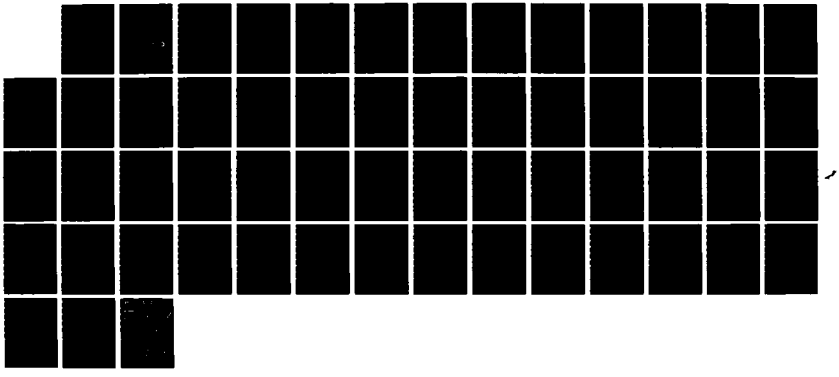
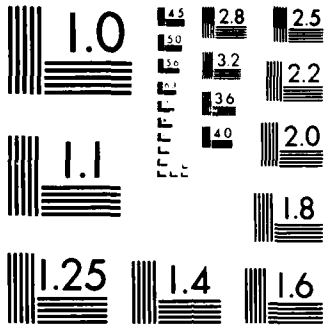
AD-A175 185 ADA COMPILER VALIDATION SUMMARY REPORT: ALSYS ALSYCOMP 003 VERSION 111 ZENITH 200(U) BUREAU D'ORIENTATION DE LA NORMALISATION EN INFORMATIQUE LE C.. 24 APR 86 1/1

UNCLASSIFIED ADA-86.4 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

(2)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

AD-A175 185

12. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
Ada Compiler Validation Summary Report:
Alsys AlsyCOMP_003 Version 1.1.1
ZENITH 200

5. TYPE OF REPORT & PERIOD COVERED
April 1986 to April 1987

6. PERFORMING ORG. REPORT NUMBER
Ada 86.4

7. AUTHOR(s)
BNI
Domaine de Voluceau ROCQUENCOURT
B.P. 105 - 78153 LE CHESNAY CEDEX, FRANCE

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS
BNI

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Ada Joint Programming Office
United States Department of Defense
Washington, D.C. 20301-3081

12. REPORT DATE
24 April 1986

13. NUMBER OF PAGES
54

14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)*
BNI
Domaine de Voluceau ROCQUENCOURT
B.P. 105-78153 LE CHESNAY CEDEX, FRANCE

15. SECURITY CLASS *(of this report)*
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)*

UNCLASSIFIED

DTIC
ELECTE
DEC 1 2 1986
D
E

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)*

See Attached.

DTIC FILE COPY

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)*

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Ada COMPILER VALIDATION SUMMARY REPORT:

Alsys
AlsyCOMP_003, version 1.1.1
ZENITH 200

Completion of On-Site Validation:
24 April 1986

Prepared By:
BNI
Domaine de Voluceau ROCQUENCOURT
B.P.105 - 78153 LE CHESNAY CEDEX
FRANCE

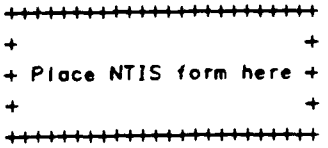
Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

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

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

86 4849

86 12 11 081


Place NTIS form here

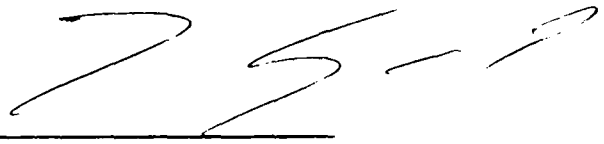
Ada Compiler Validation Summary Report:

Compiler name : AlsyCOMP_003, version 1.1.1

Host Computer	Target Computer
ZENITH 200	ZENITH 200
under	under
MS/DOS Version 3.1	MS/DOS Version 3.1

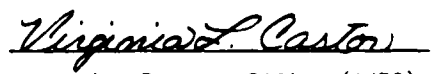
Testing Completed 24 April 1986 Using ACVC 1.7

This report has been reviewed and approved:



Ada Validation Facility (AVF)
BNI
Nicolas Malagardis represented by Jacqueline Sidi
Domaine de Voluceau ROCQUENCOURT
B.P.105 - 78153 LE CHESNAY CEDEX
FRANCE

Ada Validation Office (AVO)
John F. Kramer, Jr
Institute for Defense Analyses
Alexandria, VA



Ada Joint Program Office (AJPO)
Virginia L. Castor
Director
Washington, D.C.

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

EXECUTIVE SUMMARY

This Validation Summary Report presents the results and conclusions of testing performed on the AlsyCOMP_003, version 1.1.1. Standardized tests serve as input to an Ada compiler, producing results which are evaluated by the validation team. This summary briefly states the highlights of the AlsyCOMP_003, version 1.1.1 validation.

On-site testing was completed by 24 April 1986 at Alsys at La Celle Saint-Cloud, France under the auspices of the BNI (AVF), according to Ada Validation Office policies and procedures. The AlsyCOMP_003, version 1.1.1 is hosted on ZENITH 200 operating under MS/DOS Version 3.1. The suite of tests known as the Ada Compiler Validation Capability (ACVC), Version 1.7, was used. The ACVC is used to validate conformance of a compiler to ANSI/MIL-STD-1815A Ada. The purpose of testing is to ensure that a 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, or during execution.

The results of validation are summarized in the following table.

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	820	1014	12	9	21	1944
Failed	0	0	0	0	0	0	0
Inapplicable	0	4	306	5	2	2	319
Anomalous	0	0	0	0	0	0	0
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

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

Tests found to contain errors were withdrawn from Version 1.7 of the Ada Compiler Validation Capability (ACVC). When validation was completed, the following tests had been withdrawn:

C35904A	C41404A	C48008A
C4A014A	B4A010C	B83A06B
C92005A	C940ACA	CA1003B
BA2001E	CA3005A..D (4 tests)	BC3204C
CE2107E		

Some tests demonstrate that language features are not supported by an implementation. For this implementation the tests determined the following.

. SHORT_FLOAT is not supported:

B86001CP C34001F C35702A

. LONG_FLOAT is not supported:

B86001CQ C34001G C35702B

. Representation clauses for noncontiguous enumeration representations are not supported:

C55B16

. No other integer type other than INTEGER, SHORT_INTEGER, AND LONG_INTEGER is supported:

B86001DT

. The package SYSTEM is used by package TEXT_IO:

C86001F

. The 'SIZE clause is not supported:

C87B62A

. The 'STORAGE_SIZE clause is not supported:

C87B62B

. The 'SMALL clause is not supported:

C87B62C

. Generic package bodies can be compiled in separate compilation files, but before any corresponding generic instantiation:

CA2009C BC3205D

. Generic subprogram bodies can be compiled in, but before any corresponding generic instantiation:

CA2009F

. Pragma INLINE is not supported for procedures:

LA3004A EA3004C CA3004E

. Pragma INLINE is not supported for functions:

LA3004B EA3004D CA3004F

. No more than one internal file can be associated with the same external file, if one of the internal files is used for writing :

CE2107B CE2107C CE2107D
CE2111D CE3111B CE3111C
CE3114B CE3111D CE3111E

. An external file associated with more than one internal file cannot be reset for writing:

CE2111H CE3115A

. An external file associated with more than one internal file cannot be deleted:

CE2110B

. The compiler's capacity with respect to levels of loop nesting is at least 17 levels, but less than 31:

D55A03E..H (4 tests)

. The compiler's capacity with respect to the levels of block nesting is less than 65:

D56001B

. The library tasks were aborted when the main program terminated.

C94004A..C (3 tests)

ACVC Version 1.7 was taken on-site via magnetic tape to Alsys at Lo Celle Saint-Cloud, France. The tape was loaded, and all tests, except the withdrawn tests and any executable tests which make use of a floating point precision greater than SYSTEM.MAX_DIGITS, were compiled on ZENITH 200. Class A, C, D, and E tests were executed on ZENITH 200.

On completion of testing, all results were analyzed for failed Class A, C, D, or E programs, and all Class B and L compilation results were individually analyzed.

The ACVC, Version 1.7, contains 2279 tests of which 1944 were applicable to AlsyCOMP_003, version 1.1.1. No anomalies were found in the testing of this compiler. Testing demonstrated that all applicable tests were passed by this compiler. The AVF concluded that the results show acceptable compliance to ANSI/MIL-STD-1815A Ada.

1- INTRODUCTION
1.1- PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-1
1.2- USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3- REFERENCES 1-3
1.4- DEFINITION OF TERMS 1-3
1.5- CONFIGURATION 1-5

2- TEST RESULTS
2.1- ACVC Test Classes 2-1
2.1.1- Class A Tests 2-3
2.1.2- Class B Tests 2-4
2.1.3- Class C Tests 2-5
2.1.4- Class D Tests 2-6
2.1.5- Class E Tests 2-7
2.1.6- Class L Tests 2-8
2.1.7- Support Units 2-9
2.2- WITHDRAWN TESTS 2-9
2.3- INAPPLICABLE TESTS 2-11
2.4- IMPLEMENTATION CHARACTERISTICS 2-13

3- COMPILER ANOMALIES AND NONCONFORMANCES
3.1- ANOMALIES 3-1
3.2- NONCONFORMANCES 3-1

4- ADDITIONAL TESTING INFORMATION
4.1- PRE-VALIDATION 4-1
4.2- TEST SITE 4-1
4.3- TEST TAPE INFORMATION 4-1
4.4- TESTING LOGISTICS 4-2

5- SUMMARY AND CONCLUSIONS

Appendix A - COMPLIANCE STATEMENT

Appendix B - TEST PARAMETERS

Appendix C - COMMAND SCRIPTS

Appendix D - TEST NAMING

CHAPTER 1

INTRODUCTION

The Validation Summary Report describes how an Ada compiler conforms to the language standard. This report explains all technical terms used within and thoroughly reports the Ada Compiler Validation Capability (ACVC) test results. Ada compilers must be written according to the language specification as given in the ANSI/MIL-STD-1815A Ada. All implementation-defined features must be included for the compiler to conform to the Standard. Following the guidelines of the Standard ensures continuity between compilers. That is, the entire Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Standard, it must be understood that some differences do exist between implementations. ANSI/MIL-STD-1815A permits some implementation dependencies, e.g., the maximum length of identifiers, the maximum values of integer types, etc. These implementation-dependent features limit the portability of programs between compilers. Other differences between compilers are due to limitations imposed on a compiler by the operating system and by the hardware. All of these dependencies are given in the report.

Validation summary reports are written according to a standardized format. Compiler users can, therefore, more easily compare the reports from several compilers when selecting a compiler for a given task. The validation report can be completed mostly from the test results produced during validation testing. Additional testing information is given at the end of the report and states problems and details which are unique for a specific compiler. The format of the validation report limits variance between reports, enhances readability of the report, and accelerates report readiness.

1.1- PURPOSE OF THIS VALIDATION SUMMARY REPORT

The Validation Summary Report documents the results of the testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To identify any language constructs supported by the translator that do not conform to the Ada Standard

. To identify any unsupported language constructs required by the Ada Standard

. To describe the implementation-dependent behavior allowed by the Ada Standard

Testing of this compiler was conducted under the supervision of BNI according to policies and procedures established by the Ada Validation Office (AVO). Testing was completed by 24 April 1986 at Alsys at La Celle Saint-Cloud, France.

1.2- USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Validation Office 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 any statement or statements set forth in this report are accurate or complete, or that the subject compiler has no nonconformances to the Ada Standard other than those presented. This report is not intended for the purpose of publicizing the findings summarized herein.

Questions regarding this report or the validation tests should be directed to:

Ada Validation Office
Institute for Defense Analyses
1801 N. Beauregard
Alexandria VA 22311

and to:

BNI
Domaine de Voluceau ROCQUENCOURT
B.P.105 - 78153 LE CHESNAY CEDEX
FRANCE

1.3- REFERENCES

. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, Feb 1983.

. Ada Validation Organization : Policies and Procedures, T.H. Probert, June 1982, The MITRE Corporation MTR-82W00103.

. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., Dec 1984.

1.4- DEFINITION OF TERMS

Anomaly	A test result that, given pre-validation analysis, is not expected during formal validation but is judged allowable under the circumstances.
ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The BNI. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Office. In the context of this report, the AVO is responsible for setting policies and procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations.
Withdrawn test	A test that has an invalid test objective, fails to meet its test objective, or contains illegal use of the language.

1.5- CONFIGURATION

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

Compiler: AlsyCOMP_003, version 1.1.1

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

Machine(s):	ZENITH 200
Operating System:	MS/DOS Version 3.1
Memory Size:	augmented to 4 Megabytes

Target Computer:

Machine(s):	ZENITH 200
Operating System:	MS/DOS Version 3.1
Memory Size:	augmented to 4 Megabytes

One ZENITH 200 with the above configuration was used to process the ACVC tests.

CHAPTER 2

TEST RESULTS

2.1- ACVC Test Classes

Conformance to ANSI/MIL-STD-1815A is measured using the Ada Compiler Validation Capability (ACVC). The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. Legal programs are compiled and executed while illegal programs are just compiled. Support packages are used to report the results of the legal programs. A compiler must correctly process each of the tests in the suite and demonstrate conformance 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. Tests that are found to contain errors are withdrawn from the ACVC. Detailed test results are listed in the Appendix D. The results of validation testing are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	820	1014	12	9	21	1944
Failed	0	0	0	0	0	0	0
Inapplicable	0	4	306	5	2	2	319
Anomalous	0	0	0	0	0	0	0
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

A total of 1985 tests were processed during this validation attempt. The 16 withdrawn tests in Version 1.7 were not processed, nor were 278 Class C tests that were inapplicable because they use floating point types having digits that exceed the maximum value for the implementation. All other tests were processed.

Some conventions are followed in the ACVC to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic 55 character set, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be

supported in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values. The values used for this validation are listed in Appendix B.

2.1.1- Class A Tests

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a message indicating that it has passed. If a Class A test cannot be compiled and executed because of its size, then the test is split into a set of smaller subtests that can be processed. Splits were required for 2 tests:

AE2101A AE2101F

The following table shows that all applicable Class A tests were passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	15	9	0	5	2	12	13	3	0	0	0	9	68
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	0	0	0	0	0	0	0
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	15	9	0	5	2	12	13	3	0	0	0	9	68

2.1.2- Class B Tests

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 manually 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. If one or more errors are not detected, then a version of the test is created that contains only the undetected errors. The resulting "split" is compiled and examined. The splitting process continues until all errors are detected by the compiler. Splits were required for 12 tests:

B32202A	B33006A	B37004A
B43201D	B61012A	B62001B
B91004A	BA1101B	BC3009A
BC3009C	BC3204D	BC3205E

The following table shows that all applicable Class B tests were passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	39	86	86	113	73	67	48	87	36	8	159	18	820
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	3	0	0	0	1	0	4
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	1	0	0	0	1	0	1	0	1	0	4
TOTAL	39	86	87	113	73	67	52	87	37	8	161	18	828

2.1.3- Class C Tests

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASS/FAIL message indicating the result when it is executed. If a Class C test cannot be compiled because it exceeds the compiler's capacity, then the test is split into smaller subtests until all are compiled and executed. Splits were required for 6 tests:

C23003G..J (4 tests) C23006E C23006G

The following table shows that all applicable Class C tests were passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	37	90	162	118	82	18	93	106	40	20	56	192	1014
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	119	140	1	0	0	4	3	4	0	0	12	306
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	1	3	0	0	0	0	2	5	0	0	1	12
TOTAL	60	210	305	119	82	18	97	111	49	20	56	205	1332

2.1.4- Class D Tests

Class D tests check the compilation and execution capacities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for the number of identifiers permitted in a compilation, the number of units in a library, the number of nested loops in a subprogram body, and so on, a compiler may refuse to compile a Class D test. Each Class D test is self-checking and produces a PASS/FAIL message indicating the result when it is executed. If a Class D test fails to compile because the capacity of the compiler is exceeded, then the test is classified as inapplicable.

The following table shows that all applicable Class D tests were passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	1	0	4	4	3	0	0	0	0	0	0	0	12
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	5	0	0	0	0	0	0	0	0	5
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	1	0	4	9	3	0	0	0	0	0	0	0	17

Capacities measured by the Class D tests are detailed in section 2.4, IMPLEMENTATION CHARACTERISTICS.

2.1.5- Class E Tests

Class E tests provide information about the compiler in those areas in which the Ada Standard permits implementations to differ. Each Class E test is executable and produces messages that indicate how the Ada Standard is interpreted. However, in some cases the Ada Standard permits a compiler to detect a condition either at compile time or at execution time, and thus a Class E test may correctly fail to execute. A Class E test is passed if it fails to compile and appropriate error messages are issued, or if it executes properly and produces a message that it has passed. If a Class E test cannot be compiled and executed because of its size, then the test is split into a set of smaller subtests that can be processed. No splits were required .

The following table shows that all applicable Class E tests were passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	1	3	2	1	1	0	0	0	0	0	0	1	9
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	0	0	2	0	0	0	2
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	1	3	2	1	1	0	0	0	2	0	0	1	11

Information obtained from the Class E tests is detailed in section 2.4, IMPLEMENTATION CHARACTERISTICS.

2.1.6- Class L Tests

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 and the test does not execute.

The following table shows that all applicable Class L tests were passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	0	0	0	0	0	0	0	0	21	0	0	0	21
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	0	0	2	0	0	0	2
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	0	0	0	0	0	0	0	0	23	0	0	0	23

2.1.7- Support Units

Three packages support the self-checking features of Class C tests: REPORT, CHECK_FILE, and VAR_STRINGS. The REPORT package provides the mechanism by which executable tests report results. It also provides a set of identity functions that are used to defeat some compiler optimization strategies to cause computations to be made by the target computer instead of the by the compiler on the host computer. The CHECK_FILE package 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 VAR_STRINGS package defines types and subprograms for manipulating varying-length character strings. The operation of these three packages is checked by a set of executable tests. These tests produce messages that are examined manually to verify that the packages are operating correctly. If these packages are not operating correctly, then validation is not attempted.

An applicant is permitted to substitute the body of package REPORT with an equivalent one if for some reason the original version provided by the ACVC cannot be executed on the target computer. Package REPORT was not modified for this validation.

All support package specifications and bodies were compiled and were demonstrated to be operating correctly.

2.2- WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. When testing was performed, the following 16 tests had been withdrawn for the reasons indicated:

C35904A:
The elaboration of subtype declarations SFX3 & SFX4 may raise NUMERIC_ERROR vs. CONSTRAINT_ERROR.

C41404A: The values of 'LAST and 'LENGTH in the "if" statements from line 74 to the end of the test are incorrect.

C48008A:
This test requires that the evaluation of default initial values not occur if an exception is raised by an allocator. However, the LMC has ruled that such a requirement is incorrect (AI-00397).

B4A010C:

The object_declaration in line 18 follows a subprogram body of the same declarative part.

C4A014A:

The number declarations in lines 19-22 are not correct, because conversions are not static.

B83A06B:

The Ada Standards 8.3(17) and AI_00330 permit the label LAB_ENUMERAL of line 80 to be considered a homograph of the enumeration literal in line 25.

C92005A:

At line 40, "/=" for type PACK.BIG_INT is not visible without a "use" clause for package PACK.

C940ACA:

This test assumes that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program; however, such an execution order is not required by the Ada Standard, so the test is erroneous.

CA1003B:

This test requires all of the legal compilation units of a file containing some illegal units to be compiled and executed. But according to AI-00255, such a file may be rejected as a whole.

BA2001E:

The Ada Standards 10.2(5) states that "simple names of all subunits that have the same ancestor library unit must be distinct identifiers." This test checks for the above condition when stubs are declared, but it is not clear that the check must be made then, as opposed to when the subunit is compiled.

CA3005A..D (4 tests)

There exists no valid elaboration order for these tests.

BC3204C

The file BC3204C4 should contain the body for BC3204C0—as indicated in line 25 of BC3204C3M.

CE2107E

TEMP_HAS_NAME must be given an initial value of TRUE.

2.3- INAPPLICABLE TESTS

Some tests use features of the Ada language that the Ada Standard does not require a compiler to support; thus these tests may be inapplicable to a particular compiler. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 319 tests were inapplicable for the reasons indicated:

B86001DT (1 test)
This test is inapplicable because this implementation has no predefined type other than INTEGER, FLOAT, SHORT_INTEGER, SHORT_FLOAT, LONG_INTEGER, LONG_FLOAT and DURATION.

C24113C..Y
C35705C..Y
C35706C..Y
C35707C..Y
C35708C..Y
C35802C..Y
C45241C..Y
C45321C..Y
C45421C..Y
C45424C..Y
C45521C..Z
C45621C..Z (10*23 + 2*24 = 278 tests)
These tests are inapplicable because this implementation limits digits to 6.

B86001CP
C34001F
C35702A (1*3 = 3 tests)
These tests are inapplicable because this implementation does not support SHORT_FLOAT.

B86001CQ
C34001G
C35702B (1*3 = 3 tests)
These tests are inapplicable because this implementation does not support LONG_FLOAT.

C55B16A
C87B62A..C (1+3 = 4 tests)
These tests are inapplicable because this implementation does not support representation clauses.

C86001F (1 test)
This test is inapplicable because package SYSTEM is used by TEXT_IO.

BC3205D
CA2009C
CA2009F (1•3 = 3 tests)
These tests are inapplicable because this implementation does not support instantiating missing generic bodies.

CA3004E..F
EA3004C..D
LA3004A..B (3•2 = 6 tests)
These tests are inapplicable because this implementation does not support pragma INLINE. These tests ignore the pragma and are processed correctly.

CE2107B..D
CE2110B
CE2111D
CE2111H
CE3111B..E
CE3114B
CE3115A (3+1+1+1+4+1+1 = 12 tests)
These tests are inapplicable because this implementation does not support the sharing of external file by several internal files when one of the external file is opened for writing.

D55A03E..H (4 tests)
These tests are inapplicable because the compiler's capacity with respect to levels of loop nesting is at least 17 levels, but less than 31.

D56001B
This test is inapplicable because the compiler's capacity with respect to the levels of block nesting is less than 65.

C94004A..C (3 tests)
These tests are inapplicable because the library tasks were aborted when the main program terminated.

2.4- IMPLEMENTATION CHARACTERISTICS

One of the purposes of validation 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, inapplicable tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

. Non-graphic characters.

Non-graphic characters are defined in the ASCII character set but are not permitted in the texts of Ada programs. The compiler correctly recognizes these characters as illegal in Ada compilations. The characters are printed in the output listing.

. Capacities.

The compiler correctly processes compilations containing loop statements nested to at least 17 levels (but less than 31), procedures nested to at least 17 levels (but less than 31), and 723 variables.

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly.

. Predefined types.

This implementation supports the predefined types SHORT_INTEGER, INTEGER, LONG_INTEGER, FLOAT and DURATION. It does not support any other predefined numeric types.

. Based literals.

An implementation is allowed to reject a based literal with value exceeding SYSTEM.MAX_INT during compilation or it may raise NUMERIC_ERROR during execution. This compiler raises NUMERIC_ERROR during execution.

. Array types.

An implementation is allowed to raise NUMERIC_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. When an array type is declared with an index range exceeding INTEGER values and with a component that is a null BOOLEAN array, this compiler does not raise any exception.

When an array type is declared with an index range exceeding SYSTEM.MAX_INT values and with a component that is a null BOOLEAN array, this compiler raises NUMERIC_ERROR.

A packed BOOLEAN array of length INTEGER_LAST+3 does not raise any exception. A packed two-dimensional BOOLEAN array with INTEGER_LAST+3 components does not raise any exception.

NOTE : this compiler does not support pragma PACK.

A null array with one dimension of length exceeding INTEGER'LAST does not raises NUMERIC_ERROR.

In assigning one-dimensional array types, the entire expression is evaluated 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 entire expression is NOT evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype.

. Discriminated types.

In assigning record types with discriminants, the entire expression is evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype.

An incompletely declared type with discriminants may be used in an access type definition and constrained either there or in later subtype indications.

. Aggregates.

When evaluating the choices of a multi-dimensional aggregate the order in which choices are evaluated and index subtype checks are made depends upon the aggregate itself.

When evaluating an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds.

. Functions.

The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is rejected by the implementation.

. Representation clauses.

'SMALL length clauses are not supported.

Enumeration representation clauses are not supported.

. Tasks.

A task object's storage size is not allowed to change after the task is activated.

. Generics.

When given a separately compiled generic declaration, some illegal instantiations, and a body, the compiler ignores the body because it is not in the same compilation as its declaration and it is compiled after the instantiations. It issues a warning for each instantiation, stating that a null body is assumed.

. Package CALENDAR.

TIME_OF and SPLIT are inverses when SECONDS is a non-model number.

. Pragmas.

Pragma INLINE is not supported for procedures. It is not supported for functions.

. Input/output.

Package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. Package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. However any call to OPEN or CREATE of such instances will raise an exception.

More than one internal file can be associated with each external file for sequential I/O for reading only. An external file associated with more than one internal file cannot be deleted.

More than one internal file can be associated with each external file for direct I/O for reading only. An external file associated with more than one internal file cannot be deleted.

More than one internal file can be associated with each external file for text I/O for reading only. An external file associated with more than one internal file cannot be deleted.

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.

Dynamic creation and resetting of a sequential file is allowed.

Temporary sequential files are given a name. Temporary direct files are given a name. Temporary files given names are not deleted when they are closed.

CHAPTER 3

COMPILER ANOMALIES AND NONCONFORMANCES

3.1- ANOMALIES

An anomaly is a test result that, given the pre-validation analysis, was not expected during formal validation but which is judged allowable by the AVF and the AVO under the circumstances of the validation. No anomalies were detected in this validation attempt.

3.2- NONCONFORMANCES

Any discrepancy between expected test results and actual test results is considered to be a nonconformance. No nonconformances were detected in this validation attempt.

CHAPTER 4

ADDITIONAL TESTING INFORMATION

4.1- PRE-VALIDATION

Prior to validation, a set of test results for ACVC 1.7 produced by AlsyCOMP_003, version 1.1.1 was submitted to BNI by the applicant for pre-validation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests.

4.2- TEST SITE

Tests were compiled and executed at Alsys at La Celle Saint-Cloud, France.

4.3- TEST TAPE INFORMATION

A test tape containing ACVC Version 1.7 was taken on-site by the validation team. This tape contained all tests applicable to this validation as well as all tests inapplicable to this validation except for any Class C tests that require floating-point precision exceeding the maximum value supported by the implementation. Tests that were withdrawn from ACVC 1.7 were not written to the tape. Tests that make use of values that are specific to an implementation were customized before being written to the tape. Any split tests were also included on the test tape so that no editing of these test files was necessary when the validation team arrived on-site.

The format of the test tape was the same as the ACVC distribution tapes. The files were mounted on a VAX. They were transfered from the VAX by an ETHERNET local area network to four IBM PC/ATs and subsequently transfered on floppy disks.

4.4- TESTING LOGISTICS

Processing of the tests was begun using command scripts provided by Alsys. The text of these scripts are given in Appendix C.

The compiler supports various options that control its operation. The compiler was tested with the following option settings.

For tests from class C the following was used :

```
Alsys ADA Library Manager  Version 1.00  (c)Copyright Alsys 1986
NEW (LIBRARY  => ,
     OPTIONS  => (OVERWRITE     => NO,
                  TARGET_KIND   => I286_REAL,
                  TASKING       => YES));

COMPILE (SOURCE   => ,
         LIBRARY  => "adalib",
         DISPLAY  => (LIST_FILE          => NO,
                      RECAP              => NO,
                      WARNING            => NO,
                      BANNER             => NO,
                      TEXT               => NO,
                      DETAIL             => NO,
                      ASSEMBLY           => NO),
         FORMAT   => (LINE_LENGTH        => 79,
                      PAGE_LENGTH        => 45),
         OPTIONS  => (ERRORS             => 999,
                      LEVEL              => CODE,
                      CHECKS             => YES,
                      STACK_CHECK        => YES,
                      GENERIC_STUBS      => NO));

BIND (PROGRAM      => ,
      LIBRARY      => "adalib",
      DISPLAY      => (BIND_MAP          => NO,
                       LINK_MAP          => NO,
                       WARNING           => YES,
                       UNITS             => NO,
                       ELABORATION       => NO),
      OPTIONS      => (LEVEL             => LINK,
                       EXECUTION_MODE    => LIBRARY_DEFAULT,
                       OUTPUT_NAMES      => no_value,
                       MAIN_STACK        => 64,
                       TASK_STACK        => 8,
                       INITIAL_HEAP      => 64,
                       HEAP_INCREMENT    => 64,
                       STACK_TRACE       => YES,
                       FAST_TIMER        => NO,
                       RUNTIME_OPTIONS   => NO),
      INTERFACED   => (OBJECT_MODULES    => no_value,
                       SEARCH_LIBRARIES  => no_value));

Alsys PC AT Ada Version 1.00
(C)Copyright Alsys 1986.  All rights reserved.
```

For tests from classes A, B, D, E and L, the following was used :

```
Alsys ADA Library Manager  Version 1.00  (c)Copyright Alsys 1986
NEW (LIBRARY  => ,
     OPTIONS  => (OVERWRITE    => NO,
                  TARGET_KIND  => I286_REAL,
                  TASKING      => YES));

COMPILE (SOURCE   => ,
         LIBRARY  => "\acvc\adalib",
         DISPLAY  => (LIST_FILE          => NO,
                      RECAP              => NO,
                      WARNING            => YES,
                      BANNER             => YES,
                      TEXT               => YES,
                      DETAIL             => YES,
                      ASSEMBLY           => NO),
         FORMAT   => (LINE_LENGTH        => 79,
                      PAGE_LENGTH        => 45),
         OPTIONS  => (ERRORS             => 999,
                      LEVEL              => CODE,
                      CHECKS             => YES,
                      STACK_CHECK        => YES,
                      GENERIC_STUBS      => NO));

-- BIND_MAP=YES for L TEST ONLY

BIND (PROGRAM     => ,
      LIBRARY     => "\acvc\adalib",
      DISPLAY     => (BIND_MAP          => NO,
                      LINK_MAP          => NO,
                      WARNING           => YES,
                      UNITS             => NO,
                      ELABORATION       => NO),
      OPTIONS     => (LEVEL             => LINK,
                      EXECUTION_MODE    => LIBRARY_DEFAULT,
                      OUTPUT_NAMES      => no_value,
                      MAIN_STACK        => 64,
                      TASK_STACK        => 8,
                      INITIAL_HEAP      => 64,
                      HEAP_INCREMENT    => 64,
                      STACK_TRACE       => YES,
                      FAST_TIMER        => NO,
                      RUNTIME_OPTIONS   => NO),
      INTERFACED  => (OBJECT_MODULES    => no_value,
                      SEARCH_LIBRARIES  => no_value));

Alsys PC AT Ada Version 1.00
(C)Copyright Alsys 1986.  All rights reserved.
```

The procedure used for the validation of the ZENITH 200 was done on one machine. An overview of this procedure follows :

All the ACVC sources were copied on floppy diskettes, each floppy containing one chapter. This set of floppies was then used as a master ACVC set.

The entire directory structure (with adaworld scripts in appropriate directories) was then created.

All these preliminary tasks being done, the procedure for validating an ACVC chapter was then :

- copy the master diskette containing the ACVC sources for the chapter into the corresponding subdirectory on the machine

- invoke the adaworld script for this chapter

- when the execution of the script was finished (and then the execution of the ACVC chapter), copy the contents of the subdirectories lst (for compilation listings) and res (for execution result) on a transfer diskette, and send back the contents of that diskette on the VAX on which all results where accumulated.
  The task of uploading from diskettes to the VAX was done using four IBM PC/ATs connected to the VAX via an ETHERNET local area network.

- finally the ACVC sources and all results for the chapter were deleted from the machine before starting the next chapter, so as not to run out of disk space.

This procedure was repeated for every ACVC chapter.

CHAPTER 5

SUMMARY AND CONCLUSIONS

The BNI identified 1985 of the 2279 tests in ACVC Version 1.7 to be processed during the validation of AlsyCOMP_003, version 1.1.1. Excluded were 278 tests requiring too great a floating-point precision, and the 16 withdrawn tests. 41 tests were determined to be inapplicable after they were processed. The remaining 1944 tests were passed by the compiler.

The BNI concludes that these results demonstrate acceptable conformance to the Ada Standard.

APPENDIX A

COMPLIANCE STATEMENT

The only allowed implementation dependencies correspond to implementation-dependent pragmas and attributes, 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 AlsyCOMP_003, version 1.1.1 are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).

(1) Implementation-Dependent Pragmas

```
Pragma INTERFACE (language_name, subprogram_name);
Pragma INTERFACE_NAME (subprogram_name, string_literal);
```

(2) Implementation_Dependent Attributes

None.

(3) Package SYSTEM

The specification for package SYSTEM is

```
package SYSTEM is

    type ADDRESS is access STRING ;
    type NAME is (I_80x86) ;

    SYSTEM_NAME   : constant NAME := I_80x86 ;
    STORAGE_UNIT  : constant := 8 ;
    MEMORY_SIZE   : constant := 640 * 1024 ;

    --  System-Dependent Named Numbers:

    MIN_INT       : constant := -(2**31) ;
    MAX_INT       : constant := 2**31 - 1 ;
    MAX_DIGITS    : constant := 6 ;
    MAX_MANTISSA  : constant := 31 ;
    FINE_DELTA    : constant := 2#1.0#E-31 ;
    TICK          : constant := 1.0 / 18.2 ;

    --  Other System-Dependent Declarations

    subype PRIORITY is INTEGER range 1..10 ;

      ...
end SYSTEM;
```

(4) Representation Clause Restrictions

Representation clauses specify how the types of the language are to be mapped onto the underlying machine. The following are restrictions on representation clauses.

Address Clause

Not supported.

Length Clause

Not supported.

Enumeration Representation Clause

Not supported.

Record Representation Clause

Not supported.

(5) Conventions

The following conventions are used for an implementation-generated name denoting implementation-dependent components.

There are no implementation-generated names.

(6) Address Clauses

Adress clauses are not supported.

(7) Unchecked Conversions

The following are restrictions on unchecked conversions, including those depending on the respective sizes of objects of the source and target.

Unchecked conversions are allowed between any types which are implemented on the same physical size.

(8) Input-Output Packages

The following are implementation-dependent characteristics of the input-output packages.

SEQUENTIAL_IO Package

SEQUENTIAL_IO is defined as specified in the Standard. However SEQUENTIAL_IO is not supported for unconstrained types. The instantiation is accepted, but any call to OPEN or CREATE will raise USE_ERROR.

DIRECT_IO Package

DIRECT_IO is defined as specified in the Standard with COUNT defined as follows :

```
type COUNT is range 0 .. 2_147_483_647 ;
```

However DIRECT_IO is not supported for unconstrained types. The instantiation is accepted, but any call to OPEN or CREATE will raise USE_ERROR.

TEXT_IO Package

```
type COUNT is range 0 .. 2_147_483_647 ;

subtype FIELD is INTEGER range 0 .. 255 ;
```

LOW_LEVEL_IO

Not supported.

(9) Package STANDARD

```
type INTEGER is range -32768..32767 ;
type SHORT_INTEGER is range -128..127 ;
type LONG_INTEGER is
 range -2_147_483_648..2_147_483_647 ;

-- no other predefined integer types

type FLOAT is digits 6 range
        -2#1.111_1111_1111_1111_1111_1111#E+127
      ..  2#1.111_1111_1111_1111_1111_1111#E+127 ;
-- type SHORT_FLOAT is not implemented ;
-- type LONG_FLOAT is not implemented ;

-- no other predefined floating point types

type DURATION is delta
0.001 range -86_400.0 ..  86_400.0;
```

— no predefined types other than those required by the Standard.

(10) File Names

File names make no use of conventions except those of the operating system.

APPENDIX B

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 identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$MAX_IN_LEN Maximum input line length permitted by the implementation	255
$BIG_ID1 Identifier of size MAX_IN_LEN with varying last character.	X2345678901234567890123456789012345678901234567890123456789012345AAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAA1
$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	X2345678901234567890123456789012345678901234567890123456789012345AAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAA2

Name and Meaning	Value

$BIG_ID3
Identifier of size MAX_IN_LEN with varying middle character.

```
X2345678901234567890123456789012345678901234567890123456789012345AAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA3AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
```

$BIG_ID4
Identifier of size MAX_IN_LEN with varying middle character.

```
X2345678901234567890123456789012345678901234567890123456789012345AAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA4AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
```

$NEG_BASED_INT
A based integer literal whose highest order non-zero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.

```
8#7777777777776#
```

$BIG_INT_LIT
An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long.

```
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000298
```

$BIG_REAL_LIT
A real literal that can be either of floating or fixed point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.

```
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000069.0E1
```

Name and Meaning	Value
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{|}~"
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)
$BLANKS Blanks of length MAX_IN_LEN - 20	
$MAX_DIGITS Maximum digits supported for floating point types.	6
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, LONG_INTEGER, or DURATION. AlsyCOMP_003 supports no other type, so an arbitrary name was used.	LONG_LONG_INTEGER
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-32768

Name and Meaning	Value
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	32767
$MAX_INT The universal integer expression whose value is SYSTEM.MAX_INT	2147483647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST.	-33_554_433.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	33_554_434.0
$COUNT_LAST Value of COUNT'LAST in TEXT_IO package.	2147483647
$FIELD_LAST Value of FIELD'LAST in TEXT_IO package.	255

Name and Meaning	Value
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	X}]!@#$^&~Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	XYZ*
$ILLEGAL_EXTERNAL_FILE_NAME1 Illegal external file name.	BAD-CHARACTER**
$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file names.	MUCH-TOO-LONG-NAME-FOR-A-FILE

APPENDIX C

COMMAND SCRIPTS

File : DO_A.ADW

```
invoke acvc_env.adw, y

--
compile         a21001a.ada, list=lst\a21001a.lst
bind    a21001a
system.execute a21001a
--
compile         a22002a.ada, list=lst\a22002a.lst
bind    a22002a
system.execute a22002a
--
compile         a22006b.ada, list=lst\a22006b.lst
bind    a22006b
system.execute a22006b
--
compile         a26004a.exp, list=lst\a26004a.lst
bind    a26004a
system.execute a26004a
--
compile         a29002a.ada, list=lst\a29002a.lst
bind    a29002a
system.execute a29002a
--
compile         a29002b.ada, list=lst\a29002b.lst
bind    a29002b
system.execute a29002b
--
compile         a29002c.ada, list=lst\a29002c.lst
bind    a29002c
system.execute a29002c
--
compile         a29002d.ada, list=lst\a29002d.lst
bind    a29002d
system.execute a29002d
--
compile         a29002e.ada, list=lst\a29002e.lst
bind    a29002e
system.execute a29002e
--
compile         a29002f.ada, list=lst\a29002f.lst
bind    a29002f
system.execute a29002f
--
compile         a29002g.ada, list=lst\a29002g.lst
bind    a29002g
system.execute a29002g
--
compile         a29002h.ada, list=lst\a29002h.lst
bind    a29002h
system.execute a29002h
--
compile         a29002i.ada, list=lst\a29002i.lst
bind    a29002i
system.execute a29002i
--
compile         a29002j.ada, list=lst\a29002j.lst
bind    a29002j
system.execute a29002j
--
compile         a2a031a.ada, list=lst\a2a031a.lst
```

```
bind   a2a031a
system.execute a2a031a
--
compile        a32203b.ada, list=lst\a32203b.lst
bind   a32203b
system.execute a32203b
--
compile        a32203c.ada, list=lst\a32203c.lst
bind   a32203c
system.execute a32203c
--
compile        a32203d.ada, list=lst\a32203d.lst
bind   a32203d
system.execute a32203d
--
compile        a34008b.ada, list=lst\a34008b.lst
bind   a34008b
system.execute a34008b
--
compile        a38106d.ada, list=lst\a38106d.lst
bind   a38106d
system.execute a38106d
--
compile        a38106e.ada, list=lst\a38106e.lst
bind   a38106e
system.execute a38106e
--
compile        a38199a.ada, list=lst\a38199a.lst
bind   a38199a
system.execute a38199a
--
compile        a38199b.ada, list=lst\a38199b.lst
bind   a38199b
system.execute a38199b
--
compile        a38199c0.ada, list=lst\a38199c0.lst
compile        a38199c1.ada, list=lst\a38199c1.lst
compile        a38199c2.ada, list=lst\a38199c2.lst
bind   a38199c1m
system.execute a38199c1m
--
compile        a54b01a.ada, list=lst\a54b01a.lst
bind   a54b01a
system.execute a54b01a
--
compile        a54b02a.ada, list=lst\a54b02a.lst
bind   a54b02a
system.execute a54b02a
--
compile        a55b12a.ada, list=lst\a55b12a.lst
bind   a55b12a
system.execute a55b12a
--
compile        a55b13a.ada, list=lst\a55b13a.lst
bind   a55b13a
system.execute a55b13a
--
compile        a55b14a.ada, list=lst\a55b14a.lst
bind   a55b14a
system.execute a55b14a
```

```
--
compile         a62006d.ada, list=lst\a62006d.lst
bind    a62006d
system.execute a62006d
--
compile         a63202a.ada, list=lst\a63202a.lst
bind    a63202a
system.execute a63202a
--
compile         a71002a.ada, list=lst\a71002a.lst
bind    a71002a
system.execute a71002a
--
compile         a71004a.ada, list=lst\a71004a.lst
bind    a71004a
system.execute a71004a
--
compile         a72001a.ada, list=lst\a72001a.lst
bind    a72001a
system.execute a72001a
--
compile         a73001i.ada, list=lst\a73001i.lst
bind    a73001i
system.execute a73001i
--
compile         a73001j.ada, list=lst\a73001j.lst
bind    a73001j
system.execute a73001j
--
compile         a74006a.ada, list=lst\a74006a.lst
bind    a74006a
system.execute a74006a
--
compile         a74105b.ada, list=lst\a74105b.lst
bind    a74105b
system.execute a74105b
--
compile         a74106a.ada, list=lst\a74106a.lst
bind    a74106a
system.execute a74106a
--
compile         a74106b.ada, list=lst\a74106b.lst
bind    a74106b
system.execute a74106b
--
compile         a74106c.ada, list=lst\a74106c.lst
bind    a74106c
system.execute a74106c
--
compile         a74205e.ada, list=lst\a74205e.lst
bind    a74205e
system.execute a74205e
--
compile         a74205f.ada, list=lst\a74205f.lst
bind    a74205f
system.execute a74205f
--
compile         a83a02a.ada, list=lst\a83a02a.lst
bind    a83a02a
system.execute a83a02a
```

```
--
compile         a83a02b.ada, list=lst\a83a02b.lst
bind    a83a02b
system.execute a83a02b
--
compile         a83a06a.ada, list=lst\a83a06a.lst
bind    a83a06a
system.execute a83a06a
--
compile         a83c01c.ada, list=lst\a83c01c.lst
bind    a83c01c
system.execute a83c01c
--
compile         a83c01d.ada, list=lst\a83c01d.lst
bind    a83c01d
system.execute a83c01d
--
compile         a83c01e.ada, list=lst\a83c01e.lst
bind    a83c01e
system.execute a83c01e
--
compile         a83c01f.ada, list=lst\a83c01f.lst
bind    a83c01f
system.execute a83c01f
--
compile         a83c01g.ada, list=lst\a83c01g.lst
bind    a83c01g
system.execute a83c01g
--
compile         a83c01h.ada, list=lst\a83c01h.lst
bind    a83c01h
system.execute a83c01h
--
compile         a83c01i.ada, list=lst\a83c01i.lst
bind    a83c01i
system.execute a83c01i
--
compile a83c01j.ada, list=lst\a83c01j.lst
bind    a83c01j
system.execute  a83c01j
--
compile a85007d.ada, list=lst\a85007d.lst
bind    a85007d
system.execute  a85007d
--
compile a85013b.ada, list=lst\a85013b.lst
bind    a85013b
system.execute  a85013b
--
compile a91002m.ada, list=lst\a91002m.lst
bind    a91002m
system.execute  a91002m
--
compile a95005a.ada, list=lst\a95005a.lst
bind    a95005a
system.execute  a95005a
--
compile a97106a.ada, list=lst\a97106a.lst
bind    a97106a
system.execute  a97106a
```

```
--
compile ae2101a.ada, list=lst\ae2101a.lst
bind    ae2101a
system.execute  ae2101a
--
compile ae2101b.ada, list=lst\ae2101b.lst
bind    ae2101b
system.execute  ae2101b
--
compile ae2101c.dep, list=lst\ae2101c.lst
bind    ae2101c
system.execute  ae2101c
--
compile ae2101d.ada, list=lst\ae2101d.lst
bind    ae2101d
system.execute  ae2101d
--
compile ae2101f.ada, list=lst\ae2101f.lst
bind    ae2101f
system.execute  ae2101f
--
compile ae2101h.dep, list=lst\ae2101h.lst
bind    ae2101h
system.execute  ae2101h
--
compile ae2101s.ada, list=lst\ae2101s.lst
bind    ae2101s
system.execute  ae2101s
--
compile ae2101t.ada, list=lst\ae2101t.lst
bind    ae2101t
system.execute  ae2101t
--
compile ae2101u.ada, list=lst\ae2101u.lst
bind    ae2101u
system.execute  ae2101u
--
compile ae2101v.ada, list=lst\ae2101v.lst
bind    ae2101v
system.execute  ae2101v
--
compile ae3101a.ada, list=lst\ae3101a.lst
bind    ae3101a
system.execute  ae3101a
--
compile ae3702a.ada, list=lst\ae3702a.lst
bind    ae3702a
system.execute  ae3702a
--
compile ae3709a.ada, list=lst\ae3709a.lst
bind    ae3709a
system.execute  ae3709a
```

File : ACVC_ENV.ADW

```
default.system stay_resident=no

default.compile library     = \acvc\adalib,
                banner      = yes,
                text        = yes,
                line_length = 79,
                error       = 999

default.bind lib=\acvc\adalib

lib.new \acvc\adalib,task, overwrite
```

File : EXECUTE.BAT

```
echo on
%1 > res\%1.res
erase %1.obj
erase %1.exe
erase %1.lnk
```

APPENDIX D

TEST NAMING

Each test name indicates the class of the test and which test objective in the ACVC Implementers' Guide applies to the test.

Each test has a name that identifies the section of the Ada Standard addressed by the test objective. The name of a test is interpreted according to the table below, where the first column indicates the character position in the name and the second column, the meaning of that position:

POS	MEANING
1	Test class: A, B, C, D, E, L.
2	Implementers' Guide chapter number (in hexadecimal).
3	Implementers' Guide section number within a chapter (in Hexadecimal)
4	Implementers' Guide subsection number (in hexadecimal)
5–6	Implementers' Guide Test Objective number (in decimal)
7	Test sequence letter
8	[Optional] Compilation sequence digit or letter
9	[Optional] Main program designator in the case of a test having multiple compilation units.

Characters 8 and 9 are only present for tests that consist of several separately compiled units. A series of separately compiled units is counted as one test for reporting purposes. The eighth character indicates the order in which the units are to be compiled, with unit 0 being compiled first. The ninth character is only present for a file containing a main program for a test comprising multiple files and is always M.

A file name ending with the extension .TST indicates that the test depends on one or more of the implementation-dependent parameters listed in Appendix B. A file name ending with .DEP indicates that the test is not necessarily applicable to all implementations because it depends upon the support of language features that a compiler may legally not implement.

A test may comprise several separate compilation units contained in two or more files; the names of such files are indented under the name of the test. The letter "M" indicates which of these files contains the main procedure."

END OF DOCUMENT

END

2-87

DTIC