**Ada Compiler Validation Summary Report:** TeleSoft, TeleGen2 Ada Development System, Version 3.20 MicroVAX II (Host) and (Target), 88Q21921.09031

**Author(s):** Wright-Patterson AFB
Dayton, OH, USA

**Performing Organization and Address:**
Wright-Patterson AFB
Dayton, OH, USA

**CONTROLLING OFFICE NAME AND ADDRESS:**
Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

**REPORT NUMBER:**

**PERIOD COVERED:**

**AUTHOR(s):** Wright-Patterson AFB
Dayton, OH, USA

**Program Element, Project, Task Area & Work Unit Numbers:**

**RECIPIENT'S CATALOG NUMBER:**

**TYPE OF REPORT & PERIOD COVERED:**

**PERFORMING ORG. REPORT NUMBER:**

**REPORT DATE:**

**NUMBER OF PAGES:**

**SECURITY CLASS (of this report):** UNCLASSIFIED

**DECLASSIFICATION/DOWNGRADING SCHEDULE:** N/A

**DISTRIBUTION STATEMENT (of this report):**
Approved for public release; distribution unlimited.

**DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report):**
UNCLASSIFIED

**SUPPLEMENTARY NOTES:**

**KEYWORDS:**

**ABSTRACT:**
Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 880219W1.09031
TELESOFI
TeleGen2 Ada Development System, Version 3.20
MicroVAX II

Completion of On-Site Testing:
26 February 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

Availability Codes
Dist  Avail and/or Special
A-1

Accession For
NTIS CRA&I
DTIC TAB
Unannounced
Justification

By
Distribution

89 8 23 187
Ada Compiler Validation Summary Report:

Compiler Name: TeleGen2 Ada Development System, Version 3.20
Certificate Number: 880219W1.09031

Host: MicroVAX II under VMS, Version 4.6
Target: MicroVAX II under VMS, Version 4.6
Testing Completed 26 February 1988 Using ACVC 1.9

This report has been reviewed and is approved.

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

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

William S. Ritchie
Ada Joint Program Office
William S. Ritchie
Acting Director
Department of Defense
Washington DC 20301
Ada Compiler Validation Summary Report:

Compiler Name: TeleGen2 Ada Development System, Version 3.20
Certificate Number: 880219W1.09031

Host: MicroVAX II under VMS, Version 4.6
Target: MicroVAX II under VMS, Version 4.6
Testing Completed 26 February 1988 Using ACVC 1.9

This report has been reviewed and is approved.

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

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

William S. Ritchie
Ada Joint Program Office
Acting Director
Department of Defense
Washington DC 20301
# 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 Withdrown 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 .................................................................. 3-7

## Appendix A: Declaration of Conformance

## Appendix B: Appendix F of the Ada Standard

## Appendix C: Test Parameters

## Appendix D: Withdrown 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 26 February 1988 at TELESOFT, San Diego, 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


1.4 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-dddd.

Ada Standard  

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
INTRODUCTION

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.

Language Maintenance Panel The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and possible changes to the ANSI/MIL-STD for Ada.

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

Target The computer for which a compiler generates code.

Test An Ada 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.
INTRODUCTION

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

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.
CHAPTER 2
CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

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

ACVC Version: 1.9
Certificate Number: 880219W1.09031

Host Computer:

Machine: MicroVAX II
Operating System: VMS, Version 4.6
Memory Size: 10 megabytes

The host system has a TK50 95MB streaming tape drive, a reel-to-reel magnetic tape drive, and Local Area VAX Cluster.

Target Computer:

Machine: MicroVAX II
Operating System: VMS, Version 4.6
Memory Size: 10 megabytes

Communications Network: N/A
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 at least 65 levels, block statements nested to at least 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed \texttt{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 \texttt{LONG\_INTEGER} and \texttt{LONG\_FLOAT} in the package \texttt{STANDARD}. (See tests B86001C and B86001D.)

. Based literals.

An implementation is allowed to reject a based literal with a value exceeding \texttt{SYSTEM\_MAX\_INT} during compilation, or it may raise \texttt{NUMERIC\_ERROR} or \texttt{CONSTRAINT\_ERROR} during execution. This implementation raises \texttt{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.)

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

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

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed-point membership test is outside the range of the base type. (See test C45252A.)

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

Rounding.

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

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

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

Array types.

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

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

No exception is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

No exception 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 no exception. (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 does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

Discriminated types.

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

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

Aggregates.

In the evaluation of a multi-dimensional aggregate, 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.)

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

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 not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications of less than 16 bits for enumeration types are not supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are 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 supported. (See tests A39005E and C87B62C.)

Record representation clauses are not supported unless aligned on 16 bits. (See test A39005G.)

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

Pragmas.

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

Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)
The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

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 TEXT_IO.

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

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

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

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

Overwriting to a sequential file truncates the file to the last element written. (See test CE2208B.)

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

More than one internal file can be associated with each external file for text I/O for reading only. (See tests CE3111A..E (5 tests), CE3111B, and CE3115A.)

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

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

An internal sequential access file and an internal direct access file cannot be associated with a single external file for writing. (See test CE2107E.)

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

Generics.
Generic subprogram declarations and bodies can be compiled in separate compilations only if the body is compiled before any instantiations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations only if the body is compiled before any instantiations. (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, 24 tests had been withdrawn because of test errors. The AVF determined that 346 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. Modifications to the code, processing, or grading for 10 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

<table>
<thead>
<tr>
<th>RESULT</th>
<th>TEST CLASS</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Passed</td>
<td>106</td>
<td>1046</td>
</tr>
<tr>
<td>Inapplicable</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Withdrawn</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>TOTAL</td>
<td>113</td>
<td>1053</td>
</tr>
</tbody>
</table>
3.3 SUMMARY OF TEST RESULTS BY CHAPTER

<table>
<thead>
<tr>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passed</td>
<td>184</td>
<td>184</td>
<td>245</td>
<td>166</td>
<td>98</td>
<td>141</td>
<td>327</td>
<td>132</td>
<td>36</td>
<td>232</td>
<td>3</td>
<td>233</td>
<td>2752</td>
<td></td>
</tr>
<tr>
<td>Inapplicable</td>
<td>20</td>
<td>109</td>
<td>184</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>21</td>
<td>346</td>
</tr>
<tr>
<td>Withdrawn</td>
<td>2</td>
<td>13</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>206</td>
<td>586</td>
<td>677</td>
<td>248</td>
<td>166</td>
<td>99</td>
<td>145</td>
<td>327</td>
<td>137</td>
<td>36</td>
<td>236</td>
<td>4</td>
<td>255</td>
<td>3122</td>
</tr>
</tbody>
</table>

3.4 WITHDRAWN TESTS

The following 24 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

- B28003A
- E28005C
- C34004A
- C35502P
- A35902C
- C35904A
- C35A03E
- C35A03R
- C37213H
- C37213J
- C37215C
- C37215E
- C37215G
- C37215H
- C38102C
- C38104A
- C38106A
- C38108B
- C87B04B
- CC1311B
- BC3105A
- ADIA01A
- CE2401H

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, 346 tests were inapplicable for the reasons indicated:

- A39005B uses a length clause with a SIZE specification of eight bits. The minimum size for an allocated object under this compiler is 16 bits. Therefore, the length clause is rejected.
- A39005G uses a record representation clause which specifies an alignment of eight. This compiler forces an alignment of 16. Therefore, the record representation clause is rejected.
. C35508I..J (2 tests) and C35508M..N (2 tests) use enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1). These clauses are not supported by this compiler.

. C35702A uses SHORT_FLOAT which is not supported by this implementation.

. C35904B raises NUMERIC_ERROR instead of CONSTRAINT_ERROR. NUMERIC_ERROR is raised during elaboration of the fixed-point constraint which occurs prior to the compatibility check that would raise CONSTRAINT_ERROR.

. 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 and B86001D require the substitution of a macro for a predefined numeric type other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such type.

. 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.

. C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

. CA2009C, CA2009F, BC3204C, and BC3205D contain instantiations of generics in cases where the body is not available at the time of the instantiation. As allowed by AI-00408/07, this compiler creates a dependency on the missing body so that when the actual body is compiled, the unit containing the instantiation becomes obsolete.

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

. AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.
AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT.IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

The following 15 tests are inapplicable because associating more than one internal sequential, direct, or text file with the same external file for reading/writing or writing is not allowed:

CE2107B..E(4) CE2107G..I(3) CE2110B CE2111D
CE2111H CE3111B..E(4) CE3114B

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)
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 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 nine Class B tests and one Class E test.

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

B27005A BA3006A BA3006B BA3007B BA3008A
BA3008B BA3013A

The following tests need a 'PRAGMA LIST (ON);' added at the beginning of the source file in order to have a complete source/error listing.

B28001R B28001V E28002D
The following tests were graded using a modified evaluation criteria:

a. C45651A requires that the result of the expression in line 227 be in the range given in line 228; however, this range excludes some acceptable results. This implementation passes all other checks of this test, and the AVO ruled that the test is passed.

b. C46014A contains an assignment that is intended to raise an exception. This implementation performs an optimization--given that the assignment is to a dead variable--and thus does not raise the exception. Because the optimization only affected one check in the test, the AVO ruled that the test should be graded as passed, as all other checks produced the expected results and the test objective was met.

c. C96001A assumes that DURATION'SMALL >= SYSTEM.TICK; however, the Ada Standard does not require such a relation. This implementation executes delay statements with greater accuracy than CALENDAR.CLOCK can resolve, and so the check on line 97 is failed. This implementation passes all other checks of this test, and the AVO ruled that the test is passed.

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 TeleGen2 Ada Development System compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the TeleGen2 Ada Development System compiler using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a cluster of three MicroVAX II hosts/targets operating under VMS, Version 4.6.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.
The contents of the magnetic tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled, linked, and run on the MicroVAX II. Results were printed from the host computer.

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

Class B tests

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMS</td>
<td>choose VAX/VMS to VAX/VMS compiler.</td>
</tr>
<tr>
<td>MONITOR</td>
<td>verbose.</td>
</tr>
<tr>
<td>PROCEED</td>
<td>continue compilation despite errors, no continuation prompting at each error.</td>
</tr>
<tr>
<td>LIST</td>
<td>interspersed listing.</td>
</tr>
<tr>
<td>OPTIMIZE</td>
<td>* see note below.</td>
</tr>
<tr>
<td>VIRTUAL_SPACE</td>
<td>set virtual space of library manager greater than default.</td>
</tr>
</tbody>
</table>

Executable tests

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMS</td>
<td>choose VAX/VMS to VAX/VMS compiler.</td>
</tr>
<tr>
<td>MONITOR</td>
<td>verbose.</td>
</tr>
<tr>
<td>PROCEED</td>
<td>continue compilation despite errors, no continuation prompting at each error.</td>
</tr>
<tr>
<td>OPTIMIZE</td>
<td>* see note below.</td>
</tr>
<tr>
<td>VIRTUAL_SPACE</td>
<td>set virtual space of library manager greater than default.</td>
</tr>
<tr>
<td>BIND</td>
<td>invokes the binder.</td>
</tr>
<tr>
<td>NOSHARE</td>
<td>do not use a sharable runtime.</td>
</tr>
<tr>
<td>LINK=&quot;/NOMAP&quot;</td>
<td>do not generate a link map.</td>
</tr>
<tr>
<td>NOSAVE</td>
<td>do not save the link script.</td>
</tr>
<tr>
<td>OUTPUT='testname'</td>
<td>name of executable file.</td>
</tr>
</tbody>
</table>

The optimizer switch OPTIMIZE is equivalent to "OPTIMIZE=ALL" in which "ALL" stands for "PARALLEL, RECURSE, INLINE, AUTOINLINE".

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARALLEL</td>
<td>indicates that one or more of the subprograms being optimized may be called from parallel tasks.</td>
</tr>
<tr>
<td>RECURSE</td>
<td>indicates that one or more of the subprograms interior to the unit/collection being optimized could be called recursively by an exterior subprogram.</td>
</tr>
<tr>
<td>INLINE</td>
<td>enables inline expansion of those subprograms marked with an INLINE pragma or generated</td>
</tr>
</tbody>
</table>
AUTOINLINE enables automatic inline expansion of any subprogram called from only one place, as well as those marked by an INLINE pragma or generated by the compiler.

Tests were compiled, linked, and executed (as appropriate) using a cluster of three host/target computers. 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 TELESOF T, San Diego, CA and was completed on 26 February 1988.
TELESOFT has submitted the following Declaration of Conformance concerning the TeleGen2 Ada Development System compiler.
DECLARATION OF CONFORMANCE

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

Base Configuration

Base Compiler Name: TeleGen2 Ada Development System
Version: 3.20
Host Architecture ISA: MicroVAX II
OS&VER #: VMS, Version 4.6
Target Architecture ISA: MicroVAX II
OS&VER #: VMS, Version 4.6

Implementer's Declaration
I, the undersigned, representing TELESOF, 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 TELESOF 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.

Raymond A. Parra, Director, Contracts/Legal

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

Raymond A. Parra, Director, Contracts/Legal

Date: 2/26/86
APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the TeleGen2 Ada Development System compiler, Version 3.20, are described in the following sections, which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

package STANDARD is

... type INTEGER is range -32768 .. 32767;
    type LONGINTEGER is range -2147483648 .. 2147483647;
    type FLOAT is digits 6 range -1.93428E+25 .. 1.93428E+25;
    type LONG_FLOAT is digits 9 range -2.1267647922655E+37 .. 2.1267647922655E+37;
    type DURATION is delta 2#1.0#E-14 range -86400 .. 86400;

... end STANDARD;
1. Predefined Pragmas.

pragma LIST(ON/OFF);
It may appear anywhere a pragma is allowed. The pragma has the effect of generating the source compilation. The listing will begin at the first PRAGMA LIST(ON) statement if no previous PRAGMA LIST(OFF) statement was encountered. Otherwise, the listing will begin at the top of the source.

2. Implementation-Dependent Pragmas.
The following implementation-dependent pragmas are supported:

pragma COMMENT (<string_literal>);
It may appear within a compilation unit. The pragma comment has the effect of embedding the given sequence of characters in the object code of the compilation unit.

pragma LINKNAME (<subprogram name>, <string_literal>);
It may appear in any declaration section of a unit. This pragma must also appear directly after an interface pragma for the same <subprogram name>. The pragma linkname has the effect of making <string_literal> apparent to the linker.

3. Implementation-Dependent Attributes.
There are no implementation-dependent attributes supported.

4. Specification of Package SYSTEM.

package SYSTEM is
  type ADDRESS is access INTEGER;
type SUBPROGRAM_VALUE is private;

type NAME is (TeleGen2);

SYSTEM NAME : constant NAME := TeleGen2;
STORAGE_UNIT : constant := 8;
MEMORY_SIZE : constant := 2147483647;

-- System-dependent named numbers:
MIN_INT : constant := -2147483648;
MAX_INT : constant := 2147483647;
MAX_DIGITS : constant := 9;
MAX_MANTISSA : constant := 31;
FINE_DELTA : constant := 1.0 / (2**MAX_MANTISSA);
TICK : constant := 10.0E-3;

-- Other system-dependent declarations:
APPENDIX F OF THE Ada STANDARD

subtype PRIORITY is INTEGER range 0..63;

MAX_TEXT_IO_COUNT : constant := 32676;
MAX_TEXT_IO_FIELD : constant := 1000;

private
  type SUBPROGRAM_VALUE is
    record
      PROC_ADDR : ADDRESS;
      STATIC_LINK : ADDRESS;
      GLOBAL_FRAME : ADDRESS;
    end record;
  end SYSTEM;

5. Restrictions on Representation Clauses.
The compiler supports the following representation clauses:

Length Clauses : for enumeration and derived integer
  types 'SIZE attribute.
  for access types 'STORAGE_SIZE
  attribute.
  for task types 'STORAGE_SIZE attribute.
  for fixed-point types 'SMALL attribute.

Enumeration Clauses : for character and enumeration
  types other than character and
  boolean.

Record Representation Clauses.

Address Clauses : for objects and entries.

The compiler does NOT support the following representation
  clauses:

Enumeration Clauses : for boolean.

Address Clauses : for subprograms, packages, and tasks.

This compiler contains a restriction that allocated objects
  must have a minimum allocation size of 16 bits.

6. Implementation-Dependent Naming.
There are no implementation-generated names denoting
  implementation-dependent components.

7. Interpretation of Expressions in Address Clauses.
Expressions that appear in address specifications are
  interpreted as the first storage unit of the object.
8. Restrictions on Unchecked Conversions.
Unchecked conversions are allowed between any types unless
the target type is an unconstrained record or array type.

Instantiations of DIRECT.IO and SEQUENTIAL.IO are supported
with the following exceptions:

- Unconstrained array types.
- Unconstrained types with discriminants without default
  values.

In DIRECT.IO the type COUNT is defined as
type COUNT is range 0..2_147_483_647;

In TEXT.IO the type COUNT is defined as
type COUNT is range 0..32766;

In TEXT.IO the type FIELD is defined as
subtype FIELD is INTEGER range 0..1000;
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.

<table>
<thead>
<tr>
<th>Name and Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BIG_ID1</td>
<td>(1..199 =&gt; 'A', 200 =&gt; '1')</td>
</tr>
<tr>
<td>Identifier the size of the maximum input line length with varying last character.</td>
<td></td>
</tr>
<tr>
<td>$BIG_ID2</td>
<td>(1..199 =&gt; 'A', 200 =&gt; '2')</td>
</tr>
<tr>
<td>Identifier the size of the maximum input line length with varying last character.</td>
<td></td>
</tr>
<tr>
<td>$BIG_ID3</td>
<td>(1..100 =&gt; 'A', 101 =&gt; '3', 102..200 =&gt; 'A')</td>
</tr>
<tr>
<td>Identifier the size of the maximum input line length with varying middle character.</td>
<td></td>
</tr>
<tr>
<td>$BIG_ID4</td>
<td>(1..100 =&gt; 'A', 101 =&gt; '4', 102..200 =&gt; 'A')</td>
</tr>
<tr>
<td>Identifier the size of the maximum input line length with varying middle character.</td>
<td></td>
</tr>
<tr>
<td>$BIG_INT_LIT</td>
<td>(1..197 =&gt; '0', 198..200 =&gt; &quot;298&quot;)</td>
</tr>
<tr>
<td>An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.</td>
<td></td>
</tr>
</tbody>
</table>
### TEST PARAMETERS

<table>
<thead>
<tr>
<th>Name and Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BIG_REAL_LIT</td>
<td>(1..194 =&gt; '0', 195..200 =&gt; &quot;69.0E1&quot;)</td>
</tr>
<tr>
<td>$BIG_STRING1</td>
<td>(1 =&gt; '&quot;', 2..102 =&gt; 'A', 103 =&gt; '&quot;&quot;)</td>
</tr>
<tr>
<td>$BIG_STRING2</td>
<td>(1 =&gt; '&quot;', 2..101 =&gt; 'A', 102..103 =&gt; &quot;1&quot;&quot;)</td>
</tr>
<tr>
<td>$BLANKS</td>
<td>(1..180 =&gt; ' ')</td>
</tr>
<tr>
<td>$COUNT_LAST</td>
<td>32766</td>
</tr>
<tr>
<td>$FIELD_LAST</td>
<td>1000</td>
</tr>
<tr>
<td>$FILE_NAME_WITH_BAD_CHARS</td>
<td>&quot;BAD-CHARS}]!@#$&amp;-Y&quot;</td>
</tr>
<tr>
<td>$FILE_NAME_WITH_WILD_CARD_CHAR</td>
<td>&quot;WILD-CHAR#.NAM&quot;</td>
</tr>
<tr>
<td>$GREATER_THAN_DURATION</td>
<td>100,000.0</td>
</tr>
</tbody>
</table>

A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.

A string literal which when concatenated with BIG\_STRING2 yields the image of BIG\_ID1.

A string literal which when concatenated to the end of BIG\_STRING1 yields the image of BIG\_ID1.

A sequence of blanks twenty characters less than the size of the maximum line length.

A universal integer literal whose value is TEXT\_IO\_COUNT\_LAST.

A universal integer literal whose value is TEXT\_IO\_FIELD\_LAST.

An external file name that either contains invalid characters or is too long.

An external file name that either contains a wild card character or is too long.

A universal real literal that lies between DURATION\_BASE\_LAST and DURATION\_LAST or any value in the range of DURATION.
<table>
<thead>
<tr>
<th>Name and Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GREATER_THAN_DURATIONS_BASE_LAST</td>
<td>131.073.0</td>
</tr>
<tr>
<td>A universal real literal that is greater than DURATION'BASE'LAST.</td>
<td></td>
</tr>
<tr>
<td>$ILLEGAL_EXTERNAL_FILE_NAME1</td>
<td>&quot;BADCHAR*/%&quot;</td>
</tr>
<tr>
<td>An external file name which contains invalid characters.</td>
<td></td>
</tr>
<tr>
<td>$ILLEGAL_EXTERNAL_FILE_NAME2</td>
<td>(1..256 =&gt; 'A')</td>
</tr>
<tr>
<td>An external file name which is too long.</td>
<td></td>
</tr>
<tr>
<td>$INTEGER_FIRST</td>
<td>-32768</td>
</tr>
<tr>
<td>A universal integer literal whose value is INTEGER'FIRST.</td>
<td></td>
</tr>
<tr>
<td>$INTEGER_LAST</td>
<td>32767</td>
</tr>
<tr>
<td>A universal integer literal whose value is INTEGER'LAST.</td>
<td></td>
</tr>
<tr>
<td>$INTEGER_LAST_PLUS_1</td>
<td>32768</td>
</tr>
<tr>
<td>A universal integer literal whose value is INTEGER'LAST + 1.</td>
<td></td>
</tr>
<tr>
<td>$LESS_THAN_DURATIONS</td>
<td>-100.00.0</td>
</tr>
<tr>
<td>A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.</td>
<td></td>
</tr>
<tr>
<td>$LESS_THAN_DURATIONS_BASE_FIRST</td>
<td>-131.073.0</td>
</tr>
<tr>
<td>A universal real literal that is less than DURATION'BASE'FIRST.</td>
<td></td>
</tr>
<tr>
<td>$MAX_DIGITS</td>
<td>9</td>
</tr>
<tr>
<td>Maximum digits supported for floating-point types.</td>
<td></td>
</tr>
<tr>
<td>$MAX_IN_LEN</td>
<td>200</td>
</tr>
<tr>
<td>Maximum input line length permitted by the implementation.</td>
<td></td>
</tr>
<tr>
<td>$MAX_INT</td>
<td>2147483647</td>
</tr>
<tr>
<td>A universal integer literal whose value is SYSTEM_MAX_INT.</td>
<td></td>
</tr>
<tr>
<td>$MAX_INT_PLUS_1</td>
<td>2147483648</td>
</tr>
<tr>
<td>A universal integer literal whose value is SYSTEM_MAX_INT+1.</td>
<td></td>
</tr>
<tr>
<td>Name and Meaning</td>
<td>Value</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>$MAX_LEN_INT_BASED_LITERAL</code></td>
<td>A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.</td>
</tr>
<tr>
<td><code>$MAX_LEN_REAL_BASED_LITERAL</code></td>
<td>A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.</td>
</tr>
<tr>
<td><code>$MAX_STRING_LITERAL</code></td>
<td>A string literal of size MAX_IN_LEN, including the quote characters.</td>
</tr>
<tr>
<td><code>$MIN_INT</code></td>
<td>A universal integer literal whose value is SYSTEM.MIN_INT.</td>
</tr>
<tr>
<td><code>$NAME</code></td>
<td>A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.</td>
</tr>
<tr>
<td><code>$NEG_BASED_INT</code></td>
<td>A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.</td>
</tr>
</tbody>
</table>

Value

(1..2 => "2:", 3..197 => '0', 198..200 => "11:")

(1..3 => "16:", 4..196 => '0', 197..200 => "F.E:"

(1 => 'n', 2..199 => 'A', 200 => 'n')

-2147483648

SHORT_SHORT_INTEGER (NOT supported)

16#FFFFFFFE#
APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 24 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.

- 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.
WITHDRAWN TESTS

- 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.

- 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, 76 & 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.