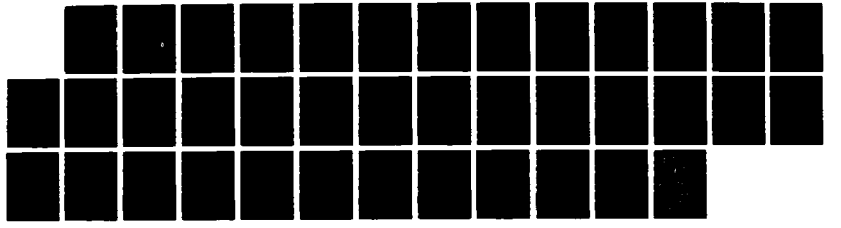
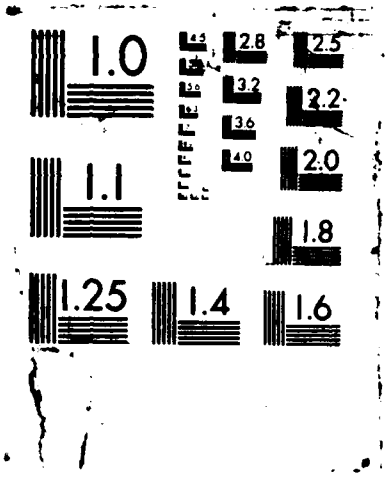
AD-A190 362 ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT: 1/1
APOLLO COMPUTER INC.. (U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI.. 19 JUN 87
UNCLASSIFIED F/G 12/5 NL




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

SECURI

ered

DTIC FILE COPY

(2)

PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. RE **AD-A190 362** 2. GOVT ACCESSION NO. 3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Ada Compiler Validation Summary Report:
Apollo Computer, Inc. DOMAIN/Ada, Version 1.0 Apollo DN3000

5. TYPE OF REPORT & PERIOD COVERED
19 June 1987 to 19 June 1988

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Wright-Patterson AFB

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS
Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

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

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

12. REPORT DATE
19 June 1987

13. NUMBER OF PAGES
35

14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office)
Wright-Patterson

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 JAN 06 1988 S D

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

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE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

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the DOMAIN/Ada, Version 1.0, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The DOMAIN/Ada is hosted on an Apollo DN3000 operating under DOMAIN/IX, Version SR 9.6. Programs processed by this compiler may be executed on an Apollo DN3000 operating under DOMAIN/IX, Version SR 9.6.

On-site testing was performed 15 June 1987 through 19 JUNE 1987 at Chelmsford, MA, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2210 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 170 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2210 tests were processed, results for Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were eight of the processed tests determined to be inapplicable. The remaining 2202 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	102	252	334	244	161	97	138	261	130	32	218	233	2202
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	73	86	3	0	0	1	1	0	0	0	0	178
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

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

AVF Control Number: AVF-VSR-99.0887
87-02-09-ACI

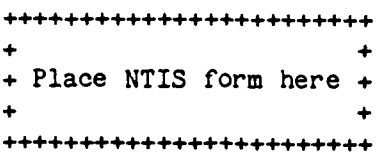
Ada® COMPILER
VALIDATION SUMMARY REPORT:
Apollo Computer, Inc.
DOMAIN/Ada, Version 1.0
Apollo DN3000

Completion of On-Site Testing:
19 JUNE 1987

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

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

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


Place NTIS form here

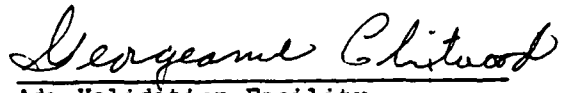
Ada® Compiler Validation Summary Report:

Compiler Name: DOMAIN/Ada, Version 1.0

Host:	Target:
Apollo DN3000 under DOMAIN/IX, BSD 4.2 Release SR 9.6	Apollo DN3000 under DOMAIN/IX, BSD 4.2 Release SR 9.6

Testing Completed 19 JUNE 1987 Using ACVC 1.8

This report has been reviewed and is approved.



Ada Validation Facility
Georgeanne Chitwood
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

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

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

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

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the DOMAIN/Ada, Version 1.0, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The DOMAIN/Ada is hosted on an Apollo DN3000 operating under DOMAIN/IX, Version SR 9.6. Programs processed by this compiler may be executed on an Apollo DN3000 operating under DOMAIN/IX, Version SR 9.6.

On-site testing was performed 15 June 1987 through 19 JUNE 1987 at Chelmsford, MA, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2210 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 170 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2210 tests were processed, results for Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were eight of the processed tests determined to be inapplicable. The remaining 2202 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	102	252	334	244	161	97	138	261	130	32	218	233	2202
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	73	86	3	0	0	1	1	0	0	0	0	178
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

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

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

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

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

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

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

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

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

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 15 June 1987 through 19 JUNE 1987 at Chelmsford, MA.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

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

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

1.3 REFERENCES

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

2. Ada Validation Organization: Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

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

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	A test found to be incorrect and not used to check conformity to the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

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

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

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

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

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers

permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

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

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

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

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

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation.

Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

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

Compiler: DOMAIN/Ada, Version 1.0

ACVC Version: 1.8

Certificate Number: 870615W1.08073

Host Computer:

Machine:	Apollo DN3000
Operating System:	DOMAIN/IX, BSD 4.2 Release SR 9.6
Memory Size:	2 megabytes

Target Computer:

Machine:	Apollo DN3000
Operating System:	DOMAIN/IX, BSD 4.2 Release SR 9.6
Memory Size:	2 megabytes

2.2 IMPLEMENTATION CHARACTERISTICS

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

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types SHORT_INTEGER, SHORT_FLOAT, and TINY_INTEGER in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

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

- Array types.

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

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

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array subtype is declared. (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 NUMERIC_ERROR when the array type is declared. (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, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

  In the evaluation of an aggregate containing subaggregates, 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.)

. Functions.

An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declaration, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declaration. (See test E66001D.)

. Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation accepts 'SIZE and 'STORAGE_SIZE for tasks, 'STORAGE_SIZE for collections, and 'SMALL clauses. Enumeration representation clauses, including those that specify noncontiguous values, appear to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is supported for procedures and functions. (See tests CA3004E and CA3004F.)

. Input/output.

The packages SEQUENTIAL_IO and DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

An existing text file can be opened in OUT_FILE mode, 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 both reading and writing. (See tests CE3111A..E (5 tests).)

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

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

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

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

. Generics.

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

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

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of DOMAIN/Ada was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 178 tests were inapplicable to this implementation, and that the 2202 applicable tests were passed by the implementation.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	865	1192	17	13	46	2202
Failed	0	0	0	0	0	0	0
Inapplicable	0	2	176	0	0	0	178
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	102	252	334	244	161	97	138	261	130	32	218	233	2202
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	73	86	3	0	0	1	1	0	0	0	0	178
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

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 is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 178 tests were inapplicable for the reasons indicated:

- C34001E, B52004D, B55B09C, and C55B07A use LONG INTEGER which is not supported by this compiler.
- C34001G and C35702B use LONG FLOAT which is 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.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

- The following 170 tests require a floating-point accuracy that exceeds the maximum of 15 supported by the implementation:

C24113L..Y (14 tests)
C35705L..Y (14 tests)
C35706L..Y (14 tests)
C35707L..Y (14 tests)
C35708L..Y (14 tests)
C35802L..Y (14 tests)
C45241L..Y (14 tests)
C45321L..Y (14 tests)
C45421L..Y (14 tests)
C45424L..Y (14 tests)
C45521L..Z (15 tests)
C45621L..Z (15 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for 19 Class B tests:

B24204A	B33301A	B67001A
B24204B	B37201A	B67001B
B24204C	B38008A	B67001C
B2A003A	B41202A	B67001D
B2A003B	B44001A	B91003B
B2A003C	B64001A	B95001A
		B97102A

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by the DOMAIN/Ada was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and that the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the DOMAIN/Ada using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of an Apollo DN3000 operating under DOMAIN/IX, Version SR 9.6.

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 splits during the prevalidation testing were included in their split form on the magnetic tape.

The contents of the magnetic tape were loaded onto a DN 500-T computer, which was acting as a file server for thirteen Apollo DN 3000 computers used to run the tests. After the test files were loaded to disk, the full set of tests was compiled on the Apollo DN3000, and all executable tests were linked and run. Results were printed.

The compiler was tested using command scripts provided by Apollo Computer, Inc., and was reviewed by the validation team. The following options were in effect for testing:

Option	Effect
-el	Produce an error listing (Class B tests only).
-M	Compile and link using the source file name as the name of the main entry point (Single file executable tests only).

Tests were compiled, linked, and executed (as appropriate) using thirteen Apollo DN 3000 computers that were identical except for the amount of real memory in each, which varied from four megabytes to eight megabytes (most had eight megabytes). 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

The validation team arrived at Chelmsford, MA on 15 June 1987, and departed after testing was completed on 19 June 1987.

APPENDIX A

DECLARATION OF CONFORMANCE

Apollo Computer, Inc. has submitted the following declaration of conformance concerning the DOMAIN/Ada.

DECLARATION OF CONFORMANCE

Compiler Implementor: Apollo Computers, Inc.
Ada Validation Facility: ASD/SCOL, Wright-Patterson AFB, OH
Ada Compiler Validation Capability (ACVC) Version: 1.8

Base Configuration

Base Compiler Name: DOMAIN/Ada	Version: Version 1.0
Host Architecture ISA: Apollo DN3000	OS&VER #: DOMAIN/IX, BSD4.2 Release SR 9.6
Target Architecture ISA: Apollo DN3000	OS&VER #: DOMAIN/IX, BSD4.2 Release SR 9.6

Implementor's Declaration

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

Alan P. Lehotsky Date: 13 June, 1987
Apollo Computers, Inc.
Alan P. Lehotsky, Ada Compiler Project Engineer

Owner's Declaration

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

Alan P. Lehotsky Date: 13 June, 1987
Apollo Computers, Inc.
Alan P. Lehotsky, Ada Compiler Project Engineer

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

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the DOMAIN/Ada, Version 1.0, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -2 147 483 648 .. 2 147 483 647;
     type SHORT_INTEGER is range -32768 .. 32767;
     type TINY_INTEGER  is range -128 .. 127;

     type FLOAT is digits 15 range -1.79769313486231E+308
                                .. 1.79769313486231E+308;
     type SHORT_FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;

     type DURATION is delta 2#1.0#E-14 range
                                -2#100000000000000000.0#
                             .. 2#1111111111111111.11111111111111#;

     ...

end STANDARD;
```

14. Ada RM Appendix F

14.1 Implementation-dependent Pragmas

DOMAIN Ada provides for sharing of generic bodies (procedures and packages), when the generic parameters are restricted to enumeration types, integer types, and floating types.

PRAGMA SHARE_BODY is used to indicate desire to share or not share an instantiation. The pragma may reference the generic unit or the instantiated unit. When it references a generic unit, it sets sharing on/off for all instantiations of that generic, unless overridden by specific SHARE_BODY pragmas for individual instantiations. When it references an instantiated unit, sharing is on/off only for that unit. The default is to share all generics that can be shared, unless the unit uses PRAGMA INLINE.

PRAGMA SHARE_BODY is only allowed in the following places: immediately within a declarative part, immediately within a package specification, or after a library unit in a compilation, but before any subsequent compilation unit. The form of this pragma is shown below.

```
pragma SHARE_BODY ( generic-name, boolean-literal )
```

Note that a parent instantiation is independent of any individual instantiation, therefore recompilation of a generic with different parameters has no effect on other compilations that reference it. The unit that caused compilation of a parent instantiation need not be referenced in any way by subsequent units that share the parent instantiation.

Sharing generics causes a slight execution time penalty because all type attributes must be indirectly referenced (as if an extra calling argument were added). However, it substantially reduces compilation time in most circumstances and reduces program size.

We have compiled a unit, SHARED_IO, in the **standard** library that instantiates all Ada generic I/O packages. Thus, any instantiation of an Ada I/O generic package will share one of the parent instantiation generic bodies. The PRAGMA SHARE_BODY takes the name of a generic instantiation or a generic unit as the first argument and one of the identifiers TRUE or FALSE as the second argument. This pragma is only allowed immediately at the place of a declarative item in a declarative part or package specification, or after a library unit in a compilation, but before any subsequent compilation unit.

When the first argument is a generic unit, the pragma applies to all instantiations of that generic. When the first argument is the name of a generic

instantiation the pragma applies only to the specified instantiation, or overloaded instantiation.

If the second argument is TRUE, the compiler will try to share code generated for a generic instantiation with code generated for other instantiations of the same generic. When the second argument is FALSE, each instantiation will get a unique copy of the generated code. The extent to which code is shared between instantiations depends on this pragma and the kind of generic formal parameters declared for the generic unit.

PRAGMA INTERFACE_OBJECT allows variables defined in another language to be referenced directly in Ada. PRAGMA INTERFACE_OBJECT will replace all occurrences of *object_name* with an external reference to *link_name* in the object file using the format shown below.

```
pragma INTERFACE_OBJECT (object-name, "link-name");
```

This pragma is allowed at the place of a declarative item in a package specification and must apply to an object declared earlier in the same package specification. The object must be declared as a scalar or an access type. The object *cannot* be any of the following.

loop variable
constant
initialized variable
array
record

PRAGMA EXTERNAL_NAME allows the user to specify a *link_name* for an Ada variable or subprogram so that the Ada object can be referenced from other languages using the syntax shown below.

```
pragma EXTERNAL_NAME (object-or-subprogram-name, "link-name");
```

The object must be a variable defined in a package specification; a subprogram can be library level or within a package specification.

14.2 Implementation-dependent Attributes

There are no implementation-dependent attributes in DOMAIN Ada.

14.3 Specification of package SYSTEM

```
package SYSTEM is

   type NAME is (apollo_4_2_unix);

   SYSTEM_NAME        : constant NAME := apollo_4_2_unix;
```

```
   STORAGE_UNIT        : constant        := 8;
   MEMORY_SIZE         : constant        := 16_777_216;

   -- System-dependent Named Numbers

   MIN_INT             : constant        := -2_147_483_647 - 1;
   MAX_INT             : constant        := 2_147_483_647;
   MAX_DIGITS          : constant        := 15;
   MAX_MANTISSA        : constant        := 31;
   FINE_DELTA          : constant        := 2.0 ** (-14);
   TICK                : constant        := 0.01;

   -- Other System-dependent Declarations

   subtype PRIORITY is INTEGER range 0 .. 7;

   MAX_REC_SIZE : integer := 64*1024;

   type ADDRESS is private;

   NO_ADDR: constant ADDRESS;

   function PHYSICAL_ADDRESS(I: INTEGER) return ADDRESS;
   function ADDR_GT(A, B: ADDRESS) return BOOLEAN;
   function ADDR_LT(A, B: ADDRESS) return BOOLEAN;
   function ADDR_GE(A, B: ADDRESS) return BOOLEAN;
   function ADDR_LE(A, B: ADDRESS) return BOOLEAN;
   function ADDR_DIFF(A, B: ADDRESS) return INTEGER;
   function INCR_ADDR(A: ADDRESS; INCR: INTEGER) return ADDRESS;
   function DECR_ADDR(A: ADDRESS; DECR: INTEGER) return ADDRESS;

   function ">"(A, B: ADDRESS) return BOOLEAN renames ADDR_GT;
   function "<"(A, B: ADDRESS) return BOOLEAN renames ADDR_LT;
   function ">="(A, B: ADDRESS) return BOOLEAN renames ADDR_GE;
   function "<="(A, B: ADDRESS) return BOOLEAN renames ADDR_LE;
   function "-"(A, B: ADDRESS) return INTEGER renames ADDR_DIFF;
   function "-"(A: ADDRESS; INCR: INTEGER) return ADDRESS renames
      INCR_ADDR;
   function "-"(A: ADDRESS; DECR: INTEGER) return ADDRESS renames
      DECR_ADDR;

   pragma built_in(PHYSCIAL_ADDRESS);
   pragma inline(ADDR_GT);
   pragma inline(ADDR_LT);
   pragma inline(ADDR_GE);
   pragma inline(ADDR_LE);
   pragma inline(ADDR_DIFF);
   pragma inline(INCR_ADDR);
   pragma inline(DECR_ADDR);

private

   type ADDRESS is new INTEGER;
   NO_ADDR : constant ADDRESS := 0;

   end SYSTEM;
```

14.4 Restrictions on Representation Clauses

PRAGMA PACK — Objects and components are packed to the nearest power of two bits.

Record Representation Clauses — The only restriction on record representation specifications is the following: if a field does not start and end on a storage unit boundary, it must be possible to move it into a register with a single instruction.

Address Clauses — Address clauses are not supported.

Interrupts — Interrupts are not supported.

Representation Attributes — The ADDRESS attribute is not supported for the following entities.

static constants
packages
tasks
labels
entries

Machine Code Insertions — Machine code insertions are supported.

14.5 Conventions for Implementation-generated Names

There are no implementation generated names.

14.6 Interpretation of Expressions in Address Clauses

Address clauses are not supported.

14.7 Restrictions on Unchecked Conversions

The predefined generic function UNCHECKED_CONVERSION cannot be instantiated with a target type that is an unconstrained array type or an unconstrained record type with discriminants.

14.8 Implementation Characteristics of I/O Packages

Instantiations of DIRECT_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds

that value. For example, for unconstrained arrays such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_RECORD_SIZE is defined in SYSTEM and can be changed by a program before instantiating DIRECT_IO to provide an upper limit on the record size. In any case, the maximum size supported is 1024 * 1024 * STORAGE_UNITs. DIRECT_IO will raise USE_ERROR if MAX_REC_SIZE exceeds this absolute limit.

Instantiations of SEQUENTIAL_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_RECORD_SIZE is defined in SYSTEM and can be changed by a program before instantiating INTEGER_IO to provide an upper limit on the record size. SEQUENTIAL_IO imposes no limit on MAX_REC_SIZE.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..498 => 'A', 499 => '1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..498 => 'A', 499 => '2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..249 => 'A', 250 => '3', 251..499 => 'A')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..249 => 'A', 250 => '4', 251..499 => 'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..496 => '0', 497..499 => "298")

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be the size of the maximum line length.	(1..493 => '0', 494..499 => "69.0E1")
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1..479 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	2_147_483_647
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters, or is too long if no invalid characters exist.	"/illegal/file name/2{]$%2102C.DAT"
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character, or is too long if no wild card character exists.	"illegal/file name/CE2102C*.DAT"
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in the range of DURATION.	100_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	10_000_000.0

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	"/no/such/directory/ILLEGAL_EXTERNAL_FILE_NAME1"
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	"/no/such/directory/ILLEGAL_EXTERNAL_FILE_NAME2"
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2 147 483_648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2 147 483 647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-100 000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-10 000 000.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	15
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	499
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2 147 483 647

TEST PARAMETERS

Name and Meaning	Value
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	TINY_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFD#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- C32114A: An unterminated string literal occurs at line 62.
- B33203C: The reserved word "IS" is misspelled at line 45.
- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.
- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.
- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.
- C41404A: The values of 'LAST and 'LENGTH are incorrect in the if statements from line 74 to the end of the test.
- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.
- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.
- B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.
- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

WITHDRAWN TESTS

- B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

- C87B50A: The call of "/=" at line 31 requires a use clause for package A.

- C92005A: The "/=" for type PACK.BIG INT at line 40 is not visible without a use clause for the package PACK.

- C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- BC3204C: The body of BC3204C0 is missing.

END

DATE

FILMED

5-88

DTIC