UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A208 303

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: NAVAL UNDERWATER SYSTEMS COMMAND, ADAUYK44 (ALS/N Ada/M), Version 1.0, VAX 11/785 (host) to AN/UYK-44 (target) 880719S1.09155	5. TYPE OF REPORT & PERIOD COVERED 19 July 1988 - 19 July 1989 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) National Bureau of Standards, Gaithersburg, Maryland, USA	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Bureau of Standards, Gaithersburg, Maryland, USA	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 July 1988 13. NUMBER OF PAGES 43 p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) National Bureau of Standards, Gaithersburg, Maryland, USA	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 MAY 25 1989 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)

ADAUYK44 (ALS/N Ada/M), Version 1.0, NAVAL UNDERWATER SYSTEMS COMMAND, National Bureau of Standards, VAX 11/785 under VAX/VMS, Version 4.5 (host) to AN/UYK-44 under Bare machine (target), ACVC 1.09

89 5 24 014

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE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)

AVF Control Number: NBS88VUSN525_4

Ada Compiler
VALIDATION SUMMARY REPORT:
Certificate Number: 880719S1.09155
NAVAL UNDERWATER SYSTEMS COMMAND
ADAUYK44 (ALS/N Ada/M), Version 1.0
VAX 11/785 Host and AN/UYK-44 Target

Completion of On-Site Testing:
19 July 1988

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

AVF Control Number: NBS88VUSN525_4

Ada Compiler Validation Summary Report:

Compiler Name: ADAUYK44 (ALS/N Ada/M), Version 1.0

Certificate Number: 880719S1.09155

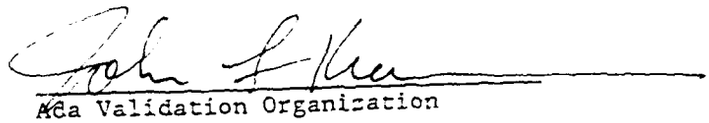
Host:
VAX 11/785 under
VAX/VMS,
Version 4.5

Target:
AN/UYK-44 under
Bare machine

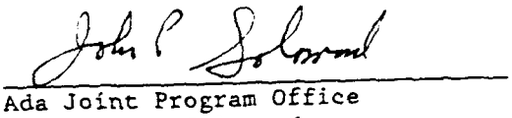
Testing Completed 19 July 1988 Using ACVC 1.9

This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899



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



Ada Joint Program Office
Dr. John P. Solomond
Director
Department of Defense
Washington DC 20301

AVF Control Number: NBS88VUSN525_4

Ada Compiler Validation Summary Report:

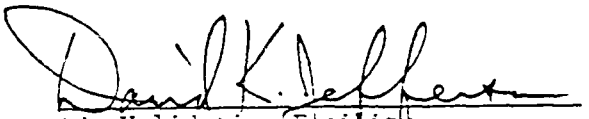
Compiler Name: ADAUYK44 (ALS/N Ada/M), Version 1.0

Certificate Number: 880719S1.09155

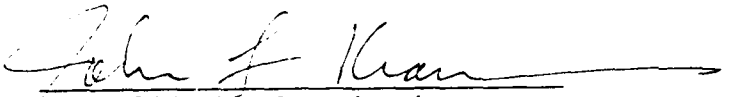
Host:	Target:
VAX 11/785 under VAX/VMS, Version 4 5	AN/UYK-44 under Bare machine

Testing Completed 19 July 1988 Using ACVC 1.9

This report has been reviewed and is approved.



Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899



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

Ada Joint Program Office
Dr. John P. Solomond
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

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

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

This 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 test 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 the National Bureau of Standards, under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was completed 19 July 1988, at Syscon Corportation, Washington, D.C.

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:

> Software Standards Validation Group
> Institute for Computer Sciences and Technology
> National Bureau of Standards
> Building 225, Room A266
> Gaithersburg, Maryland 20899

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

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

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

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

3. Ada Compiler Validation Capability Implementers' Guide., December 1986.

1.4 DEFINITION OF TERMS

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

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Language Maintenance Panel	The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and possible changes to the ANSI/MIL-STD for Ada.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	An Ada program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

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

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A

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

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

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

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

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

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

Two library units, the package REPORT and the procedure CHECK FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of

REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

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

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

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

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

Compiler: ADAUYK44 (ALS/N Ada/M), Version 1.0

ACVC Version: 1.9

Certificate Number: 880719S1.09155

Host Computer:

Machine:	VAX 11/785
Operating System:	VAX/VMS Version 4.5
Memory Size:	12 MB

Target Computer:

Machine:	AN/UYK-44
Operating System:	Bare machine
Memory Size:	576 K words (64K Core and 512K semi-conductor)

Communications Network: PORTAL/44

All test pre-runtime processing (compile, link, export) was performed on the host computer. The generated executables were run on the target using a Navy-developed tool called PORTAL/44. PORTAL/44 was used to download the executables to the target, execute the test, and upload the contents of the report buffer to the host. The buffer was then analyzed to determine test pass/fail status.

2.2 IMPLEMENTATION CHARACTERISTICS

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

- Capacities.

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

- Universal integer calculations.

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

- Predefined types.

  This implementation supports the additional predefined type LONG_INTEGER in the package STANDARD. (See tests B86001BC and B86001D.)

- Based literals.

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

- Expression evaluation.

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

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

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

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

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

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

- Rounding.

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

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

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

- Array types.

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

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

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

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

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises STORAGE_ERROR. (See test C52103X.)

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

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

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

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

In assigning record types with disciminants, 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.)

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

- Representation clauses.

  An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is not supported, then the implementation must reject it.

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

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

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

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

  Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

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

  Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

  Record representation clauses are not supported. (See test A39005G.)

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

- Pragmas.

  The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

  The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants

without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

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

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

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

RESET and DELETE are not supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are not supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2106A and CE2106B.)

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

An existing text file cannot be opened in OUT_FILE mode, cannot be created in OUT_FILE mode, and cannot be created in IN_FILE mode. (See test EE3102C.)

- Generics.

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

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

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

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tests, 28 tests had been withdrawn because of test errors. The AVF determined that 539 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing. Modifications to the code, processing, or grading for 29 tests were required to successfully demonstrate the test objective. (See section 3.6.)

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

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	105	1048	1329	15	12	46	2555
Inapplicable	5	3	524	2	5	0	539
Withdrawn	3	2	21	0	2	0	28
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	181	452	465	245	163	98	141	327	137	36	234	3	73	2555
Inapplicable	23	120	209	3	2	0	2	0	0	0	0	0	180	539
Withdrawn	2	14	3	0	1	1	2	0	0	0	2	1	2	28
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 28 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C	C35904A
C35904B	C35A03E	C35A03R	C37213H	C37213J	C37215C
C37215E	C37215G	C37215H	C38102C	C41402A	C45332A
C45614C	E66001D	A74106C	C85018B	C87B04B	CC1311B
BC3105A	AD1A01A	CE2401H	CE3208A		

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 539 test were inapplicable for the reasons indicated:

C35702A uses SHORT_FLOAT which is not supported by this implementation.

C35702B uses LONG_FLOAT which is not supported by this implementation.

A35801E attempts to define a floating-point type with a range FLOAT'FIRST..FLOAT'LAST; but these bounds are not safe numbers of the

base type (FLOAT) and thus the declaration is rejected.

A39005G uses a record representation clause which is not supported by this compiler.

The following (14) tests use SHORT_INTEGER, which is not supported by this compiler.

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

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

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

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

C4A012B requires CONSTRAINT_ERROR to be raised in a context where NUMERIC_ERROR is relevant.

D64005F and D64005G compile successfully but do not link in the 64K memory capability of the target.

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

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.

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

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

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

CE2102C	CE2102G..H(2)	CE2102K	CE2104A..D(4)
CE2105A..B(2)	CE2106A..B(2)	CE2107A..I(9)	CE2108A..D(4)
CE2109A..C(3)	CE2110A..C(3)	CE2111A..E(5)	CE2111G..H(2)
CE2115A..B(2)	CE2201A..C(3)	CE2201F..G(2)	CE2204A..B(2)

CE2208B	CE2210A	CE2401A..C(3)	CE2401E..F(2)
CE2404A	CE2405B	CE2406A	CE2407A
CE2408A	CE2409A	CE2410A	CE2411A
CE3102B	EE3102C	AE3101H	CE3103A
CE3104A	CE3107A	CE3108A..B(2)	CE3109A
CE3110A	CE3111A..E(5)	CE3112A..B(2)	CE3114A..B(2)
CE3115A	CE3203A	CE3301A..C(3)	CE3302A
CE3305A	CE3402A..D(4)	CE3403A..C(3)	CE3403E..F(2)
CE3404A..C(3)	CE3405A..D(4)	CE3406A..D(4)	CE3407A..C(3)
CE3408A..C(3)	CE3409A	CE3409C..F(4)	CE3410A
CE3410C..F(4)	CE3411A	CE3412A	CE3413A
CE3413C	CE3602A..D(4)	CE3603A	CE3604A
CE3605A..E(5)	CE3606A..B(2)	CE3704A..B(2)	CE3704D..F(3)
CE3704M..O(3)	CE3706D	CE3706F	CE3804A..E(5)
CE3804G	CE3804I	CE3804K	CE3804M
CE3805A..B(2)	CE3806A	CE3806D..E(2)	CE3905A..C(3)
CE3905L	CE3906A..C(3)	CE3906E..F(2)	

The following 327 tests require a floating-point accuracy that exceeds the maximum of 6 digits supported by this implementation:

C24113C..Y (23 tests)	C35705C..Y (23 tests)
C35706C..Y (23 tests)	C35707C..Y (23 tests)
C35708C..Y (23 tests)	C35802C..Z (24 tests)
C45241C..Y (23 tests)	C45321C..Y (23 tests)
C45421C..Y (23 tests)	C45521C..Z (24 tests)
C45524C..Z (24 tests)	C45621C..Z (24 tests)
C45641C..Y (23 tests)	C46012C..Z (24 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

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

Modifications were required for 24 Class B tests, 2 Class C tests, and 3 Class A tests.

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

B2A003A	B2A003B	B2A003C	B33201C	B33202C	B33203C
B33301A	B37106A	B37201A	B37301I	B37303A	B37307B
B38001C	B38003A	B38003B	B38009A	B38009B	B44001A

B51001A B54A01C B54A01L BC1008A BC1201L BC3013A

AE2101A, AE2101F, and AE3702A contain 11, 11, and 6 instantiations of SEQUENTIAL_IO, DIRECT_IO, and INTEGER_IO, respectively; C35A06N and CC1221A contain 17 and 21 instantiations of test-defined generic units. These tests result in object modules that will not fit in one virtual window (64K words). Although the AVF initially considered these tests to be not applicable, the AVO directed that subsequent testing be done with split versions of these tests; the tests were split into smaller units and were passed.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the ADAUYK44 (ASL/N Ada/M) was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the ADAUYK44 (ALS/N Ada/M) using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 11/785 operating under VAX/VMS, Version 4.5 and a AN/UYK-44 target operating under Bare machine. The host and target computers were linked via PORTAL/44.

A magnetic tape containing all tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on-site after the magnetic tape was loaded. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape. The contents of the magnetic tape were not loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the VAX 11/785, and all executable tests were run on the AN/UYK-44. Object files were linked on the host computer, and executable images were transferred to the target computer via PORTAL/44. Results were printed from the host computer, with results being transferred to the host computer via PORTAL/44.

The compiler was tested using command scripts provided by Naval Underwater Systems Command and reviewed by the validation team. The compiler was tested using all default option settings.

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target computer. Test output,

compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Syscon Corportation, Washington, D.C., and was completed on 19 July 1988.

APPENDIX A

CONFORMANCE STATEMENT

APPENDIX A

U.S. NAVY/SOFTECH INC. DECLARATION OF CONFORMANCE

Compiler Implementer: SofTech, Inc.
2000 N. Beauregard St.
Alexandria, VA 22311-1794

Ada Validation Facility: National Bureau of Standards
Institute for Computer Science and Technology (ICST)
Ada Validation Facility (AVF)
Building 225 Room A266
Gaithersburg, MD 20899

Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name: ADAUYK44 (ALS/N Ada/M) Version: 1.0
Host Architecture ISA: Digital VAX OS&VER #: VMS 4.5
Target Architecture ISA: AN/UYK-44 OS&VER #: N/A

Implementer's Declaration

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

Robert L. Talkan, Ada/M Program Manager — Date: April 14 1989

Owner's Declaration

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

William L. Wilder, Ada/APSE Development Branch Head — Date: April 15, 1989

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the ADAUYK44 (ALS/N Ada/M), Version 1.0, are described in the following sections which discuss topics in Appendix F of the Ada Standard. Implementation- specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is


     type INTEGER is range -32_768 .. 32_767;
     type LONG_INTEGER is range -2_147_483_648 .. 2_147_483_647;

     type FLOAT is digits 6 range
                    -(2#0.1111_1111_1111_1111_1111_1111#E252) ..
                     (2#0.1111_1111_1111_1111_1111_1111#E252);
     type DURATION is delta 2#0.0000_0000_0000_01# range
                                          -86_400.0 .. 86_400.0;


end STANDARD;
```

APPENDIX F

APPENDIX F OF THE ADA LRM FOR THE ADAUYK44 TOOLSET

F.1 PRAGMAS

F.1.1 Implementation-Defined Pragmas

This paragraph describes the use and meaning of those pragmas recognized by Ada/M which are not specified in Appendix B of [ANSI/MIL-STD-1815A].

pragma TITLE (arg);

This is a listing control pragma. It takes a single argument of type string. The string specified will appear on the second line of each page of every listing produced for a compilation unit. At most one such pragma may appear for any compilation unit, and it must be the first lexical unit in the compilation unit (excluding comments).

F.1.2 Language-Defined Pragmas

This paragraph specifies implementation specific changes to those pragmas described in Appendix B of [ANSI/MIL-STD-1815A].

pragma CONTROLLED (arg);

No change.

pragma ELABORATE (arg [,arg,...]);

No change.

pragma INLINE (arg [,arg,...]);

Subprograms specified as the argument to an INLINE pragma, and which are directly recursive are not expanded in-line at the point of the recursive invocation. Such calls use normal Ada subprogram calling semantics.

pragma INTERFACE (arg, arg);

The first argument specifies the language and type of interface to be used in calls to the externally supplied subprogram, specified by the second argument. The only value allowed for the first argument (language name) is MACRO_NORMAL.

pragma LIST (arg);

No change.

pragma MEMORY_SIZE (arg);

This pragma must appear at the start of the first compilation when creating a program library. If it appears elsewhere, a diagnostic of severity WARNING is generated and the pragma has no effect.

pragma OPTIMIZE (arg);

The argument is either TIME or SPACE. The default is SPACE. This pragma will be effective only when the OPTIMIZE option has been given to the Compiler.

pragma PACK (arg);

No change.

pragma PRIORITY (arg);

The argument is an integer static expression in the range 0..15. If the value of the argument is out of range, the pragma will have no effect other than to generate a WARNING diagnostic. A value of zero will be used if priority is not defined. The pragma will have no effect when not specified in a task (type) specification or the outermost declarative part of a subprogram. If the pragma appears in the declarative part of a subprogram, it will have no effect unless that subprogram is designated as the main subprogram at link time.

pragma SHARED (arg);

No change.

pragma STORAGE_UNIT (arg);

This pragma must appear at the start of the first compilation when creating a program library. If it appears elsewhere, a diagnostic of severity WARNING is generated and the pragma has no effect.

pragma SUPPRESS (arg[,arg]);

This pragma is unchanged with the following exceptions:

Suppression of OVERFLOW_CHECK applies only to integer operations; and a SUPPRESS pragma has effect only within the compilation unit in which it appears except that suppression of ELABORATION_CHECK applied at the declaration of a subprogram or task unit applies to all calls or activations.

ragma SYSTEM_NAME (arg);

This pragma must appear at the start of the first compilation when creating a program library. If it appears elsewhere, a diagnostic of severity WARNING is generated and the pragma has no effect. The allowable values for the argument are the enumeration literals ANUYK44 and ANAYK14. For other values, a WARNING diagnostic is generated.

'.1.3 Scope Of Pragmas

The scope for each pragma recognized by the Ada/M Compiler is iven below.

ONTROLLED	Applies only to the access type named in its argument.
LABORATE	Applies to the entire compilation unit in which the pragma appears.
NLINE	Applies only to subprograms named in its arguments. If the argument is an overloaded subprogram name, the INLINE pragma applies to all definitions of that subprogram name which appear in the same declarative part as the INLINE pragma.
NTERFACE	Applies to all invocations of the named imported subprogram.
IST	Applies from the point of its appearance until the next LIST pragma in the source or included text, or if none, the end of the compilation unit.
EMORY_SIZE	Applies to the entire Program Library in which the pragma appears.
PTIMIZE	Applies to the entire compilation unit in which the pragma appears.
ACK	Applies only to the array or record named as the argument.
AGE	No scope is associated with PAGE.
RIORITY	Applies to the task specification in which it appears, or

to the environment task if it appears in the main subprogram.

SHARED — Applies to the scope of the scalar or access variable named by the argument.

STORAGE_UNIT — Applies to the entire Program Library in which the pragma appears.

SUPPRESS — Applies to the block or body that contains the declarative part in which the pragma appears.

SYSTEM_NAME — Applies to the entire Program Library in which the pragma appears.

TITLE — Applies to the compilation unit in which the pragma appears.

F.2 ATTRIBUTES

There is one implementation-defined attributes in addition to the predefined attributes found in Appendix A of [ANSI/MIL-STD-1815A]. These are defined below.

p'DISP — For a prefix p that denotes a data object:

Yields a value of type universal_integer, which corresponds to the offset of p from the beginning of the frame containing p. For example, if p names a parameter, p'DISP is the offset of p from the argument pointer on the stack.

This attribute differs from the ADDRESS attribute in that ADDRESS supplies the virtual absolute address whereas DISP supplies a displacement. It is the user's responsibility to determine the base value, (i.e., frame pointer, argument pointer, psect), relevant to the object p.

The following definitions augment the language-required definitions of the predefined attributes found in Appendix A of [ANSI/MIL-STD-1815A].

T'MACHINE_ROUNDS — is false.

T'MACHINE_RADIX — is 16.

T'MACHINE_MANTISSA — is 6.

T'MACHINE_EMAX — is 63.

T'MACHINE_EMIN — is -64.

T'MACHINE_OVERFLOWS is true.

F.3 PREDEFINED LANGUAGE ENVIRONMENT

The package STANDARD contains the following definitions in addition to those specified in Appendix C of [ANSI/MIL-STD-1815A].

```
TYPE integer IS RANGE -32_768..32_767;

TYPE long_integer IS RANGE -2_147_483_648..2_147_483_647;

TYPE float IS DIGITS 6 RANGE
    -(2#0.1111_1111_1111_1111_1111_1111#E252)
   ..(2#0.1111_1111_1111_1111_1111_1111#E252);

TYPE duration IS
    delta 2#0.0000_0000_0000_01# range -86_400.0 .. 86_400.0
          -- 1/16384                    -- one day.
```

The package SYSTEM for Ada/M is as given below.

```
TYPE name IS (anuyk44, anayk14);
             -- only one compatible system name.

system_name : CONSTANT system.name := system.anuyk44;
              -- name of current system.

storage_unit : CONSTANT := 16;
               -- word-oriented system (not configurable).

memory_size : CONSTANT := 65_536;
              -- virtual memory size (not configurable).

TYPE address IS RANGE 0..system.memory_size - 1;
                -- virtual address.

FOR address'SIZE USE 16;
                 -- virtual address is a 16-bit quantity.

physical_memory_size : CONSTANT :=  2**22;
                       -- maximum physical memory size (not
                       -- configurable).

TYPE physical_address IS
   RANGE 0..system.physical_memory_size - 1;
   -- absolute address.

TYPE external_interrupt_word IS RANGE 0 .. 65_536;
                             -- Parameter type for Class III Priority
                             -- 2
                             -- External Interrupt entry.
```

```
        min_int : CONSTANT := -(2**31);
                  -- most negative integer.

        max_int : CONSTANT := (2**31)-1;
                  -- most positive integer.

        max_digits : CONSTANT := 6;
                     -- most decimal digits in floating point
                     -- constraint.

        max_mantissa : CONSTANT := 31;
                       -- most binary digits for fixed point subtype.

        fine_delta : CONSTANT
           := 2#0.0000_0000_0000_0000_0000_0000_0000_001#;
           -- 2**(-31) is minimum fixed point constraint.

        tick : CONSTANT := 3.125e-05;
               -- 1/32000 seconds is the basic clock period.

        SUBTYPE priority IS integer RANGE 0..15;
                            -- task priority, lowest = default = 0.

        -- implementation-defined exceptions.
        unresolved_reference : EXCEPTION;
        capacity_error       : EXCEPTION;
        system_error         : EXCEPTION;

    END system;
```

F.4 REPRESENTATION AND DECLARATION RESTRICTIONS

Representation specifications are described in Section 13 of [ANSI/MIL-STD-1815A]. Declarations are described in Section 3 of [ANSI/MIL-STD-1815A].

In the following specifications, the capitalized word SIZE indicates the number of bits used to represent an object of the type under discussion. The upper case symbols D, L, R, correspond to those discussed in Section 3.5.9 of [ANSI/MIL-STD-1815A].

F.4.1 Integer Types

Integer types are specified with constraints of the form

```
RANGE L..R
```

where

R <= system.max_int

d

L >= system.min_int

r a prefix "t" denoting an integer type, length specifications of the rm

FOR t'SIZE USE n ;

ay specify integer values n such that

n in 2..16,

nd such that

R <= 2**(n-1)-1

nd

L >= -(2**(n-1))

r else such that

R <= (2**n)-1

nd

L >= 0

and

1 < n <= 15

For a stand-alone object of integer type, a default SIZE of 16 is used when R <= 2**15-1 and L >= -2**15; otherwise, a SIZE of 32 is used. For components of integer types within packed composite objects, the smaller of the default stand-alone SIZE and the SIZE from a length specification is used.

F.4.2 Floating Point Types

Floating types are specified with constraints of the form:

DIGITS D

where

D is an integer in the range 1..6.

For a prefix "t" denoting a floating point type, length specifications of the form

```
FOR t'SIZE USE n;
```

are permitted only when the integer value n = 32. All floating point values have SIZE = 32.

F.4.3 Fixed Point Types

Fixed types are specified with constraints of the form

```
DELTA D RANGE L..R
```

where

$$\frac{\mathrm{MAX}\,(\mathrm{ABS}(R),\ \mathrm{ABS}(L))}{\text{actual delta}} \leq 2^{31}-1.$$

The actual delta defaults to the largest integral power of 2 less than or equal to the specified delta D. (This implies that fixed values are stored right-aligned.) For specifications of the form

```
FOR t'ACTUAL_DELTA USE n;
```

n must be specified as an integral power of 2 such that n <= D.

For a prefix "t" denoting a fixed point type, length specifications of the form

```
FOR t'SIZE USE n;
```

are permitted only when n = 32. All fixed values have SIZE = 32.

F.4.4 Enumeration Types

In the absence of a representation specification for an enumeration type "t", the integral representation of t'FIRST is 0. The default size for a stand-alone object of enumeration type "t" is 16, so the internal representations of t'FIRST and t'LAST both fall within the range

```
-2**15 .. 2**15 - 1.
```

Length specifications of the form

```
FOR t'SIZE USE n;
```

and/or enumeration representations of the form

FOR t USE aggregate;

are permitted for n in 2..16, provided the representations and the SIZE conform to the relationship specified above, or else for n in 1..15, provided that the internal representation of t'FIRST >= 0 and the representation of t'LAST <= 2**(t'SIZE) - 1.

For components of enumeration types within packed composite objects, the smaller of the default stand-alone SIZE or the SIZE length specification is used.

Enumeration representations on types derived from the predefined type standard.boolean will not be accepted, but length specifications will be accepted.

F.4.5 Access Types

The value of t'STORAGE_SIZE for an access type "t" specifies the maximum number of storage_units used for all objects in the collection for type "t". This includes all space used by the allocated objects, plus any additional storage required to maintain the collection.

F.4.6 Arrays And Records

A length specification of the form

FOR t'size USE n;

may cause arrays and records to be packed, if required, to accommodate the length specification. If the size specified is not large enough to contain any value of the type, a diagnostic message of severity ERROR is generated.

The PACK pragma may be used to minimize wasted space between components of arrays and records. The pragma causes the type representation to be chosen such that the storage space requirements are minimized at the possible expense of data access time and code space.

A record type representation specification may be used to describe the allocation of components in a record. Bits are numbered 0..15 from the left. Bit 16 starts at the left of the next higher numbered word. Each location specification must allow at least n bits of range, where n is large enough to hold any value of the subtype of the component being allocated. Otherwise, a diagnostic message of severity ERROR is generated. Components such as arrays, records, tasks, or access

ariables may not be allocated to specific locations. If a pecification of this form is entered, a diagnostic message of severity RROR is generated.

The alignment clause of the form

AT MOD n

ay only specify alignments of 1 or 2 (corresponding to word or oubleword alignment, respectively).

If it is determinable at compile time that the SIZE of a record or rray type or subtype is outside the range of standard.long_integer, a iagnostic of severity WARNING is generated. Declaration of such a ype or subtype would raise NUMERIC_ERROR when elaborated.

.4.7 Other Length Specifications

The value of t'STORAGE_SIZE for a task type "t" specifies the umber of system.storage_units that are allocated for the execution of ach task object of type "t." This includes the runtime stack for the ask object but does not include objects allocated at runtime by the ask object.

.5 SYSTEM GENERATED NAMES

There are no system generated names.

.6 RESTRICTIONS ON THE MAIN SUBPROGRAM

Refer to Section 10.1(8) of [ANSI/MIL-STD-1815A] for a description f the main subprogram. The subprogram designated as the main ubprogram cannot have parameters. The designation as the main ubprogram of a subprogram whose specification contains a formal_part esults in a diagnostic of severity ERROR at link time.

The main subprogram can be a function, but the return value will ot be available upon completion of the main subprogram's execution.

.7 ADDRESS CLAUSES

Address clauses are not supported.

F.8 UNCHECKED CONVERSIONS

Refer to Section 13.10.2 of [ANSI/MIL-STD-1815A] for a description of UNCHECKED_CONVERSION. It is erroneous if the User-Written_Ada_Program performs UNCHECKED_CONVERSION when the source and target objects have different sizes.

F.9 CHARACTER SET

Ada compilations may be expressed using the following characters in addition to the basic character set:

lower case letters:

a b c d e f g h i j k l m n o p q r s t u v w x y z

special characters:

! $? @ [\ ] ^ { } ~
` (accent grave)
%

The following transliterations are not permitted:

a. exclamation point for vertical bar,

b. colon for sharp, and

c. percent for double-quote.

F.10 IMPLEMENTATION DEFINED EXCEPTIONS

UNRESOLVED_REFERENCE
The exception UNRESOLVED_REFERENCE is raised whenever a call is made to a subprogram whose body is not included in the linked program image. In addition, this exception is raised whenever a reference to a data object cannot be resolved by the Linker/Exporter. Since Ada/M provides a selective linking capability through the use of a units list file, the subprogram body may not always be present in the linked program image.

SYSTEM_ERROR
The exception SYSTEM_ERROR signifies an internal error in the Run-Time Operating System that is not the fault of the User-Written_Ada_Program.

CAPACITY_ERROR
The exception CAPACITY_ERROR is raised by the RTExec when Pre-Runtime specified resource limits are exceeded.

F.11 I/O IMPLEMENTATION DETAILS

USE_ERROR is raised by every attempt to create or open a text, sequential, or direct access file. No allowance is provided in the I/O subsystem for memory resident files (i.e., files which do not reside on a peripheral device). This is true even in the case of temporary files.

F.11.1 Naming Conventions

The naming conventions for external files in Ada/M are of particular importance to the user. All of the system-dependent information needed by the I/O subsystem about an external file is contained in the file name. External files may be named using one of three file naming conventions: standard, temporary, and user-derived.

F.11.1.1 Standard File Names -

The standard external file naming convention used in Ada/M identifies the specific location of the external file in terms of the physical device on which it is stored. For this reason, the user should be aware of the configuration of the peripheral devices on the AN/UYK-44 at a particular user site.

Standard file names may be six to twenty characters long; however, the first six characters follow a predefined format. The first and second characters must be either "CT" or "TT", designating an AN/USH-26 Signal Data Recorder/Reproducer Set or the AN/USQ-69 Data Terminal Set, respectively. These characters must be in upper case.

The third and fourth characters specify the channel on which the peripheral device is connected. Since there are sixty-four channels on the AN/UYK-44, the values for the third and fourth positions must lie in the range "00" to "63".

The range of values for the fifth position in the external file name (the unit number) depends upon the device specified by the characters in the first and second positions of the external file name. If the specified peripheral device is the AN/USH-26 magnetic tape drive, then the character in the fifth position must be one of the characters "0", "1", "2", or "3". This value determines which of the four tape cartridge units available on the AN/USH-26 is to be accessed. If the specified peripheral device is the AN/USQ-69 militarized display terminal, the character in the fifth position must be a "0". This is the only allowable value for the unit number because the AN/USQ-69 may have only one unit on a channel.

The colon, ":", is the only character allowed in the sixth position. If any character other than the colon is in this position, the file name will be considered non-standard and the file will reside on the default device defined during the elaboration of CONFIGURE_IO.

Positions seven through twenty are optional to the user-written Ada program and may be used as desired. These positions may contain any printable character the user chooses in order to make the file name more intelligible. Embedded blanks, however, are not allowed.

The location of an external file on a peripheral device is thus a function of the first six characters of the file name regardless of the characters that might follow. For example, if the external file "CT000:Old_Data" has been created and not subsequently closed, an attempt to create the external file "CT000:New_Data" will cause the exception DEVICE_ERROR (rather than NAME_ERROR or USE_ERROR) to be raised because the peripheral device on channel "00" and cartridge "0" is already in use.

The user is advised that any file name beginning with "xxxxx:" (where x denotes any printable character) is assumed to be a standard external file name. If this external file name does not conform to the Ada/M standard file naming conventions, the exception NAME_ERROR will be raised.

F.11.1.2 Temporary File Names -

Section 14.2.1 of ANSI/MIL-STD-1815A defines a temporary file to be an external file that is not accessible after completion of the main subprogram. If the null string is supplied for the external file name, then the external file is considered temporary. In this case, the high level I/O packages internally create an external file name to be used by the lower level I/O packages. The internal naming scheme used by the I/O subsystem is a function of the type of file to be created (text, direct or sequential), the temporary nature of the external file, and the number of requests made thus far for creating temporary external files of the given type. This scheme is consistent with the requirement specified in ANSI/MIL-STD-1815A that all external file names be unique.

The first three characters of the file name are "TEX", "DIR", or "SEQ". The next six characters are "_TEMP_". The remaining characters are the image of an integer which denotes the number of temporary files of the given type successfully created. There are two types of temporary files; one is used by SEQUENTIAL_IO and DIRECT_IO, and the other is used by TEXT_IO. For instance, the temporary external file name "TEX_TEMP_10" would be the name of the tenth temporary external file successfully created by the user-written Ada program through calls to TEXT_IO.

F.11.1.3 User-Derived File Names -

A random string containing a sequence of characters of length one to twenty may also be used to name an external file. External files with names of this nature are considered to be permanent external files. The user is cautioned to refrain from using names which conform to the scheme used by the I/O subsystem to name temporary external files (see subsection (b) above).

APPENDIX C

TEST PARAMETERS

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

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	1..119 => 'A', 120 => '1'
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	1..119 => 'A', 120 => '2'
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	1..59 => 'A', 60 => '3', 61..120 => 'A'
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	1..59 => 'A', 60 => '4', 61..120 => '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..117 => '0', 118..120 => '298'
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	1..115 => '0', 116..120 => '690.0'

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	1 => '"', 2..61 => 'A', 62 => '"'
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	1 => '"', 2..60 => 'A', 61 => '1', 62 => '"'
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	1..100 => ' '
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	32_767
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	32_767
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	BAD-CHARS^#.%!X
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	WILD-CHAR*.NAM
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	90_000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131_073.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BADCHAR^@-@.~!

$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	THIS-FILE-NAME-WOULD-BE-PERFECTLY-LEGAL-IF-IT-WERE-NOT SO-LONG--IT-HAS-NEARLY-ONE-HUNDRED-SIXTY-CHARACTERS-WHICH-MAKES-IT-MUCH-TOOL-LONG-FOR-VMS.SO-THERE
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32_768
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32_767
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32_768
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-90_000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131_073.0
$MAX_DIGITS Maximum digits supported for floating-point types.	6
$MAX_IN_LEN Maximum input line length permitted by the implementation.	120
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2_147_483_648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..2 => '2:', 3..117 => '0', 118..120 => '11:'

$MAX_LEN_REAL_BASED_LITERAL A universal real based literal value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..3 => '16:', 4..116 => '0', 117..120 => 'F.E:' whose
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	1 => '"', 2..119 => 'A', 120 => '"'
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_ INT.	-2_147_483_648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	No_Such_Type
$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#FFFE#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 28 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A: A basic declaration (line 36) wrongly follows a later declaration.

E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.

C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.

C35502P: Equality operators in lines 62 & 69 should be inequality operators.

A35902C: Line 17's assignment of the nomimal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

C35A03E, & R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

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

C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

C37215C, E, G, H: Various discriminant constraints are wrongly expected to be incompatible with type CONS.

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

C41402A: 'STORAGE_SIZE is wrongly applied to an object of an access type.

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

C45614C: REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).

E66001D Pending further comment from AJPO. Test should be a B Test rather than an E Test.

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

BC3105A: Lines 159..168 are wrongly expected to be illegal; they are legal.

AD1A01A: The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.

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

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