298	244	161	97	137	261	124	32	218	216	2104
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	20	105	122	3	0	0	2	1	6	0	0	17	276
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	BC3204C
B33203C	B45116A	C87B50A	
C34018A	C48008A	C92005A	
C35904A	B49006A	C940ACA	
B37401A	B4A010C	CA3005A..D (4 tests)	

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, 276 tests were inapplicable for the reasons indicated:

- C34001E, B52004D, B55B09C, and C55B07A use LONG_INTEGER which is not supported by this compiler.
- C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.

- C34001G and C35702B use LONG_FLOAT which is not supported by this compiler.

- C87B62B..C (2 tests) use the length clauses 'STORAGE_SIZE for access types and 'SMALL 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.

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

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

- AE2101C, CE2201D, and CE2201E use an instantiation of package SEQUENTIAL_IO with unconstrained array types which is not supported by this compiler.

- AE2101H and CE2401D use an instantiation of package DIRECT_IO with unconstrained array types which is not supported by this compiler.

- CE2107B..E (4 tests), CE2110B, CE2111D, CE2111H, CE3111B..E (4 tests), and CE3114B are inapplicable because multiple internal files cannot be associated with the same external file when one of the internal files is open for writing. The proper exception is raised when multiple access is attempted.

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

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

TEST INFORMATION

Splits were required for three Class B tests:

B59001A B59001E B85013C

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 ICC Ada 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 ICC Ada Compiler using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX-11/750 operating under VMS, Version 4.5.

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 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 and linked on the VAX-11/750, and all executable tests were run. Results were transferred via RS232 to a Gould 6050 and printed.

The compiler was tested using command scripts provided by Irvine Compiler Corporation and reviewed by the validation team. The following options were in effect for testing:

Option	Effect
-LIST	produces a list file
-QUIET	turns off messages to console
-STACK_CHECK	adds code to check stack size for tasks

Tests were compiled, linked, and executed (as appropriate) using one computer. Test output and compilation listings 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 Irvine Compiler Corporation in Irvine CA on 21 June 1987, and departed after testing was completed on 27 June 1987.

APPENDIX A

DECLARATION OF CONFORMANCE

Irvine Compiler Corporation has submitted the following Declaration of Conformance concerning the ICC Ada Compiler.

DECLARATION OF CONFORMANCE

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

Base Configuration

Base Compiler Name: ICC Ada Compiler Version: Release 4.0
Host Architecture ISA: VAX-11/750 OS&VER #: VMS, Version 4.5
Target Architecture ISA: VAX-11/750 OS&VER #: VMS, Version 4.5

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.

Dan Eilers Date: June 25, 1987
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 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.

Dan Eilers Date: June 25, 1987
Irvine Compiler Corporation
Dan Eilers, President

®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 ICC Ada Compiler, Release 4.0, 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 -2147483648 .. 2147483647;
     type SHORT_INTEGER is range -32768 .. 32767;
     type TINY_INTEGER is range -128 .. 127;

     type FLOAT is digits 9 range
          -4.25352957066610E+37 .. 4.25352957066610E+37;

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

     ...

end STANDARD;
```

Appendix F
ICC Ada Implementation

DEC VAX-11/750 / VMS, Version 4.5

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

May 28, 1987

1 ICC Ada Implementation

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

2 Pragmas

The following pragmas are added by ICC:

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.

Compress This pragma is a complement to the representation clause size. It takes the name of a discrete subtype as the single argument, and specifies that the subtype should be represented as compactly as possible, regardless of the representation of the subtype's base type. This pragma must appear prior to any reference to the named subtype.

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.

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**.

3 Preprocessor Directives

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

If, Elsif, Else, End If These preprocessor directives provide a conditional compilation mechanism.

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

4 Input/Output Facilities

4.1

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;
```

4.2 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 TRUE then all TEXT_IO routines dealing with page terminators are disabled. They can be called, they simply do 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.

READ_INCOMPLETE This field applies only to DIRECT_IO and SEQUENTIAL_IO and informs the package about what should 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.

5 Line Length

The maximum line length is 254 characters.

6 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

Unsigned tiny and short integer types are available via the SIZE representation clause.

Type float is available.

Attribute	**FLOAT** value
size	64 bits
digits	9
first	$-4.25352957066610E+37$
last	$+4.25352957066610E+37$

7 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.

8 Representation Clauses

Address clauses are implemented, but only for objects.

The 'SIZE length clause is implemented.

Enumeration representation clauses are implemented.

Record representation clauses are implemented.

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

Unchecked_conversion is implemented without restriction.

9 Main Programs

Main programs may have parameters and return values of discrete types or unconstrained strings. Default values are also permitted.

```
------------------------------------------------------------------
--       The following software is the sole property of         --
--                 Irvine Compiler Corporation                  --
--     containing its proprietary, confidential information.    --
--                                                              --
--      Copyright (C) 1982-1987 Irvine Compiler Corporation     --
------------------------------------------------------------------
package system is

   type name is (vax);

   -- Language Defined Constants

   system_name : constant name := vax;
   storage_unit: constant := 8;                    -- Storage unit size in bits.
   memory_size : constant := 4096 * 1024;          -- Bytes.
   min_int     : constant := -2**31;
   max_int     : constant := 2**31-1;
   max_digits  : constant := 9;
   max_mantissa: constant := 31;
   fine_delta  : constant := 2.0**(-30);
   tick        : constant := 1.0/100.0;

   type address is range min_int..max_int;         -- Signed 32 bit range.
   subtype priority is integer range 0..254;       -- 0 is default priority.

   -- Constants for the HEAPS package

   bits_per_bmu : constant := 8;                   -- Bits per basic machine unit.
   max_alignment: constant := 4;                   -- Maximum alignment required.
   min_mem_block: constant := 1024;                -- Minimum chunk request size.

   -- Constants for the HOST package

   host_clock_resolution: constant := 1;           -- 1 microsecond.
   base_date_correction : constant := 25_202;      -- Unix base date is 1/1/1970.

   -- Constants for VMS_IO

   text_size: constant := 392;                     -- Size of file descriptors for
   file_size: constant := 392;                     -- I/O packages (in BMUs).

   pragma put_line("Target: ", system_name);
end system;
```

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 | 128..254 => 'A', 127 => '3')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..126 | 128..254 => 'A', 127 => '4')
$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 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..248 => '0', 249..254 => "69.0E1")
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1..234 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	2147483647
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters, or is too long if no invalid characters exist.	xxx&
$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.	xxx*
$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.	524_287.5
$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.	[xxx.xxx]xxx
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	[xxx.xxx]xx2
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2147483647
$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.	-524_287.5
$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.	14
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	254
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2147483647

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.	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#
$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 if 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 end case; is missing from line 42.
- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

WITHDRAWN TESTS

- 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

9-88

DTIC