DTIC FILE COPY

AVF Control Number :AVF-VSR-AFNOR-88-4

AD-A204 458

Ada* Compiler
VALIDATION SUMMARY REPORT:
Certificate Number: 880609A1.09101
ALSYS
AlsyCOMP_018, Version 3.21
VAX 11/750 VMS

Completion of On-Site Testing:
9 June 1988

Prepared By:
AFNOR
Tour Europe
Cedex 7
F-92080 Paris la Défense

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

DTIC
ELECTE
S 13 FEB 1989
D
E

This document has been approved for public release and sale; its distribution is unlimited.

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

89 2 13 114

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA204458

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Ada Compiler Validation Summary Report: ALSYS, AlsyCOMP 018, Version 3.21, VAX 11/750 (Host and Target).
(880609A1.09101)

5. TYPE OF REPORT & PERIOD COVERED
9 June 1988 to 9 June 1989

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

AFNOR,
Paris, France.

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS

AFNOR,
Paris, France.

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
9 June 1988

13. NUMBER OF PAGES
38 p.

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

AFNOR,
Paris, France.

15. SECURITY CLASS (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

UNCLASSIFIED

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)

AlsyCOMP_018, Version 3.21, ALSYS, AFNOR, VAX 11/750 under VMS, Version 4.5 (Host and Target), ACVC 1.9.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ada* Compiler Validation Summary Report:

Compiler Name: AlsyCOMP_018, Version 3.21

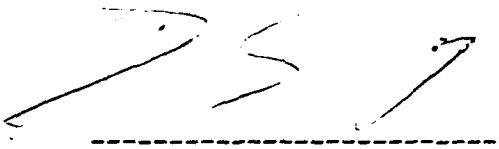
Certificate Number: 880609A1.09101

Host :	Target :
VAX 11/750 under	VAX 11/750 under
VMS	VMS
Version 4.5	Version 4.5

Testing Completed 9 June, 1988 Using ACVC 1.9

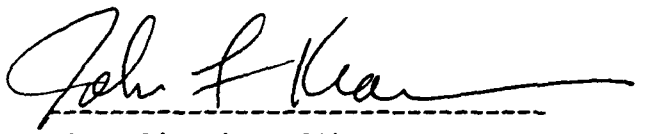
This report has been reviewed and is approved.

DTIC
COPY
INSPECTED
6



AFNOR
Dr Jacqueline Sidi
Tour Europe
Cedex 7
F-92080 Paris la Défense

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



Ada Validation Office
Dr.John F. Kramer
Institute for Defense Analyses
USA-Alexandria VA 22311

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
USA - Washington DC 20301

*Ada is a registred 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 the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

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

(KR)

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 under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was completed on 9 June, 1988 at Alsys at La Celle Saint-Cloud, France.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the 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)
USA - Washington DC 20301-3081

or from:

AFNOR
Tour Europe
Cedex 7
F-92080 Paris la Défense

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
USA - Alexandria VA 22311

1.3 REFERENCES

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

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

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

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

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

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

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

Host — The computer on which the compiler resides.

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

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

Target — The computer for which a compiler generates code.

Test — An Ada program that checks a compiler's conformity regarding a particular feature or 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 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 or 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 attemped.

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 are listed 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:	AlsyCOMP_018, Version 3.21
ACVC Version:	1.9
Certificate Number:	880609A1.09101
Host Computer:	
Machine:	VAX 11/750
Operating System:	VMS Version 4.5
Memory Size:	12 Mb
Target Computer:	
Machine:	VAX 11/750
Operating System:	VMS Version 4.5
Memory Size:	12 Mb
Communications Network:	none

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 tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests) and D29002K.)

- Universal integer calculations.

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

- Predefined types.

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

- Based literals.

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

- Expression evaluation.

  Apparently no default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

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

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

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

. Rounding

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

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

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

. Array types.

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

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

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

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

. Discriminated types.

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

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

. Aggregates.

In the evaluation of a multi-dimensional aggregate, 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, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

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

. Representation clauses

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

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

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

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are supported. (See tests C35508I..J, 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, EA3004C, EA3004D, CA3004E, and CA3004F.)

However the pragma INLINE is not supported for functions when they are called inside a package specification (see test EA3004D) or inside a task body (see test LA3004B).

- Input/output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO can 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 supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102G and CE2102K.)

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

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

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

More than one internal file can be associated with each external file for text I/O for reading only. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

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

More than one internal file can be associated with each external file for direct I/O for both reading and writing (See tests CE2107E..I (5 tests) and CE2111H.)

An internal sequential access file and an internal direct access file can be associated with a single external file for writing. (See test CE2107E.)

An external file associated with more than one internal file can be deleted for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO. (See test CE2110B.)

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

- 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 comprised 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 233 tests were inapplicable to this implementation. All inapplicable tests were processed during testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 30 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	109	1049	1627	17	15	45	2862
Inapplicable	1	2	226	0	3	1	233
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	190	499	549	248	166	98	140	326	135	36	234	3	238	2862
Inapplicable	14	73	125	0	0	0	3	1	2	0	0	0	15	233
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

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

B28003A	E28005C	C34004A	C35502P	A35902C	C35904A	C35904B
C35A03E	C35A03R	C37213H	C37213J	C37215C	C37215E	C37215G
C37215H	C38102C	C41402A	C45332A	C45614C	A74106C	C85018B
C87B04B	CC1331B	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, 233 tests were inapplicable for the reasons indicated:

- C35702A uses SHORT_FLOAT which is not supported by this implementation.
- A39005G uses a record representation clause which is not supported by this compiler.
- C45231D requires a macro substitution for any predefined numeric type other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT and LONG_FLOAT. This compiler does not support any such type.

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

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

- 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.

- C87B62B applies the attribute 'STORAGE_SIZE to an access type for which no STORAGE_SIZE length clause is given. In this case, STORAGE_ERROR is raised; the AVO ruled that this behavior is acceptable, since the interpretation of what value the attribute should return where no length clause is given is under review.

- B91001H contains an address clause which is rejected by the compiler.

- EA3004D and LA3004B require that errors be detected if pragma INLINE is supported for functions. But because this pragma has no effect when a function is called inside of a package specification or inside a task body, one of the intended errors is not detected. The AVO ruled that this is acceptable.

- EE2401D and EE2401G are inapplicable because USE_ERROR is raised when the CREATE of an instantiation of DIRECT_IO with unconstrained array type is called.

- CE2107C, CE2107D, CE2107H, CE2107I, CE2108A, CE2108C and CE3112A are inapplicable because no name is associated to a temporary file.

- CE3111B..E (4 tests), CE3114B and CE3115A are inapplicable because multiple internal files cannot be associated with the same external file for TEXT_IO, if any file is open for writting. The proper exception is raised when multiple access is attempted.

- The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

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

Modifications were required for 22 Class B tests, and 7 Class C tests.

The following 22 Class B tests were split because errors at one point resulted in the compiler not detecting other errors in the test:

B24007A	B24009A	B25002A	B26005A	B27005A	B32202A	B32202B
B32202C	B33001A	B36307A	B37004A	B74401F	B62001B	B74401R
B61012A	B91004A	B95004A	B95032A	B95069A	B95069B	BA1101B2
BA1101B4						

For the following tests, modification of the pass/fail criteria was needed. The AVO ruled that they are passed for the reason indicated :

- C34007A,D,G,M,P and S (6 tests) include a check that the STORAGE_SIZE attribute returns a value greater than 1 when applied to an access type for which no STORAGE_SIZE length clause has been provided; this implementation fails this check. However, the Ada Standard does not support the tests on this point, and the issue is under review. All other checks made by these tests were passed as expected.

- C4A012B checks that 0.0 raised to a negative value raises CONSTRAINT_ERROR. However NUMERIC_ERROR is also an acceptable exception to be raised. This implementation raises NUMERIC_ERROR.

- BA2001E requires that duplicate names of subunits with a common ancestor be detected and rejected at compile time. This implementation detects the error at link time, and the AVO ruled that this behavior is acceptable.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by AlsyCOMP_018, 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 AlsyCOMP_018 using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 11/750 operating under VMS, Version 4.5.

A tape containing all tests was taken on-site by the validation team for processing.Tests that make use of implementation-specific values were customized by ALSYS after loading of the tape.

The contents of the tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled, linked and all executable tests were run. Results were stored on tape after checking.

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

Option / Switch	Effect
REDUCTION=PARTIAL	Some High Level optimization performed
OBJECT=PEEPHOLE	Low Level optimization are performed
CALLS=INLINED	The pragma INLINE are taken into account
GENERIC=INLINE	Code of generic instantiation is placed inline in the same unit.

Tests were compiled, linked, and executed (as appropriate) using a single host computer. Test outputs, compilation listings, and job logs were captured on a 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 La Celle Saint-Cloud, France and was completed on 9 June 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

ALSYS has submitted the following conformance statement concerning the AlsyCOMP_018.

DECLARATION OF CONFORMANCE

Compiler Implementor: ALSYS

Ada* Validation Facility:
AFNOR, Tour Europe, Cedex 7, F-92080 Paris la Défense

Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name: AlsyCOMP_018, Version: Version 3.21

Host Architecture ISA: VAX 11/750 OS&VER #: VMS, Version 4.5

Target Architecture ISA: VAX 11/750 OS&VER #: VMS, Version 4.5

Implementor's declaration

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

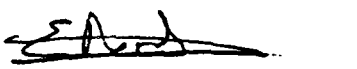
Date:_____

ALSYS
Etienne Morel, Managing Director

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

Owner's Declaration

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



Date: 27 SEPT 1990

ALSYS
Etienne Morel, Managing Director

*Ada is a registred 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 the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the AlsyCOMP_018, Version 3.21, 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 SHORT_INTEGER is range -(2**7) .. (2**7-1);
 type INTEGER       is range -(2**15) .. (2**15-1);
 type LONG_INTEGER  is range -(2**31) .. (2**31-1);


 type FLOAT is digits 6 range
 -(2.0-2.0**(-23)) * 2.0**127 .. +(2.0-2.0**(-23)) * 2.0**127;


 type LONG_FLOAT is digits 15 range
 -(2.0-2.0**(-51)) * 2.0**1023 .. +(2.0-2.0**(-51)) * 2.0**1023;


 type DURATION is delta 2.0**(-14) range -86_400.0 .. 86_400.0;
 -- DURATION'SMALL = 2.0**(-14).

 ...

end STANDARD ;
```

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

APPENDIX F

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 IMPLEMENTATION-DEPENDENT PRAGMAS

In this part of the Appendix, the implementation-dependent aspects of the use of **pragma** are described.

1.1 Interfacing the Language Ada with Other Languages

Programs written in Ada can interface with external subprograms written in another language, by use of the INTERFACE pragma. The format of the pragma is:

pragma INTERFACE (*language_name* , *Ada_subprogram_name*) ;

where the *language_name* can be

* ASSEMBLER

To allow the use of non-Ada naming conventions, such as special characters, or case sensitivity, an implementation-dependent pragma INTERFACE_NAME has been introduced:

pragma INTERFACE_NAME (*Ada_subprogram_name*, *name_string*) ;

The pragma INTERFACE_NAME may be used anywhere in an Ada program where INTERFACE is allowed (see [13.9]). INTERFACE_NAME must occur after the corresponding pragma INTERFACE and within the same declarative part.

1.3 Pragma Indent

This pragma is only used with the Alsys *Reformatter*; this tool offers the functionalities of a pretty-printer in an Ada environment.

The pragma is placed in the source file and interpreted by the Reformatter.

pragma INDENT(OFF)

The Reformatter does not modify the source lines after the pragma.

pragma INDENT(ON)

The Reformatter resumes its action after the pragma.

1.4 Pragmas not Implemented

The following pragmas are not implemented:

CONTROLLED
MEMORY_SIZE
OPTIMIZE
PACK
SHARED
STORAGE_UNIT
SYSTEM_NAME

F.2 IMPLEMENTATION-DEPENDENT ATTRIBUTES

In addition to the Representation Attributes of [13.7.2] and [13.7.3], there are four attributes which are listed under F.5 below, for use in record representation clauses.

Limitations on the use of the attribute ADDRESS

The attribute ADDRESS is implemented for all prefixes that have meaningful addresses. The following entities do not have meaningful addresses and will therefore cause a compilation error if used as prefix to ADDRESS:

- A constant that is implemented as an immediate value i.e., does not have any space allocated for it.
- A package specification that is not a library unit.
- A package body that is not a library unit or a subunit.

F.3 PACKAGES SYSTEM AND STANDARD

```
package SYSTEM is

-- Standard Ada definitions

   type NAME is (VMS) ;
   SYSTEM_NAME      :      constant NAME := VMS;
   STORAGE_UNIT     :      constant := 8 ;
   MEMORY_SIZE      :      constant := 2**32 ;
   MIN_INT          :      constant := -(2**31) ;
   MAX_INT          :      constant := 2**31-1 ;
   MAX_DIGITS       :      constant := 15 ;
   MAX_MANTISSA     :      constant := 31 ;
   FINE_DELTA       :      constant := 2#1.0#e-31 ;
   TICK             :      constant := 1.0 ;

   type ADDRESS is private ;
   NULL_ADDRESS : constant ADDRESS ;
```

```
    subtype PRIORITY is INTEGER range 1..127 ;

-- Address arithmetic

    function TO_LONG_INTEGER (LEFT : ADDRESS)
        return LONG_INTEGER ;
    function TO_ADDRESS (LEFT : LONG_INTEGER)
        return ADDRESS ;

    function "+" (LEFT : LONG_INTEGER ; RIGHT : ADDRESS)
        return ADDRESS ;
    function "+" (LEFT : ADDRESS ; RIGHT : LONG_INTEGER)
        return ADDRESS ;

    function "-" (LEFT : ADDRESS ; RIGHT : ADDRESS)
        return LONG_INTEGER ;
    function "-" (LEFT : ADDRESS ; RIGHT : LONG_INTEGER)
        return ADDRESS ;

    function "mod" (LEFT : ADDRESS ; RIGHT : POSITIVE)
        return NATURAL ;

    function "<"  (LEFT : ADDRESS ; RIGHT : ADDRESS)
        return BOOLEAN ;
    function "<=" (LEFT : ADDRESS ; RIGHT : ADDRESS)
        return BOOLEAN ;
    function ">"  (LEFT : ADDRESS ; RIGHT : ADDRESS)
        return BOOLEAN ;
    function ">=" (LEFT : ADDRESS ; RIGHT : ADDRESS)
        return BOOLEAN ;

    function IS_NULL (LEFT : ADDRESS)
        return BOOLEAN ;

    function WORD_ALIGNED (LEFT : ADDRESS)
        return BOOLEAN ;

    function ROUND (LEFT : ADDRESS)
        return ADDRESS ;
    -- Return the given address rounded to the next lower even value

    procedure COPY (FROM: ADDRESS ;TO: ADDRESS ;SIZE: NATURAL) ;
    -- Copy SIZE storage units. The result is undefined if the two areas
    -- overlap.

-- Direct memory access

    generic
        type ELEMENT_TYPE is private ;
    function FETCH (FROM : ADDRESS) return ELEMENT_TYPE ;
    -- Return the bit pattern stored at address FROM, as a value of the
    -- specified ELEMENT_TYPE. This function is not implemented
    -- for unconstrained array types.
```

```
    generic
        type ELEMENT_TYPE is private ;
    procedure STORE (INTO : ADDRESS ; OBJECT : ELEMENT_TYPE) ;
    -- Store the bit pattern representing the value of OBJECT, at
    -- address INTO. This function is not implemented for
    -- unconstrained array types.

    private

    -- private part of the compiler

end SYSTEM ;
```

The package STANDARD

The following are the implementation-dependent aspects of the package STANDARD:

```
type SHORT_INTEGER is range -(2**7) .. (2**7 -1);
type INTEGER is range -(2**15) .. (2**15 -1);
type LONG_INTEGER is range -(2**31) .. (2**31 -1);

type FLOAT is digits 6 range
    -(2.0 - 2.0**(-23)) * 2.0**127 ..
    +(2.0 - 2.0**(-23)) * 2.0**127 ;

type LONG_FLOAT is digits 15 range
    -(2.0 - 2.0**(-51)) * 2.0**1023 ..
    +(2.0 - 2.0**(-51)) * 2.0**1023;

type DURATION is delta 2.0**(-14) range -86_400.0 .. 86_400.0;
```

F.4 RESTRICTIONS ON REPRESENTATION CLAUSES

The facilities covered in [13] are provided, except for the following features:

* There is no bit implementation for any of the representation clauses.
* Address clauses are not implemented.
* Change of representation for RECORD type is not implemented.
* Machine code insertions are not implemented.
* For the length clause :
    - Size specification: T'SIZE is not implemented for types declared in a generic unit.
    - Specification of storage for a task activation: T'STORAGE_SIZE is not implemented when T is a task type.

- Specification of *small* for a fixed point type: T'SMALL is restricted to a power of 2, and the absolute value of the exponent must be less than 31.

* The Enumeration Clause is not allowed if there is a range constraint on the parent subtype.

* The Record Clause is not allowed for a derived record type.

F.5 IMPLEMENTATION-GENERATED NAMES

There are four such attributes with corresponding generated names:

T'RECORD_SIZE For a prefix T that denotes a record type. This attribute refers to the record component introduced by the compiler in a record to store the size of the record object. This component exists for objects of a record type with defaulted discriminants when the sizes of the record objects depend on the values of the discriminants.

T'VARIANT_INDEX For a prefix T that denotes a record type. This attribute refers to the record component introduced by the compiler in a record to assist in the efficient implementation of discriminant checks. This component exists for objects of a record type with variant part.

C'ARRAY_DESCRIPTOR For a prefix C that denotes a record component of array type whose component subtype definition depends on discriminants. This attribute refers to the record component introduced by the compiler in a record to store information on subtypes of components that depend on discriminants.

C'RECORD_DESCRIPTOR For a prefix C that denotes a record component of record type whose component subtype definition depends on discriminants. This attribute refers to the record component introduced by the compiler in a record to store information on subtypes of components that depend on discriminants.

F.6 ADDRESS CLAUSES

Address clauses [13.5] are not implemented in this version of the Alsys Ada Compiler.

F.7 UNCHECKED CONVERSIONS

Unconstrained arrays are not allowed as target types. Unconstrained record types without defaulted discriminants are not allowed as target types.

If the source and the target types are each scalar or access types, the sizes of the objects of the source and target types must be equal.

If a composite type is used either as source type or as target type this restriction on the size does not apply.

If the source and the target types are each of scalar or access type or if they are both of composite type, the effect of the function is to return the operand.

In other cases the effect of unchecked conversion can be considered as a copy:

- if an unchecked conversion is achieved of a scalar or access source type to a composite target type, the result of the function is a copy of the source operand: the result has the size of the source.

- if an unchecked conversion is achieved of a composite source type to a scalar or access target type, the result of the function is a copy of the source operand: the result has the size of the target.

F.8 INPUT-OUTPUT CHARACTERISTICS

In this part of the Appendix the implementation-specific aspects of the input-output system are described.

8.1 Introduction

In Ada, input-output operations (IO) are considered to be performed on *objects* of a certain file type rather than being performed directly on external files. An *external file* is anything external to the program that can produce a value to be read or receive a value to be written. Values transferred for a given file must be all of one type.

Generally, in Ada documentation, the term *file* refers to an object of a certain file type, whereas a physical manifestation is known as an *external file*. An external file is characterized by

* Its NAME, which is a string defining a legal path name under the current version of the operating system.

* Its FORM, which gives implementation-dependent information on file characteristics.

Both the NAME and FORM appear explicitly in the Ada CREATE and OPEN procedures. Though a file is an object of a certain file type, ultimately the object has to correspond to an external file. Both CREATE and OPEN associate a NAME of an external file (of a certain FORM) with a program file object.

Ada IO operations are provided by means of standard packages [14].

SEQUENTIAL_IO A generic package for sequential files of a single element type.

DIRECT_IO	A generic package for direct (random) access files.
TEXT_IO	A generic package for human-readable (text, ASCII) files.
IO_EXCEPTIONS	A package which defines the exceptions needed by the above three packages.

The generic package LOW_LEVEL_IO is not implemented in this version.

The upper bound for index values in DIRECT_IO and for line, column and page numbers in TEXT_IO is given by

```
COUNT'LAST = 2**31 -1
```

The upper bound for fields widths in TEXT_IO is given by

```
FIELD'LAST = 255
```

8.2 The FORM Parameter

The FORM parameter to both the CREATE and OPEN procedures in Ada specifies the characteristics of the external file involved.

The CREATE procedure establishes a new external file, of a given NAME and FORM, and associates it with a specified program FILE object. The external file is created (and the FILE object set) with a certain file MODE. If the external file already exists, the file will be erased. The exception USE_ERROR is raised if the file mode is IN_FILE.

The OPEN procedure associates an existing external file, of a given NAME and FORM, with a specified program FILE object. The procedure also sets the current FILE mode. If there is an inadmissible change of MODE, then an Ada USE_ERROR is generated.

The general form of any attribute is a keyword followed by => and then a qualifier. The qualifier may sometimes be omitted. The format for an attribute specifier is thus either of

KEYWORD

KEYWORD => QUALIFIER

We will discuss each attribute in order.

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.	'X234567890' & (24 * '1234567890') & '12341'
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	'X234567890' & (24 * '1234567890') & '12342'
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	'X234567890' & (11 * '1234567890') & '12345xx3xx12345' & (12 * '1234567890')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	'X234567890' & (11 * '1234567890') & '12345xx4xx12345' & (12 * '1234567890')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(252 * '0') & '298'

Name and Meaning	Value
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(250 * '0') & '690.0'
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	'X234567890' & (11 * '1234567890')
$BIG_STRING2 A string literal which when catenated with the end of BIG_STRING1 yields the image of BIG_ID1.	(13 * '1234567890') &'12341'
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(235 * ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	255
$FILE_NAME_WITH_BAD_CHAR An external file name that either contains invalid characters or is too long.	/"/"
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	/*/*
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100_000.0

Name and Meaning	Value
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	100_000_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	/~/*/f1
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long (or illegal).	/*/~/f2
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32768
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-100_000_000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	255

Name and Meaning	Value
$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 whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	'2#' & (250 * '0') & '11#'
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	'16:' & (248 * '0') & 'F.E:'
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(25 * '1234567890') & '123'
$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#FFFFFFFE#

APPENDIX D

WITHDRAWN TESTS

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

B28003A: A basic declaration (line 36) 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 ALMP.

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 nominal 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.INDENT_INT has an argument of the wrong type (LONG_INTEGER).

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 incorrect; they are correct.

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 thar 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.