UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(2)

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:** TLD Systems Ltd., TLD HP/1750A Ada Compiler System, Version 1.3.0, Hewlett-Packard 9000/350 and HP 64000 with TASCO MDC281 Emulator (Host) and (Target), 880829W1.09158	5. TYPE OF REPORT & PERIOD COVERED 3 Sept. 1988 to 3 Sept. 1988 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson Air Force Base Dayton, OH	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson Air Force Base Dayton, OH	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) Wright-Patterson Air Force Base Dayton, OH	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)

TLD Systems, Ltd., TLD HP/1750A Ada Compiler System, Version 1.3.0, Wright-Patterson AFB, Hewlett-Packard 9000/350 under HP-UX, Version 6.0 (Host) to HP 64000 with TASCO MDC281 Emulator using the TLD 1750A Single Program Kernel (Target), ACVC 1.9.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE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

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AVF Control Number: AVF-VSR-209.1288
88-05-31-TLD

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 880829W1.09158
TLD Systems, Ltd.
TLD HP/1750A Ada Compiler System, Version 1.3.0
Hewlett-Packard 9000/350 and HP 64000 with TASCO MDC281 Emulator

Completion of On-Site Testing:
3 September 1988

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

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

Ada Compiler Validation Summary Report:

Compiler Name: TLD HP/1750A Ada Compiler System, Version 1.3.0

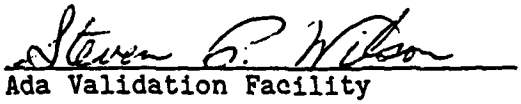
Certificate Number: 880829W1.09158

Host:
Hewlett-Packard 9000/350 under
HP-UX, Version 6.0

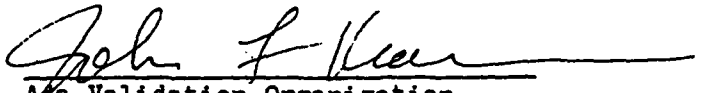
Target:
HP 64000 with TASCO MDC281 Emulator
using the TLD 1750A Single Program
Kernel

Testing Completed 3 September 1988 Using ACVC 1.9

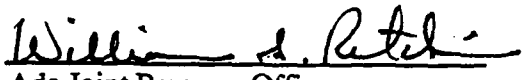
This report has been reviewed and is approved.



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



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



Ada Joint Program Office
William S. Ritchie
Acting Director
Department of Defense
Washington, DC 20301

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

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-2
1.2 USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3 REFERENCES 1-3
1.4 DEFINITION OF TERMS 1-3
1.5 ACVC TEST CLASSES 1-4

CHAPTER 2 CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED 2-1
2.2 IMPLEMENTATION CHARACTERISTICS 2-2

CHAPTER 3 TEST INFORMATION

3.1 TEST RESULTS 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER 3-2
3.4 WITHDRAWN TESTS 3-2
3.5 INAPPLICABLE TESTS 3-2
3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS . . 3-4
3.7 ADDITIONAL TESTING INFORMATION 3-5
3.7.1 Prevalidation 3-5
3.7.2 Test Method 3-5
3.7.3 Test Site 3-6

APPENDIX A DECLARATION OF CONFORMANCE

APPENDIX B APPENDIX F OF THE Ada STANDARD

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS

CHAPTER 1

INTRODUCTION

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

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

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

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

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

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 3 September 1988 at Torrance, 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/SCEL
Wright-Patterson AFB OH 45433-6503

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

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

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

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

support for Ada validations to ensure consistent practices.

Compiler — A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test — An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

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

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

Target — The computer for which a compiler generates code.

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

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

1.5 ACVC TEST CLASSES

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

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

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

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

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

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

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

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

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and

place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

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

Compiler: TLD HP/1750A Ada Compiler System, Version 1.3.0

ACVC Version: 1.9

Certificate Number: 880829W1.09158

Host Computer:

Machine:	Hewlett-Packard 9000/350
Operating System:	HP-UX, Version 6.0
Memory Size:	16 Mbytes

Target Computer:

Machine:	HP 64000 with TASCO MDC281 Emulator
Operating System:	TLD 1750A Single Program Kernel
Memory Size:	64K 16 bit words

Communications Network: Hewlett-Packard HP-1B Intermodule Bus

2.2 IMPLEMENTATION CHARACTERISTICS

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

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 6 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

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

- Based literals.

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

- Expression evaluation.

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

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

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

No exception is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

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

Apparently underflow is gradual. (See tests C45524A..Z.)

- Rounding.

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

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

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

- Array types.

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

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

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

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

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

  A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

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

- Discriminated types.

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

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

- Aggregates.

  In the evaluation of a multi-dimensional aggregate, index subtype checks appear to be made as choices are evaluated. (See tests C43207A and C43207B.)

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

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

- Representation clauses.

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

For this implementation:

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J and C35508M..N.)

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

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

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

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

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

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

- Pragmas.

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

- Input/output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. However, when SEQUENTIAL_IO is instantiated with unconstrained array types and record types with discriminants without defaults, then each call to CREATE raises USE_ERROR. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. However, when DIRECT_IO is instantiated with unconstrained array types and record types with discriminants without defaults, then each call to CREATE raises USE_ERROR. (See tests AE2101H,

EE2401D, and EE2401G.)

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

- Generics.

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

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

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

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 500 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 285 executable tests that use floating-point precision exceeding that supported by the implementation and 174 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 6 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	107	1048	1370	15	11	44	2595
Inapplicable	3	3	483	2	7	2	500
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	184	468	490	245	164	98	140	327	131	36	234	3	75	2595
Inapplicable	20	104	184	3	2	0	3	0	6	0	0	0	178	500
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

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

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

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, 500 tests were inapplicable for the reasons indicated:

- C35702A uses SHORT_FLOAT which is not supported by this implementation.
- A39005C and C87B62B use length clauses with STORAGE_SIZE specifications for access types which are not supported by this implementation.

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

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

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	

- C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.

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

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

- D64005F and D64005G use nested procedures as subunits to a level of 10 which exceeds the capacity of the compiler.

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

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

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

- EE2201D and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. When CREATE is called, USE_ERROR is raised.

- EE2401D and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. When CREATE is called, USE_ERROR is raised.

- The following 174 tests are inapplicable because sequential, text, and direct access files are not supported:

CE2102C	CE2102G..H(2)	CE2102K	CE2104A..D(4)
CE2105A..B(2)	CE2106A..B(2)	CE2107A..I(9)	CE2108A..D(4)
CE2109A..C(3)	CE2110A..C(3)	CE2111A..E(5)	CE2111G..H(2)
CE2115A..B(2)	CE2201A..C(3)	CE2201F..G(2)	CE2204A..B(2)
CE2208B	CE2210A	CE2401A..C(3)	CE2401E..F(2)
CE2404A	CE2405B	CE2406A	CE2407A
CE2408A	CE2409A	CE2410A	CE2411A
AE3101A	CE3102B	EE3102C	CE3103A
CE3104A	CE3107A	CE3108A..B(2)	CE3109A
CE3110A	CE3111A..E(5)	CE3112A..B(2)	CE3114A..B(2)
CE3115A	CE3203A	CE3301A..C(3)	CE3302A
CE3305A	CE3402A..D(4)	CE3403A..C(3)	CE3403E..F(2)
CE3404A..C(3)	CE3405A..D(4)	CE3406A..D(4)	CE3407A..C(3)
CE3408A..C(3)	CE3409A	CE3409C..F(4)	CE3410A
CE3410C..F(4)	CE3411A	CE3412A	CE3413A
CE3413C	CE3602A..D(4)	CE3603A	CE3604A
CE3605A..E(5)	CE3606A..B(2)	CE3704A..B(2)	CE3704D..F(3)
CE3704M..O(3)	CE3706D	CE3706F	CE3804A..E(5)
CE3804G	CE3804I	CE3804K	CE3804M
CE3805A..B(2)	CE3806A	CE3806D..E(2)	CE3905A..C(3)
CE3905L	CE3906A..C(3)	CE3906E..F(2)	

  Results of running a subset of these tests showed that the proper exceptions are raised for unsupported file operations.

- The following 285 tests require a floating-point accuracy that exceeds the maximum of 9 digits supported by this implementation:

C24113F..Y (20 tests)	C35705F..Y (20 tests)
C35706F..Y (20 tests)	C35707F..Y (20 tests)
C35708F..Y (20 tests)	C35802F..Z (21 tests)
C45241F..Y (20 tests)	C45321F..Y (20 tests)
C45421F..Y (20 tests)	C45521F..Z (21 tests)
C45524F..Z (21 tests)	C45621F..Z (21 tests)
C45641F..Y (20 tests)	C46012F..Z (21 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

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

one exception instead of another).

Modifications were required for 6 Class B tests.

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

B24009A	B44004D	B49003A
B49005A	B59001A	B59001E

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the TLD HP/1750A Ada Compiler System, Version 1.3.0, 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 TLD HP/1750A Ada Compiler System, Version 1.3.0, using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a Hewlett-Packard 9000/350 host operating under HP-UX, Version 6.0, and a HP 64000 with TASCO MDC281 Emulator target using the TLD 1750A Single Program Kernel. The host and target computers were linked via a Hewlett-Packard HP-1B Intermodule Bus.

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

The contents of the magnetic tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled and linked on the Hewlett-Packard 9000/350, and all executable tests were run on the HP 64000 using the TASCO MDC281 Emulator. Object files were linked on the host computer, and executable images were transferred to the target computer via the HP-1B bus. Results were then transferred back to the Hewlett-Packard 9000/350, where they were saved on magnetic tape and then loaded onto a Data General MV 8000 for printing.

The compiler was tested using command scripts provided by TLD Systems, Ltd. and reviewed by the validation team. The compiler was tested using all default switch settings except for the following:

Switch	Effect
MDC281	Indicate target processor.
DEL	Delete dead assignments.
MAKLIB	Point to parent library.
ROOTLIB	Indicate runtime library.
NOINFO	Suppress information messages.
NOPHASE	Suppress logging of compiler phase timing on output.

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

3.7.3 Test Site

Testing was conducted at Torrance, CA and was completed on 3 September 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

TLD Systems, Ltd. has submitted the following Declaration of Conformance concerning the TLD HP/1750A Ada Compiler System, Version 1.3.0.

DECLARATION OF CONFORMANCE

Compiler Implementor: TLD Systems, Ltd.
Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB, OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version: 1.9

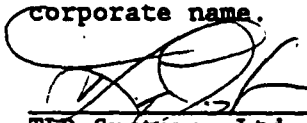
Base Configuration

Base Compiler Name: TLD HP/1750A Ada Compiler System Version: 1.3.0

Host Architecture ISA: Hewlett Packard 9000/350
OS&VER #: HP-UX Version 6.0
Target Architecture ISA: HP 64000 with the TASCO MDC281 Emulator
OS&VER #: TLD 1750A Single Program Kernel, Version: 1.3.0

Implementor's Declaration

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

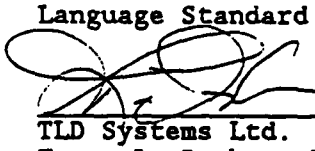


Date: 30 August 1988

TLD Systems, Ltd.
Terry L. Dunbar, President

Owner's Declaration

I, the undersigned, representing TLD Systems, Ltd., take full responsibility for implementation and maintenance of the Ada compiler 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.



Date: 30 August 1988

TLD Systems Ltd.
Terry L. Dunbar, President

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the TLD HP/1750A Ada Compiler System, Version 1.3.0, are described in the following sections, which discuss topics in Appendix F of the Ada Standard. Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

    ...

    type INTEGER is range -32768 .. 32767;
    type LONG_INTEGER is range -1073741824 .. 1073741823;

    type FLOAT is digits 6 range -1.0*2.0**127 .. 0.999999*2.0**127;
    type LONG_FLOAT is digits 9 range -1.0*2.0**127 .. 0.999999*2.0**127;

    type DURATION is delta 2.0**(-12) range -86400.0 .. 86400.0;

    ...

end STANDARD;
```

The Ada language definition allows for certain machine_dependencies in a controlled manner. No machine-dependent syntax of semantic extensions or restrictions are allowed. The only allowed implementation-dependencies correspond to implementaton-dependent pragmas and attributes, certain machine-dependent conventions as mentioned in chapter 13, and certain allowed restrictions on representation clauses.

The full definition of the implementation-dependent characteristics of the TLD VAX/1750A Ada Compiler System is presented in this Appendix F.

Implementation-Dependent Pragmas

The TLD ACS supports pragma identifiers Interface, its logical complement, Export; and source maintenance commands in the form of pragma syntax: If, Elsif, Else, EndIf, and Include.

```
Pragma Export(Language, Name {, Optional_String});
```

This pragma is used to identify a static object name or procedure name that is to be exposed to the linker for reference by object modules written in other than the Ada Language. The third parameter is the name by which the Ada entity named by the second parameter may be referenced rather than a name assigned by the compiler. The only language supported at present is Assembly. If the entity named is a subprogram, this pragma must be placed in the declarative region of the subprogram. If the entity named is an Ada object, this pragma must appear following the declaration of the object but within the same declarative region as the object.

```
Pragma Include ( File_Path_Name_String );
```

This directive in the form of a language pragma is processed by the Ada Compiler Front End to permit inclusion of another source file in place of the pragma. This pragma may occur any place a language defined pragma, statement, or declaration may occur. This directive is used to facilitate source program portability and configurability.

```
Pragma If ( Compile_Time_Expression );
Pragma Elsif (Compile_Time_Expression );
Pragma Else;
Pragma Endif;
```

These source maintenance directives may be used to enclose conditionally compiled source to enhance program portability and configuration adaptation. These directives may occur at the place that language defined pragmas, statements, or declarations may occur. Source occurring following these pragmas will be compiled or ignored similar to the semantics of the corresponding Ada statements depending upon whether the compile time

expression is true or false, respectively. The primary difference between these directives and the corresponding Ada statements are that the directives may enclose declarations and other pragmas.

Implementation-Dependent Attributes

None.

Representation Clause Restrictions

Pragma Pack is not supported.

Length clauses are supported for 'size applied to objects other than task and access type objects and denote the number of bits allocated to the object.

Length clauses are not supported for 'Storage_Size when applied to access types.

Length clauses are supported for 'Storage_Size when applied to a task type and denote the number of words of stack to be allocated to the task.

Length clauses are not supported for 'Small.

Enumeration representation clauses are supported for value ranges of Integer'First to Integer'Last.

Record representation clauses are supported to arrange record components within a record. Record components may not be specified to cross a word boundary unless they are arranged to encompass two or more whole words. A record component of type record that has itself been "rep specificationed" may only be allocated at bit 0. Bits are numbered from left to right with bit 0 indicating the sign bit.

The alignment clause is not supported.

Address clauses are supported for both variable and constant objects and designate the virtual address of the object. The TLD Ada Compiler System treats the address specification as a means to access objects allocated by other than Ada means and accordingly does not treat the clause as a request to allocate the object at the indicated address.

Address clauses are not supported for packages, tasks, or task entries.

Implementation-Generated Names

The TLD Ada Compiler System defines no implementation dependent names for compiler generated components.

Address Clause Expressions

Address expression values and type Address represent a location in logical memory, (in the program's current address state). For objects, the address specifies a location within the 64K word logical operand space. The 'Address attribute applied to a subprogram represents a 16 bit word address within the logical instruction space.

Unchecked Conversion Restrictions

None

I/O Package Characteristics

The following implementation-defined types are declared in Text_Io.

```
subtype Count is integer range 0 .. 511;

subtype Field is Integer range 0 .. 127;
```

Package Standard

The implementation-defined types of package Standard are:

```
type Integer is         range -32_768 .. 32_767;
type Long_Integer is    range -1_073_741_824 .. 1_073_741_823;
type Float is digits 6 range -1.0*2.0**127 .. 0.999999*2.0**127;
type Long_Float is digits 9 range -1.0*2.0**127 .. 0.999999*2.0**127;
type Duration is delta 2.0**(-12) range -86_400.0..86_400.0;
```

Other System Dependencies

LRM Chapter 1.

None.

LRM Chapter 2.

Maximum source line length -- 120 characters.

Source line terminator -- Determined by the Editor used.

Maximum name length -- 120 characters.

External representation of name characters.

Maximum String literal -- 120 characters.

LRM Chapter 3.

LRM defined pragmas are recognized and processed as follows:

Controlled -- Has no effect.

Elaborate -- As described in the LRM.

Inline -- Not presently supported.

Interface -- Supported as a means of importing foreign language components into the Ada Program Library. May be applied either to a subprogram declaration as being specially implemented, -- read Interface as Import --, or to an object that has been declared elsewhere. Interface languages supported are System for producing a call obeying the standard calling conventions except that the BEX instruction is used to cause a software interrupt into the kernel supervisor mode; Assembly for calling assembly language routines; and Mil-Std-1750A for defining built in instruction procedures. An optional third parameter is used to define a name other than the name of the Ada subprogram for interfacing with the linker.

List -- As defined in the LRM.

Memory Size -- Has no effect.

Optimize -- Has no effect. Optimization controlled by compiler command option.

Pack -- Has no effect.

Page -- As defined in the LRM.

Priority -- As defined in the LRM. Priority may range from 0 to 16366. Default priority is 1.

Shared -- As defined in the LRM. May be applied to scalar objects only.

Storage_Unit -- Has no effect.

Suppress -- As defined in the LRM for suppressing checks; all standard checks may be suppressed individually plus "Exception_Info" and "All_Checks". Suppression of Exception_Info eliminates data used to provide symbolic debug information in the event of an unhandled exception. The All_Checks selection eliminates all checks with a single pragma. In addition to the pragma, a compiler permits control of check suppression by command line option without the necessity of source changes.

System_Name -- Has no effect.

Number declarations are not assigned addresses and their names are not permitted as a prefix to the 'address attribute. (Clarification only).

Objects are allocated by the compiler to occupy one or more 16 bit 1750A words. Only in the presence record representation clauses are objects allocated to less than a word.

Except for access objects, uninitialized objects contain an undefined value. An attempt to reference the value of an uninitialized object is not detected.

The maximum number of enumeration literals of all types is limited only by available symbol table space.

The predefined integer types are:

Integer range -32_768 .. 32_767 and is implemented as a 1750A single precision fixed point data.
Long_Integer range -1_073_741_824 .. 1_073_741_823 and implemented as 1750A double precision data.
Short_Integer is not supported.

System.Min_Int is -1_073_741_824.
System.Max_Int is 1_073_741_823.
} testing proved -2**31 & 2**31 - 1 are the actual values.

The predefined real types are:

Float digits 6.
Long_Float digits 9.
Short_Float is not presently supported.

System.Max_Digits is presently 9 and is implemented as 1750A 48-bit floating point data.

Fixed point is implemented as 1750A single and double precision data as is appropriate for the range and delta.

On the 1750A, index constraints as well as other address values such as access types are limited to an unsigned range of 0 .. 65_536 or a signed range of -32_768 .. 32_767.

The maximum array size is limited to the size of virtual memory -- 64K words.

The maximum String length is the same as for other arrays.

Access objects are implemented as an unsigned 16 bit 1750A integer. The access literal Null is implementated as one word of zero on the 1750A.

There is no limit on the number of dimensions of an array type. Array types are passed as parameters opposite unconstrained formal parameters using a 3 word dope vector illustrated below:

Word address of first element
Low bound value of first dimension
Upper bound value of first dimension

Additional dimension bounds follow immediately for arrays with more than one dimension.

LRM Chapter 4.

Machine_Overflows is True for the 1750A.

Pragma Controlled has no effect for the TLD VAX/1750A Compiler since garbage collection is never performed.

LRM Chapter 5.

The maximum number of statements in an Ada source program is undefined and limited only by Symbol Table space.

Case statements unless they are quite sparse, are allocated as indexed jump vectors and are, therefore, quite fast.

Loop statements with a for implementation scheme are implemented most efficiently on the 1750A if the range is in reverse and down to zero.

Data declared in block statements on the 1750A is elaborated as part of its containing scope.

LRM Chapter 6.

Arrays, records and task types are passed on the 1750A by reference.

Pragma Inline is not presently supported for subprograms.

LRM Chapter 7.

Package elaboration is performed dynamically permitting a warm restart without the necessity to reload the program.

LRM Chapter 8.

LRM Chapter 9.

Task objects are implemented as access types pointing to a Task Information Block (TIB).

Type Time in package Calendar is declared as a record containing two double precision integer values: the date in days and the real time clock.

Pragma Priority is supported with a value of 1 to 16366.

Pragma Shared is supported for scalar objects.

LRM Chapter 10.

Multiple Ada Program Libraries are supported with each library containing an optional ancester library. The predefined packages are contained in the TLD standard library, ADA.LIB.

LRM Chapter 11.

Exceptions are implemented by the TLD Ada Compiler System to take advantage of the normal policy in embedded computer system design to utilize only 50% of the duty cycle. By executing a small number of instructions in the prologue of a procedure or block containing an exception handler then at the occurrence of an exception, a branch may be taken directly to a handler rather than performing the time consuming code of unwinding procedure calls and stack frames. The philosophy taken is that an exception signals an exceptional condition, perhaps a serious one involving recovery or reconfiguration, and that quick response in this situation is more important and worth the tradeoff in a real time environment.

LRM Chapter 12.

A single generic instance is generated for a generic body. Generic specifications and bodies need not be compiled together nor need a body be compiled prior to the compilation of an instantiation. Because of the single expansion, this implementation of generics tend to be more favorable on the 1750A because of the usual space savings achieved. To achieve this tradeoff, the instantiations must by nature be more general and are, therefore, somewhat less efficient timewise.

LRM Chapter 13.

Representation clause support and restrictions are defined above.

A comprehensive Machine_Code package is provided and supported.

Unchecked_Deallocation and Unchecked_Conversion are supported.

The implementation dependent attributes are all supported except 'Storage_Size for an access type.

LRM Chapter 14.

File I/O operations are not supported for the 1750A. Text_Io and Low_Level_Io are supported.

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..119 => 'A', 120 => '1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..119 => 'A', 120 => '2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..80 | 82..120 => 'A', 81 => '3')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..80 | 82..120 => 'A', 81 => '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..117 => '0', 118..120 => "298")

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..115 => '0', 116..120 => "690.0")
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1..60 => 'A')
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1..59 => 'A', 60 => '1')
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..100 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	511
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	127
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	"BAD-CHARS#.%!X"
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	"WILD-CHAR*.NAM"
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	90000.0

Name and Meaning	Value
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131073.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	"BADCHAR@.!"
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	"THISFILENAMEWOULDBEPERFECTLY" & "LEGALIFITWERENOTSOLONG.SOTHERE"
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32768
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-900000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131073.0
$MAX_DIGITS Maximum digits supported for floating-point types.	9
$MAX_IN_LEN Maximum input line length permitted by the implementation.	120
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648

TEST PARAMETERS

Name and Meaning	Value
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2 => "2:", 3..118 => '0', 119..120 => "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..116 => '0', 117..120 => "F.E:")
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1..2 => "A", 3..118 => 'A', 119..120 => "A")
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	SHORT_SHORT_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- B28003A: A basic declaration (line 36) incorrectly follows a later declaration.
- E28005C: This test requires that "PRAGMA LIST (ON);" not appear in a listing that has been suspended by a previous "PRAGMA LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the AJPO.
- C34004A: The expression in line 168 yields a value outside the range of the target type T, but there is no handler for CONSTRAINT_ERROR.
- C35502P: The equality operators in lines 62 and 69 should be inequality operators.
- A35902C: The assignment in line 17 of the nominal upper bound of a fixed-point type to an object raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.
- C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.
- C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may, in fact, raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

WITHDRAWN TESTS

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

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

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

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

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

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

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

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

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

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

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

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

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