AD-A211 640

THE FILE COPY (2)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) **Ada Compiler Validation Summary Report:** York Software Engineering Limited, York Ada Compiler Environment (ACE) Release 4, Intergraph Inter Pro 340 (Host & Target) 890531N1.10097	5. TYPE OF REPORT & PERIOD COVERED 31 May 1989 to 31 May 1990 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) National Computing Centre Limited, Manchester, United Kingdom.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Computing Centre Limited, Manchester, United Kingdom.	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 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) National Computing Centre Limited, Manchester, United Kingdom.	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

DTIC ELECTE S AUG 22 1989 D B

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)

York Software Engineering Limited, York Ada Compiler Environment (ACE) Release 4, Intergraph Inter Pro 340 under UNIX System V.3 (Host & Target), ACVC 1.10.

89 [illegible] 9

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE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

UNCLASSIFIED

AVF Control Number: AVF-VSR-90502/51

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890531N1.10097
York Software Engineering Limited
York Ada Compiler Environment (ACE) Release 4
Intergraph Inter Pro 340 Host

Completion of On-Site Testing:
31 May 1989

Prepared By:
Testing Services
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

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

Ada Compiler Validation Summary Report:

Compiler Name: **York Ada Compiler Environment (ACE) Release 4**

Certificate Number: **890531N1.10097**

Host: **Intergraph Inter Pro 340 under UNIX System V.3**

Target: **Intergraph Inter Pro 340 under UNIX System V.3**

Testing Completed 31 May 1989 Using ACVC 1.10

This report has been reviewed and is approved.

J. Pink

Jane Pink
Testing Services Manager
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

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

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

for Edward W. Liddraddt
Deputy Director

Ada Joint Program Office
Dr. John Solomond
Director AJPO
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 is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

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

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any 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 The National Computer Centre Limited according to procedures established by the Ada Joint Program Office and administered by the Ada Validation

Organization (AVO). On-site testing was completed **31 May 1989** at **UNIVERSITY OF YORK, YORK, Y01 5DD, ENGLAND.**

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 Stre
Washington DC 20301-3081

or from:

Testing Services
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
England

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 which executes the code generated by the compiler.
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 errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. 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 the run time system to ensure 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.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. 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. In some cases, an implementation may legitimately detect errors during compilation of the test.

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 each test 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 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: **York Ada Compiler Environment (ACE) Release 4**

ACVC Version: **1.10**

Certificate Number: **890531N1.10097**

Host Computer:

Machine: **Intergraph Inter Pro 340**

Operating System: **UNIX System V.3**

Memory Size: **16 Mbytes**

Target Computer:

Machine: **Intergraph Inter Pro 340**

Operating System: **UNIX System V.3**

Memory Size: **16 Mbytes**

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:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined types SHORT_INTEGER, BYTE_INTEGER, and LONG_FLOAT, in the package STANDARD. (See tests B86001T..Z (7 tests).)

c. Based literals.

(1) An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR when a value exceeds SYSTEM.MAX_INT. This implementation raises CONSTRAINT_ERROR during execution. (See test E24201A.)

d. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

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

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

(4) CONSTRAINT_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.)

(5) CONSTRAINT_ERROR 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.)

(6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

c. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round to even. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round to even. (See tests C46012A..Z (26 tests).)

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

f. 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:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

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

(3) NUMERIC_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

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

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

(4) CONSTRAINT_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.)

(5) CONSTRAINT_ERROR 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.)

(6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

e. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round to even. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round to even. (See tests C46012A..Z (26 tests).)

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

f. 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:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

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

(3) NUMERIC_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

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

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array objects are declared. (See test C52104Y.)

(6) In assigning one-dimensional array types, the expression is 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.)

(7) In assigning two-dimensional array types, the expression is not 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.)

g. 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 NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

h. Discriminated types.

(1) In assigning record types with discriminants, the expression is 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.)

i. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that index subtype checks are made as choices are evaluated. (See tests C43207A and C43207B.)

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

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

j. Pragmas.

(1) The pragma INLINE is not supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

k. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations provided that generic bodies are compiled before instantiations are made. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic subprogram declarations and bodies can be compiled in separate compilations provided that generic bodies are compiled before instantiations are made. (See tests CA1012A and CA2009F.)

(3) Generic library subprogram specifications and bodies can be compiled in separate compilations provided that generic bodies are compiled before instantiations are made. (See test CA1012A.)

(4) Generic non-library package bodies as subunits can be compiled in separate compilations provided that generic bodies (including all subunits) are compiled before instantiations are made. (See test CA2009C).

(5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs provided that generic bodies (including all subunits) are compiled before instantiations are made. (See test CA2009F.)

(6) Generic unit bodies and their subunits can be compiled in separate compilations from their stubs provided that generic bodies (including all subunits) are compiled before instantiations are made. (See test CA3011A.)

(7) Generic package declarations and bodies can be compiled in separate compilations provided that generic bodies are compiled before instantiations are made. (See tests CA2009C, BC3204C, and BC3205D.)

(8) Generic library package specifications and bodies can be compiled in separate compilation provided that generic bodies are compiled before instantiations are made. (See tests BC3204C and BC3205D.)

(9) Generic unit bodies and their subunits can be compiled in separate compilations provided that generic bodies (including all subunits) are compiled before instantiations are made. (See test CA3011A.)

l. Input and output.

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

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

(4) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)

(5) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)

(6) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)

(7) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(8) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(9) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

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

(11) Temporary sequential files are not given names. (See test CE2108A.)

(12) Temporary direct files are not given names. (See test CE2108C.)

(13) Temporary text files are not given names. (See test CE3112A.)

(14) Only one internal file can be associated with each external file for sequential files when writing or reading. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(15) Only one internal file can be associated with each external file for direct files when writing or reading. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(16) Only one internal file can be associated with each external file for text files when writing or reading. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 591 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 56 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	122	1128	1753	17	18	44	3082
Inapplicable	7	10	562	0	10	2	591
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

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	192	572	536	245	172	99	157	330	129	36	250	96	268	3082
Inapp	20	77	144	3	0	0	9	2	8	0	2	273	53	591
Withdrawn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G	B97102E	C97116A
BC3009B	CD2A62D	CD3A63A..D (4 Tests)	CD2A66A..D (4 tests)
CD2A73A..D (4 tests)	CD2A76A..D (4 tests)	CD2A81G	CD2A83G
CD2A84M..N (2 tests)	CD50110	CD2B15C	CD7205C
CD2D11B	CD5007B	ED7004B	ED7005C..D (2 tests)
ED7006C..D (2 tests)	CD7105A	CD7203B	CD7204B
CD7205D	CE2107I	CE3111C	CE3301A
CD3411B			

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, 591 tests were inapplicable for the reasons indicated:

a B23003D and B23003E are not applicable because no restrictions are made on the maximum line length

b C24113I, C24113J and C24113K are not applicable because numeric literals exceed the maximum token line length of 128

c The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

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)

d The following 204 tests are not applicable because 'SIZE, 'STORAGE_SIZE and 'SMALL representation clauses are not supported.

A39005B..E (4 tests)	C87B62A..D (4 tests)
CD1009A..L (12 tests)	CD10090..U (7 tests)
CD1C03A	CD1C03C
CD1C03E..F (2 tests)	CD1C04A..C (3 tests)
CD1C06A	CD2A21A..E (5 tests)
CD2A22A..J (10 tests)	CD2A23A..E (5 tests)
CD2A24A..J (10 tests)	CD2A31A..D (4 tests)
CD2A32A..J (10 tests)	CD2A41A..E (5 tests)
CD2A42A..J (10 tests)	CD2A51A..E (5 tests)
CD2A52A..D (4 tests)	CD2A52G..J (4 tests)
CD2A53A..E (5 tests)	CD2A54A..D (4 tests)
CD2A54G..J (4 tests)	CD2A61A..L (12 tests)
CD2A62A..C (3 tests)	CD2A65A..D (4 tests)
CD2A71A..D (4 tests)	CD2A72A..D (4 tests)
CD2A74A..D (4 tests)	CD2A75A..D (4 tests)
CD2A64A..D (4 tests)	CD2A81A..F (6 tests)
CD2A83A..C (3 tests)	CD2A83E..F (2 tests)
CD2A84B..I (8 tests)	CD2A84K..L (2 tests)
CD2A87A	CD2A91A..E (5 tests)
CD2B11B..G (6 tests)	CD2B15B
CD2B16A	CD2C11A..E (5 tests)
CD2D11A	CD2D13A
ED2A26A	ED2A56A
ED2A86A	

e E28005D is not applicable because the Pragmas LIST and PAGE are not supported by this implementation.

f C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

g C35A06N contains a fixed-print type declaration (line 17) for which this implementation chooses the base type to be the same as the type declared. As a consequence, a subsequent fixed-point type declaration (lines 30 & 32) that uses the earlier type's SAFE_LARGE value (+/-) as the bounds of its range, results in a type with 0 as its sole model number. This, in turn, causes CONSTRAINT_ERROR to be raised when the SAFE_SMALL and SAFE_LARGE values of the first type are passed as parameters of the later type. This test has been ruled not applicable due to this unexpected result.

h The following 16 tests are not applicable because this implementation does not support a predefined type LONG_INTEGER:

C45231C C45304C C45502C C45503C C45504C

C45504F C45611C C45613C C45614C C45631C
C45632C B52004D C55B07A B55B09C B86001W
CD7101F

i C45531I..P (8 tests) and C45532I..P (8 tests) are not applicable because the value of SYSTEM.MAX_MANTISSA is less than 31.

j C4A013B is not applicable because the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type would raise an exception; since the expression must be static, it is rejected at compile time.

k B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

l B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

m C87B26B and A95074D are not applicable because, for this implementation, type SYSTEM.ADDRESS is a limited private type.

n C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

o CA2009C, CA2009F, BC3204C and BC3205D compile generic specifications and bodies or specifications and bodies of subunits of generic units in separate files. This compiler requires that no instantiations of the corresponding generics occur prior to the compilations of the generic body.

p LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F are not applicable because this implementation does not support pragma INLINE.

q The following 76 tests are not applicable because, for this implementation, type SYSTEM.ADDRESS is a limited private type:

CD5003B..I (8 tests) CD5011A..I (9 tests)
CD5011K..N (4 tests) CD5011Q..S (3 tests)
CD5012A..J (10 tests) CD5012L..M (2 tests)
CD5013A..I (9 tests) CD5013K..O (5 tests)
CD5013R..S (2 tests) CD5014A..O (15 tests)
CD5014R..Z (9 tests)

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

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

t CE2102C, CE2102H, CE3102C, and CE3107A are not applicable because there are no strings that are illegal as file names.

u CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.

v CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

w CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

x CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.

y CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

z CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

aa CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

ab CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

ac CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

ad CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

ae CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

af CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

ag CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

ah CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.

ai CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

aj CE2107A..E (5 tests), CE2107L, CE2110B and CE2111D are not applicable because multiple internal files cannot be associated with the same external file for sequential files. The proper exception is raised when multiple access is attempted.

ak CE2107F..H (3 tests), CE2110D, and CE2111H are not applicable because multiple internal files cannot be associated with the same external file for direct files. The proper exception is raised when multiple access is attempted.

al CE2108B, CE2108D and CE3112B are not applicable because temporary files have no name in this implementation.

am CE3102E is inapplicable because text file CREATE with IN_FILE mode is supported by this implementation.

an CE3102F is inapplicable because text file RESET is supported by this implementation.

ao CE3102G is inapplicable because text file deletion of an external file is supported by this implementation.

ap CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

aq CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.

ar CE3102K is inapplicable because text file OPEN with OUT_FILE mode is not supported by this implementation.

as CE3111A..B (2 tests), CE3111D..E (2 tests), CE3114B and CE3115A are not applicable because multiple internal files cannot be associated with the same external file for text files. The proper exception is raised when multiple access is attempted.

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 was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 56 tests.

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

B22003A	B23004A	B28003A	B28003C	B29001A
B2A003A	B2A007A	B2A010A	B32103A	B33102A
B35103B	B37301B	B38001C	B43201A	B44001A
B44004A	B49007A	B51001A	B51003A	B54A01C
B54A20A	B55A01A	B66001C	B71001C	B91001H
B91002F	B95001A	B95004B	B95061G	B95077A
B97101A	B97102H	BA1101B	BA1101C	BA3006A
BA3006B	BA3007B	BA3008A	BA3013A	BC1001A
BC1016A	BC1202E	BC2001D	BC2001E	BC3204B

The following tests were passed as a result of modification of code as follows:

C45651A — In testing the behaviour of the function ABS for fixed point types, there are some statements that use problematic ranges. The value 1024.0 has been changed to 960.0 on line 256 to overcome this problem.

A62006D
CC1221B
CC1222A
CC1223A
CC1224A — Assignments are made to variables of type SYSTEM_ADDRESS which the implementation rejects because SYSTEM_ADDRESS is a limited private type. In none of these tests is the assignment essential for the test objectivc so the tests are modified by commenting out the assignments. The particular lines for commenting out are

A62006D - line 53 CC1221B - line 44 CC1222A - line 226
CC1223A - line 227 CC1224A - lines 91..97

The following tests were passed as a result of modified evaluation criteria as follows:

B23003F — This implementation does not have a maximum line length, but has a maximum token length of 128. This test checks the maximum line length by using an identifier which exceeds the line length by 1 and therefore serves to check that the token line length limit is enforced.

E28002A/B/D — Pragmas LIST and PAGE are not supported by this implementation. The test requirements for pragmas PAGE and LIST are not essential to the tests' objectives and are thus ignored.

BC1226A Extra error messages are given because SYSTEM_ADDRESS is limited private. These error messages appear on lines 85-88 and should be ignored.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the York Ada compiler environment (ACE) Release 4 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 York Ada Compiler Environment (ACE) Release 4 using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer	: Intergraph Inter Pro 340
Host operating system	: UNIX System V.3
Target computer	: Intergraph Inter Pro 340
Target operating system	: UNIX System V.3
Compiler	: York Ada Compiler Environment (ACE) Release 4
Assembler	: UNIX System V.3 as
Loader/Downloader	: UNIX System V.3 ld
Runtime System	: ACE Runtime System

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.

The contents of the Magnetic tape were not loaded directly onto the host computer.

The contents of the Magntic tape were loaded onto a High Level Hardware Orion 1/05 and then transferred via ethernet to the host computer using the UNIX utility 'rcp'."

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the Intergraph Inter Pro 340. Results were transferred via ethernet to a SUN 3/50 Workstation where they were printed.

The compiler was tested using command scripts provided by York Software Engineering Limited and reviewed by the validation team. The compiler was tested using the following options:

OPTION EFFECT

-M 'identifier'	Make the compilation unit 'identifier' the main subprogram

Tests were compiled, linked, and executed (as appropriate) using a single 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 University of York, York, YO1 5DD, England and was completed on 31 May 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

York Software Engineering Limited has submitted the following Declaration of Conformance concerning the York Ada Compiler Environment (ACE) Release 4.

DECLARATION OF CONFORMANCE

Compiler Implementor: **York Software Engineering Limited**

Ada Validation Facility: **The National Computing Centre Limited**
Oxford Road
Manchester
M1 7ED
United Kingdom

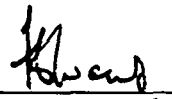
Ada Compiler Validation Capability (ACVC) Version: **1.10**

Base Configuration

Base Compiler Name:	**York Ada Compiler Environment (ACE) Release 4**
Host Architecture:	**Intergraph Inter Pro 340**
Host OS and Version:	**UNIX System V.3**
Target Architecture:	**Same as host**
Target OS and Version:	**Same as host**

Implementor's Declaration

I, the undersigned, representing **York Software Engineering Limited**, 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 **York Software Engineering Limited** 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 : 2 - 2 1989

York Software Engineering Limited
Professor I C Wand
Director

Owner's Declaration

I, the undersigned, representing **York Software Engineering Limited** 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.

Date : 31 May 1989

York Software Engineering Limited
Professor I C Wand
Director

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 **York Ada Compiler Environment (ACE) Release 4** as described in this Appendix, are provided by **York Software Engineering Limited.** Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

type INTEGER is range -(2**31) .. (2**31)-1;
type SHORT_INTEGER is range -(2**15) .. (2**15)-1;
type BYTE_INTEGER is range -128..127;
type FLOAT is digits 6 range -1.70141E+38 .. 1.70141E+38;
type LONG_FLOAT is digits 15 range -1.79769E+308 .. 1.79769E+308;
type DURATION is delta 0.0167 range -16777215.0 .. 16777215.0;

...

end STANDARD;
```

Appendix F

1. Implementation-dependent pragma LIBNAME

This pragma is used to associate a subprogram body written in a language other than Ada with a corresponding Ada subprogram specification in a package specification. The other-language body is supplied in lieu of an Ada body. The pragma must be placed immediately before the specification of the subprogram for which the foreign body is to be supplied. The single argument to pragma LIBNAME is the link-symbol of the foreign body. For C this is the routine name(case is significant). This pragma may also apply to objects.

e.g.

```
-- Ada
package TEST is
  pragma LIBNAME("iadd");
  procedure IADD(I,J: in out INTEGER);
end TEST;

-- C body for IADD
void iadd(a,b)
int *a,*b;
{
  *a = *a + *b;
}
```

2. Implementation-dependent attributes

None.

3. Package SYSTEM

```
package SYSTEM is
  type ADDRESS is limited private;
  type NAME is (UNIX);
  SYSTEM_NAME : constant NAME := UNIX;
  STORAGE_UNIT : constant := 8;
  MEMORY_SIZE : constant := 2147483647;
  MIN_INT : constant := -2147483648;
  MAX_INT : constant := 2147483647;
  MAX_DIGITS : constant := 15;
  MAX_MANTISSA : constant := 30;
  FINE_DELTA : constant := 0.00000000093132257461547851562 5;
  TICK : constant := 0.01;
  subtype PRIORITY is INTEGER range 0..4;
end SYSTEM;
```

4. Representation clauses

Length clauses and interrupts are not implemented. There are miscellaneous restrictions on record representation clauses as follows :-

(1) The static simple expression in any alignment clause must be 1, 2 or 4.

(2) Component clauses for fields whose types are non-static or whose types depend upon discriminants are not allowed.

(3) Fields with a type which is not an array or record type must have a bit length which is less than or equal to 32.

(4) Fields with a type which is an array or record type must begin on a storage unit boundary(1 byte or 8 bits) and must have a field length which is a multiple of a storage unit.

(5) Fields with a discrete type may be packed to a field length which is not a multiple of a storage unit (as long as this is feasible for the range of values for the type) but the compiler will refuse to do this for array and record types.

5. Implementation-generated names

Not implemented.

6. Unchecked conversions

No restrictions.

7. Input-output packages

SEQUENTIAL_IO and DIRECT_IO may not be instantiated with unconstrained types.

TEXT_IO.ENUMERATION_IO will raise USE_ERROR if instantiated with numeric types.

Temporary files do not have names in this implementation.

Actual parameter to "FORM" must be the empty string ("").

8. Main program

The main program should be a parameterless procedure otherwise the effect is undefined.

9. Generic bodies

Generic bodies must appear in the same file as their specifications or be already compiled when an instantiation is made.

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
$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
$BIG_ID1 An identifier the size of the maximum input line length which is identical to $BIG_ID2 except for the last character.	1..127=>'A', 128=>'1'
$BIG_ID2 An identifier the size of the maximum input line length which is identical to $BIG_ID1 except for the last character.	1..127=>'A', 128=>'2'
$BIG_ID3 An identifier the size of the maximum input line length which is identical to $BIG_ID4 except for a character near the middle.	1..63=>'A', 64=>'3', 65..128=>'A'
$BIG_ID4 An identifier the size of the maximum input line length which is identical to $BIG_ID3 except for a character near the middle.	1..63=>'A', 64=>'4', 65..128=>'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..125=>'0', 126..128=>'298'
$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..122=>0, 123..128=>'69.0E1'

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	"1..64=>'A'"
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	"1..63=>'A', 64=>'1'"
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	1..108=>' '
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	65536
$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	2147483647
$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	8
$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	UNIX
$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	0.000_000_000_931_322_574_615_478_515_625
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	40
$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_TYPE
$FLOAT_NAME The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	NO_SUCH_TYPE

$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	2147483647.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	2147483648.0
$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	4
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	MUCH_TOO_LONG_NAME_FOR_A_FILE
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	MUCH_TOO_LONG_NAME_FOR_A_FILE_1
$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 DURATION.	-2147483647.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-2147483648.0

$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	0
$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	30
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	128
$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
$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..125=>'0', 126..128=>'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..124=>'0', 125..128=>'F.E:'
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	"1..125=>'A', 126=>'1'"
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648

$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	32
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	BYTE-INTEGER
$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	UNIX
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFD#
$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.	2147483647
$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma STORAGE_UNIT, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	UNIX
$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.	32

$TICK A real literal whose value is SYSTEM.TICK.	0.01

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 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.

E28005C This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

A39005G This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E This test contains an unitended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84N & M, & CD5011O [5 tests]
These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B This test wrongly expects an implicitly declared subprogram to be at the the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.