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: DDC-I, Inc., DCS-68020/SUN, Version 4.2 (1.0), SUN-3/50 Workstation Host and Target

5. TYPE OF REPORT & PERIOD COVERED
28 July 1988 to 28 July 1988

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
NBS
Gaithersburg, MD

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS
NBS
Gaithersburg, MD

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

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office)
NBS
Gaithersburg, MD

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)
DACS-68020/SUN, Version 4.2 (1.0), DDC-I, Inc., NBS, Gaithersburg, MD, SUN-3/50 Workstation under SunOS UNIX, Version 4.2 Rel. 3.4EXPORT (Host) to SUN-3/50 Workstation under SunOS UNIX, Version 4.2 Rel. 3.4EXPORT (Target), ACVC 1.9.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE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

Ada Compiler Validation Summary Report:

Compiler Name: DACS-68020/SUN, Version 4.2 (1.0)

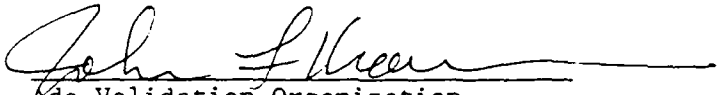
Certificate Number: 880728S1.09142

Host:	Target:
SUN-3/50 Workstation	SUN-3/50 Workstation
under SunOS UNIX	under SunOS UNIX
Version 4.2 Rel. 3.4EXPORT	Version 4.2 Rel. 3.4EXPORT

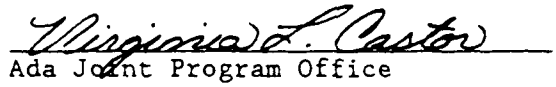
Testing Completed 28 July 1988 Using ACVC 1.9

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
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

AVF Control Number: NBS88VDDC545_3

Ada Compiler
VALIDATION SUMMARY REPORT:
Certificate Number: 880728S1.09142
DDC-I, Inc.
DACS-68020/SUN, Version 4.2 (1.0)
SUN-3/50 Workstation

Completion of On-Site Testing:
28 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

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-2
1.2 USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3 REFERENCES 1-3
1.4 DEFINITION OF TERMS 1-3
1.5 ACVC TEST CLASSES 1-4

CHAPTER 2 CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED 2-1
2.2 IMPLEMENTATION CHARACTERISTICS 2-2

CHAPTER 3 TEST INFORMATION

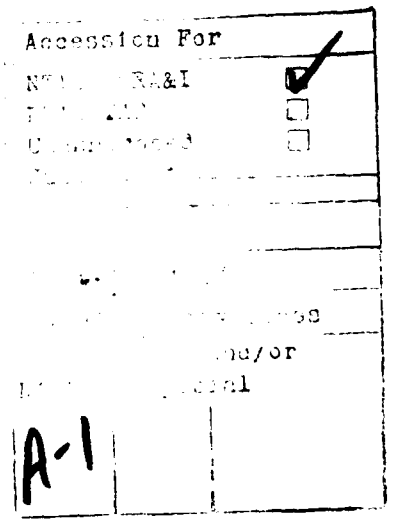
3.1 TEST RESULTS 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER 3-2
3.4 WITHDRAWN TESTS 3-2
3.5 INAPPLICABLE TESTS 3-2
3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS . . 3-5
3.7 ADDITIONAL TESTING INFORMATION 3-6
3.7.1 Prevalidation 3-6
3.7.2 Test Method 3-6
3.7.3 Test Site 3-7

APPENDIX A CONFORMANCE STATEMENT

APPENDIX B APPENDIX F OF THE Ada STANDARD

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS



CHAPTER 1

INTRODUCTION

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

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

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

On-site testing was completed 28 July 1988 at Copenhagen, Denmark.

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	The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and Panel 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: DACS-68020/SUN, Version 4.2 (1.0)

ACVC Version: 1.9

Certificate Number: 880728S1.09142

Host Computer:

Machine:	SUN-3/50 Workstation
Operating System:	SunOS UNIX Version 4.2 Rel. 3.4EXPORT
Memory Size:	4 MBytes

Target Computer:

Machine:	SUN-3/50 Workstation
Operating System:	SunOS UNIX Version 4.2 Rel. 3.4EXPORT
Memory Size:	4 MBytes

Communications Network: Ethernet LAN

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 and recursive procedures separately compiled as subunits nested to 17 levels. It does process 65 levels of block nesting. 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 types SHORT_INTEGER, and LONG_FLOAT in the package STANDARD. (See tests B86001BC, B86001CS 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 all 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 to even. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round to even. (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 packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array objects are declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression 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.)

- 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 subtype. (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 not supported, then the implementation must reject it.

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

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

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are not supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See test A39005C.)

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

Length clauses with SMALL specifications are not 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 not 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 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 record types with discriminants without defaults. (See tests AE2101H and

EE2401G.)

The package DIRECT_IO cannot be instantiated with unconstrained array types. (See test EE2401D.)

There are no strings which are illegal external file names for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102C and CE2102H.)

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, cannot be created in OUT_FILE mode, and cannot be created in IN_FILE mode. (See test EE3102C.)

Only one internal file may be associated with the same external file if only one file is opened for writing. (See test CE3111B.)

More than one internal file can be associated with each external file for text I/O for reading only or for writing only. (See tests CE3111A, CE3111C..E (3 tests), CE3114B, CE3115A, CE2110B, and CE2111D.)

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), CE2110B, and CE2111D.)

More than one internal file can be associated with each external file for direct and sequential I/O for writing only. (See test CE2107E.)

More than one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107F..I (4 tests), and CE2110B.)

More than one internal file cannot be associated with each external file for direct I/O for both reading and writing. (See test CE2111H.)

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

- Generics.

Generic subprogram declarations and bodies cannot compiled in separate compilations. (See test CA2009F.)

Generic subprogram declarations and bodies can compiled in separate compilations. (See test CA1012A.)

Generic package declarations and bodies can be compiled in separate compilations so long as no instantiations of those units precede the bodies. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected. (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 250 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing. Modifications to the code, processing, or grading for 69 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	107	1046	1612	17	16	46	2844
Inapplicable	3	5	241	0	1	0	250
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	187	493	538	245	165	98	139	326	135	36	232	3	247	2844
Inapplicable	17	79	136	3	0	0	4	1	2	0	2	0	6	250
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, 250 test were inapplicable for the reasons indicated:

C24113I..K (3 tests) were rejected because they contain declarations that exceed MAX_IN_LEN (126 characters)

C35508I..J (2 tests) and C35508M..N (2 tests) use enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1). These clauses are not supported by this compiler.

C35702A uses SHORT_FLOAT which is not supported by this implementation. A39005B and C87B62A use length clauses with SIZE specifications for derived integer types or for enumeration types which are not supported by this compiler.

A39005E and C87B62C use length clauses with SMALL specifications which are not supported by this implementation.

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

The following (13) tests use LONG_INTEGER, which is not supported by this compiler.

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45631C	C45632C
B52004D	C55B07A	B55B09C		

C45231D checks that relational and membership operations yield correct results for predefined types. This implementation supports only INTEGER, SHORT_INTEGER, FLOAT, and LONG_FLOAT.

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.

C4A013B uses a static value that is outside the range of the most accurate floating point base type. The declaration was rejected at compile time.

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.

C87B62B this test contains the application of the attribute STORAGE_SIZE to access types for which no corresponding STORAGE_SIZE length clause has been provided; this compiler rejects such an application. The AVO accepted this behavior because the Ada standard is not clear on how such a situation should be treated; the matter will be discussed by the language maintenance body.

C96005B requires the range of type DURATION to be different from those of its base type; in this implementation they are the same.

CA2009F compiles generic subprogram declarations and bodies in separate compilations; the compilation occurs following a compilation that contains instantiations of those units. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected.

CA2009C, BC3204C, and BC3205D compile generic package specifications and bodies in separate compilations. This compiler requires that generic package specifications and bodies be in a single compilation.

CE2105A checks that CREATE is permitted for an IN_FILE for SEQUENTIAL_IO. An implementation can raise USE_ERROR or NAME_ERROR (see AI-00332). This implementation raises USE_ERROR.

CE2105B checks that CREATE is permitted for an IN_FILE for DIRECT_IO. An implementation can raise USE_ERROR or NAME_ERROR (see AI-00332). This implementation raises USE_ERROR.

CE2111H checks whether resetting an internal file has any effect upon other internal files accessing the same external file for DIRECT_IO. An implementation can raise USE_ERROR or NAME_ERROR (see AI-00332). This implementation raises USE_ERROR.

CE3109A checks that CREATE is permitted for an IN_FILE. An implementation can raise USE_ERROR or NAME_ERROR (see AI-00332). This implementation raises USE_ERROR.

CE3111B checks whether more than one internal file may be associated with the same external file if only one file is opened for writing. An implementation can raise USE_ERROR or NAME_ERROR (see AI-00332). This implementation allows an external file to be simultaneously opened for reading and writing. However, the exception END_ERROR is raised. The reason for END_ERROR being raised is that this implementation allows more than one internal file to be associated with the same external file while one is OPENed in IN_FILE mode and the other is OPENed in OUT_FILE mode with the output being directed to a buffer. When the file is read, the END_ERROR exception is raised because the output has not been flushed to the external file; if the buffer were small enough or the output large enough then the output would be flushed to the external file, but this is problematic. Ada does not define the physical status of a file until that file has been closed.

EE2401D uses instantiations of package DIRECT_IO with unconstrained array types. These instantiations are rejected by this compiler.

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 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 69 Class B tests.

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

B22003A	B26001A	B26002A	B26005A
B28001D	B29001A	B2A003A	B2A003B
B2A003C	B33301A	B35101A	B37106A
B37301B	B37302A	B38003A	B38003B
B38009A	B38009B	B51001A	B53009A
B54A01C	B55A01A	B61001C	B61001D
B61001F	B61001H	B61001I	B61001M
B61001R	B61001S	B61001W	B67001A
B67001C	B67001D	B91001A	B91002A
B91002B	B91002C	B91002D	B91002E
B91002F	B91002G	B91002H	B91002I
B91002J	B91002K	B91002L	B95030A
B95061A	B95061F	B95061G	B95077A
B97101A	B97101E	B97102A	B97103E
B97104G	BA1101B0M	BA1101B1	BA1101B2
BA1101B3	BA1101B4	BC1109A	BC1109C
BC1109D	BC1202A	BC1202B	BC1202E
BC1202F	BC1202G	BC2001D	BC2001E
BC3204D			

C34007A, C34007D, C34007G, C34007M, C34007P, C34007S these tests contain the application of the attribute STORAGE_SIZE to access types for which no corresponding STORAGE_SIZE length clause has been provided; this compiler rejects such an application. The AVO accepted this behavior because the Ada standard is not clear on how such a situation should be treated; the matter will be discussed by the language maintenance body. For this implementation, the lines within each of these tests which have the STORAGE_SIZE length clause were changed to comment lines under the direction of the AVF Manager. These modified tests ran to a successful completion and produced a PASSED message.

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 DACS-68020/SUN 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 DACS-68020/SUN using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX-11/8530 computer (20 Mbytes RAM; 4 456 Mbytes disk drives; 24 terminal ports; 1 1600/6250 bpi tape-drive) operating under VMX Version 4.5 onto which was loaded the magnetic tape and the SUN-3/50 Workstation operating under SunOS UNIX Version 4.2 Rel. 3.4EXPORT which served as the host/target computer. The VAX-11/8530 was linked via an Ethernet LAN to a SUN-3/50 Workstation.

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. The contents of the magnetic tape were loaded directly onto the VAX-11/8530 and then downloaded via an Ethernet LAN to the SUN-3/50 Workstation.

After the test files were loaded to disk on the SUN-3/50 Workstation, the full set of tests was compiled, and all executable tests were linked and executed on the SUN-3/50 Workstation. Results were transferred from the SUN-3/50 Workstation via the Ethernet LAN to the VAX-11/8530. The results were then printed from the VAX-11/8530 computer.

The compiler was tested using command scripts provided by DDC-I, Inc. and reviewed by the validation team. The compiler was tested using all default option | switch settings except for the following:

Option | Switch	Effect.
-L	List option
-a	Name of the program library; both the default values and explicitly specified program libraries were used.

Tests were compiled, linked, and executed using the same Host/Target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF.

3.7.3 Test Site

Testing was conducted at Copenhagen, Denmark and was completed on 28 July 1988.

APPENDIX A

CONFORMANCE STATEMENT

APPENDIX A

CONFORMANCE STATEMENT

DDC International A/S has submitted the following Declaration of Conformance statement concerning the DACS-68020/SUN.

DECLARATION OF CONFORMANCE

Compiler Implementor: DDC International A/S
Ada Validation Facility:

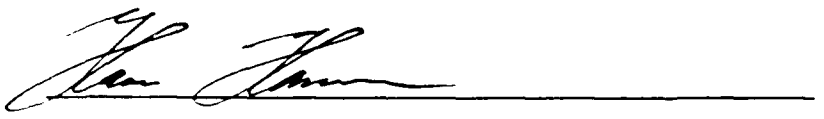
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

Base Configuration

Base Compiler Name: DACS-68020/SUN, Version: 4.2 (1.0)
Host Architecture ISA: SUN-3/50
OS&VER #: SunOS UNIX 4.2 Rel 3.4EXPORT
Target Architecture ISA: SUN-3/50
OS&VER #: SunOS UNIX 4.2 Rel 3.4EXPORT

Implementor's Declaration

I, the undersigned, representing DDC International A/S, 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 DDC International A/S 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 name.

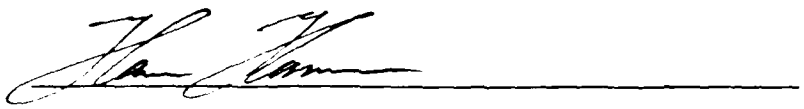


Date: 27 July 1988

DDC International A/S
Hasse Hansson, Department Manager, Product Development

Owner's Declaration

I, the undersigned, representing DDC International A/S, 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, a 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 July 1988

DDC International A/S
Hasse Hansson, Department Manager, Product Development

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the DACS-SUN-3/50 Workstation, Version 4.2 (1.0), are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).. Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is


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

   type FLOAT is digits 6 range
                     -1.70141102E+38 .. 1.70141102E+38;
   type LONG_FLOAT is digits  15 range
                -1.7976931348623157E308 .. 1.7976931348623157E308;
   type DURATION is delta 2#1.0#E-14 range -131_072.0 .. 131_071.0;
```

DDC-I Ada Compiler System® for Sun-3/SunOS®

User's Guide

9 August 1988

Author: Dan Ole Johansen,
Peter Villadsen

Document No.: DDC-I 5590/RPT/3, issue 2

F Appendix F of the Ada Reference Manual

F.0 Introduction

This appendix describes the implementation-dependent characteristics of the DDC-I Sun-3/SunOS V Ada Compiler, as required in the Appendix F frame of the Ada Reference Manual (ANSI/MIL-STD 1815A).

F.1 Implementation-Dependent Pragmas

There is one implementation-defined pragma: Interface_spelling, see section 5.6.6.2.

F.2 Implementation-Dependent Attributes

No implementation-dependent attributes are defined.

F.3 Package SYSTEM

The specification of the package SYSTEM:

```
package SYSTEM is

   type ADDRESS          is access INTEGER;
   subtype PRIORITY      is INTEGER range 0..15;
   type NAME             is (SUN_3_SUNOS);
   SYSTEM_NAME:          constant NAME  := SUN_3_SUNOS;
   STORAGE_UNIT:         constant       := 8;
   MEMORY_SIZE:          constant       := 2**32;
   MIN_INT:              constant       := -2_147_483_648;
   MAX_INT:              constant       := 2_147_483_647;
   MAX_DIGITS:           constant       := 15;
   MAX_MANTISSA:         constant       := 31;

   FINE_DELTA:           constant       := 2#1.0#E-31;
   TICK:                 constant       := 1.0/15.0E+6;

   type interface_language is (C, AS);

end SYSTEM;
```

F.4 Representation Clauses

In general, no representation clauses may be given for a derived type. The representation clauses that are accepted for non-derived types are described in the following:

Length Clause

The compiler only accepts length clauses that specify the number of storage units to be reserved for a collection and the number of storages units to be reserved for a task.

Enumeration Representation Clause

Enumeration representation clauses may specify representations only in the range of the predefined type INTEGER.

Record Representation Clause

A component clause is allowed if and only if

- the component type is a discrete type different from INTEGER
- the component type is an array type with a discrete element type different from INTEGER

No component clause is allowed if the component type is not covered by the above two inclusions. If the record type contains components not covered by a component clause, they are allocated consecutively after the component with the value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

F.5 Implementation-Dependent Names for Implementation-Dependent Components

None defined by the compiler.

F.6 Address Clauses

Not supported by the compiler.

F.7 Unchecked Conversion

Unchecked conversion is only allowed between objects of the same "size". In this context the "size" of an array is equal to that of two access values and the "size" of a packed array is equal to two access values and an integer. This is the only restriction imposed on unchecked conversion.

F.8 Input-Output Packages

The implementation supports all requirements of the Ada language. It is an effective interface to the UNIX file system, and in the case of text input-output also an effective interface to the UNIX standard input, standard output and standard error streams.

This section describes the functional aspects of the interface to the UNIX file system, including the means by which the various file control facilities are made available to the Ada programmer.

The Ada input-output concept as defined in Chapter 14 of the ARM does not constitute a complete functional specification of the input-output packages. Some aspects are not discussed at all, while others are deliberately left open to an implementation. These gaps are filled by this section.

The reader should be familiar with

[DoD 83] - The Ada Language definition

and some sections require that the reader is familiar with

[UNIX 3] - UNIX Programmer Reference Manual

F.8.1 External Files

External files can be on disc, tape, or be a character device (a line printer, terminal etc.).

Files on disc exist after the execution of the program unless given an empty NAME parameter.

The implementation will raise USE_ERROR when an operation is inappropriate for the physical device. In particular the concept of a page or end-of-file or file size are not considered to be applicable to terminal devices and attempted use of operations involving these concepts will raise USE_ERROR.

Creation of files associated with existing external files will raise USE_ERROR.

Deletion is not allowed on non-disc devices and requires write access.

Creation of files with mode IN_FILE will raise USE_ERORR.

F.8.2 File Management

This subsection contains information regarding file management:

- restriction on sequential and direct input-output,
- the NAME parameter,
- the FORM parameter,
- file access.

F.8.2.1 Restrictions on Sequential and Direct Input-Output

The only restriction is that placed on the element size, i.e. the number of bytes occupied by the ELEMENT_TYPE: the maximum size allowed is 32k bytes; and if the size of the type is variable, the maximum size must be determinable at the point of instantiation from the value of the SIZE attribute for the element type.

F.8.2.2 The NAME Parameter

The NAME parameter when non-empty must be valid UNIX path name. Access denial to any directory in the path name will raise USE_ERROR.

The UNIX names "stdin", "stdout", and "stderr" can be used in conjunction with TEXT_IO.open. No physical opening of the external file is performed and the Ada file will be associated with the already open external file.

Temporary files (NAME = "") are created using tmpname (3) and will be deleted on closure. Abnormal program termination may leave temporary files in existence.

Default naming conventions and version numbers are not applicable to UNIX.

F.8.2.3 The FORM Parameter

The FORM parameter has the following facilities:

a) Opening a FIFO special file using open(2) system call. This is achieved by the string "FIFO". If this facility is used with CREATE, the exception USE_ERROR will be raised. This facility is not available for direct_io or text_io and raises USE_ERROR.

   The default for this facility is indicated by the "ORDINARY" string designating the creation of an ordinary file. If this string is used with OPEN and the external file is of type FIFO special, the operation raises USE_ERROR.

   The O_NDELAY flag associated with FIFO specials (see open(2)) can be modified using an additional string after the "FIFO" string. The strings "O_NDELAY=ON" and "O_NDELAY=OFF" set the flag on and off respectively. The default is "O_NDELAY=OFF". Thus "FIFO O_NDELAY=ON" opens a FIFO special file and set the O_NDELAY flag on.

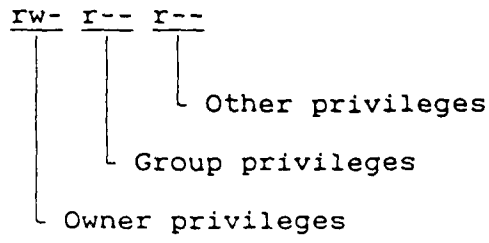
b) The use of the string "APPEND" with text-files prevents the emptying of the file for the OPEN operation. The presence of "APPEND" in the form parameter is only applicable to OPEN, and its use in CREATE will raise USE_ERROR. The string "NOAPPEND" signifies the default.

   The opened file will be treated by the routines delete as if empty.

   Opening direct and sequential files with the "APPEND" or "NOAPPEND" raises USE_ERROR.

c) The changing of default access rights by specifying the mode parameter used in the open(2) system call used to implement the Ada CREATE procedure. This is achieved by use of the string "MODE=<mode>" where <mode> is an octal, decimal or hexadecimal integer in the standard UNIX format. Only the nine least significant bits of the creation mask are used. This facility is also used by OPEN to change the access permissions by means of the chmod(2) system call.

The default for mode is 0644, allowing the owner to read and write, and the group and others to read. The bits mean (as in standard UNIX):



NOTES:

The options are delimited by commas.

If more than one of the three option types is included, the rightmost option is selected.

Blanks are not significant in any part of the string.

The FORM parameter provides all default options as required in the ARM.

F.8.2.3.1 Syntax of the Form Parameter

```
<form_parameter> := [<option> [,<option> [,<option>] ] ]

<option> := <access_rights> | <fifo_option> | <append_option>

<access_rights> := MODE= <mode>

<fifo_option> := <fifo_special> | ORDINARY

<append_option> := APPEND | NOAPPEND

<mode> := <hex_number> | <octal_number> | <decimal_number>

<fifo_special> := FIFO [<o_ndelay_parameter>]

<o_ndelay_parameter> := O_NDELAY=ON | O_NDELAY=OFF

<decimal_number> := <decimal_digit> {<decimal_digit>}

<hex_number>     := 0 x <hex_suffix>

<octal_number>   := 0 <octal_suffix>

<hex_suffix>     := <hex_digit> {<hex_digit>}

<octal_suffix>   := <octal_digit> {<octal_digit>}

<hex_digit>      := 0 | 1 ...| 9 | A | ... | F | a | ...| f

<decimal_digit>  := 0 | 1 ...| 9

<octal_digit>    := 0 | 1 ...| 7
```

F.8.2.4 File Access

Any number of files in an Ada program may be associated with any external file at any time. Each end of a FIFO special file must be accessed from two UNIX processes which will have to correspond to two Ada programs.

It is the responsibility of the programmer to consider the effects of file sharing between programs.

The RESET and OPEN operations to OUT_FILE mode empty the file in SEQUENTIAL_IO and TEXT_IO.

Interchanging between SEQUENTIAL_IO and DIRECT_IO for files of the same object types can be achieved without taking special measures.

The state of the external file at any moment is in general undefined. Closing and resetting a file will, however, flush any buffering in the input-output packages. Unpredictable results may occur if the program is terminated without calling CLOSE.

F.8.3 Sequential Input-Output

The implementation omits type checking for DATA_ERROR, in case the element type is of an unconstrained type, ARM 14.2.2(4), i.e.:

```
... f : FILE_TYPE
type et  is 1..100;
type eat is array( et range <> ) of integer;

X : eat( 1..2 );
Y : eat( 1..4 );
...
-- write X, Y:

write( f, X); write( f, Y); reset( f, IN_FILE);

-- read X into Y and Y into X:

read( f, Y); read( f, X);
```

This will give undefined values in the last 2 elements of Y, and not DATA_ERROR.

F.8.3.1 Specification of the Package Sequential IO

```
with BASIC_IO_TYPES;

with IO_EXCEPTIONS;

generic

   type ELEMENT_TYPE is private;

package SEQUENTIAL_IO is

  type FILE_TYPE is limited private;

  type FILE_MODE is (IN_FILE, OUT_FILE);

-- File management

   procedure CREATE(FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE := OUT_FILE;
                    NAME : in     STRING    := "";
                    FORM : in     STRING    := "");

   procedure OPEN  (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE;
                    NAME : in     STRING;
                    FORM : in     STRING := "");

   procedure CLOSE (FILE : in out FILE_TYPE);

   procedure DELETE(FILE : in out FILE_TYPE);

   procedure RESET (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE);

   procedure RESET (FILE : in out FILE_TYPE);

   function MODE   (FILE : in FILE_TYPE) return FILE_MODE;

   function NAME   (FILE : in FILE_TYPE) return STRING;

   function FORM   (FILE : in FILE_TYPE) return STRING;

   function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

-- input and output operations

   procedure READ  (FILE : in     FILE_TYPE;
                    ITEM :    out ELEMENT_TYPE);

   procedure WRITE (FILE : in FILE_TYPE;
                    ITEM : in ELEMENT_TYPE);
```

```
   function END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;

-- exceptions

   STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

   type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end SEQUENTIAL_IO;
```

F.8.4 Direct Input-Output

The implementation omits type checking for DATA_ERROR, in case the element type is of an unconstrained type, [Dod 83] 14.2.4(4), see F.8.3.

F.8.4.1 Specification of the Package Direct IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;

generic

   type ELEMENT_TYPE is private;

package DIRECT_IO is


   type FILE_TYPE is limited private;

   type FILE_MODE is (IN_FILE, INOUT_FILE, OUT_FILE);

   type COUNT is range 0..INTEGER'LAST;
   subtype POSITIVE_COUNT is COUNT range 1..COUNT'LAST;


-- File management

   procedure CREATE(FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE  := INOUT_FILE;
                    NAME : in     STRING     := "";
                    FORM : in     STRING     := "");

   procedure OPEN  (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE;
                    NAME : in     STRING;
                    FORM : in     STRING    := "");

   procedure CLOSE (FILE : in out FILE_TYPE);

   procedure DELETE(FILE : in out FILE_TYPE);

   procedure RESET (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE);

   procedure RESET (FILE : in out FILE_TYPE);

   function MODE   (FILE : in FILE_TYPE) return FILE_MODE;

   function NAME   (FILE : in FILE_TYPE) return STRING;

   function FORM   (FILE : in FILE_TYPE) return STRING;

   function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

-- input and output operations
```

```
   procedure READ  (FILE : in     FILE_TYPE;
                    ITEM :    out ELEMENT_TYPE;
                    FROM : in     POSITIVE_COUNT);
   procedure READ  (FILE : in     FILE_TYPE;
                    ITEM :    out ELEMENT_TYPE);

   procedure WRITE (FILE : in FILE_TYPE;
                    ITEM : in ELEMENT_TYPE;
                    TO   : in POSITIVE_COUNT);
   procedure WRITE (FILE : in FILE_TYPE;
                    ITEM : in ELEMENT_TYPE);

   procedure SET_INDEX(FILE : in FILE_TYPE;
                       TO   : in POSITIVE_COUNT);

   function INDEX(FILE : in FILE_TYPE) return POSITIVE_COUNT;

   function SIZE (FILE : in FILE_TYPE) return COUNT;

   function END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;


-- exceptions

   STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

  type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end DIRECT_IO;
```

F.8.5 Specification of the Package Text IO

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;
package TEXT_IO is

   type FILE_TYPE is limited private;

   type FILE_MODE is (IN_FILE, OUT_FILE);

   type    COUNT is range 0 .. INTEGER'LAST;
   subtype POSITIVE_COUNT is COUNT range 1 .. COUNT'LAST;
   UNBOUNDED: constant COUNT:= 0; -- line and page length

   subtype FIELD          is INTEGER range 0 .. 35;

   subtype NUMBER_BASE    is INTEGER range 2 .. 16;

   type TYPE_SET is (LOWER_CASE, UPPER_CASE);

   -- File Management

   procedure CREATE (FILE  : in out FILE_TYPE;
                     MODE  : in     FILE_MODE := OUT_FILE;
                     NAME  : in     STRING    := "";
                     FORM  : in     STRING    := "");

   procedure OPEN   (FILE  : in out FILE_TYPE;
                     MODE  : in     FILE_MODE;
                     NAME  : in     STRING;
                     FORM  : in     STRING    := "");

   procedure CLOSE  (FILE  : in out FILE_TYPE);
   procedure DELETE (FILE  : in out FILE_TYPE);
   procedure RESET  (FILE  : in out FILE_TYPE;
                     MODE  : in     FILE_MODE);
   procedure RESET  (FILE  : in out FILE_TYPE);

   function  MODE   (FILE  : in FILE_TYPE) return FILE_MODE;
   function  NAME   (FILE  : in FILE_TYPE) return STRING;
   function  FORM   (FILE  : in FILE_TYPE) return STRING;

   function  IS_OPEN(FILE  : in FILE_TYPE) return BOOLEAN;

   -- Control of default input and output files

   procedure  SET_INPUT   (FILE : in FILE_TYPE);
   procedure  SET_OUTPUT  (FILE : in FILE_TYPE);

   function   STANDARD_INPUT   return FILE_TYPE;
   function   STANDARD_OUTPUT  return FILE_TYPE;
```

```
function   CURRENT_INPUT    return FILE_TYPE;
function   CURRENT_OUTPUT   return FILE_TYPE;

-- specification of line and page lengths

procedure SET_LINE_LENGTH  (FILE : in FILE_TYPE;
                            TO   : in COUNT);
procedure SET_LINE_LENGTH  (TO   : in COUNT);

procedure SET_PAGE_LENGTH  (FILE : in FILE_TYPE;
                            TO   : in COUNT);
procedure SET_PAGE_LENGTH  (TO   : in COUNT);

function  LINE_LENGTH      (FILE : in FILE_TYPE) return
                                                 COUNT;
function  LINE_LENGTH                            return
                                                 COUNT;

function  PAGE_LENGTH      (FILE : in FILE_TYPE) return
                                                 COUNT;
function  PAGE_LENGTH                            return
                                                 COUNT;

-- Column, Line, and Page Control

procedure NEW_LINE    (FILE    : in FILE_TYPE;
                       SPACING : in POSITIVE_COUNT := 1);
procedure NEW_LINE    (SPACING : in POSITIVE_COUNT := 1);

procedure SKIP_LINE   (FILE    : in FILE_TYPE;
                       SPACING : in POSITIVE_COUNT := 1);
procedure SKIP_LINE   (SPACING : in POSITIVE_COUNT := 1);

function  END_OF_LINE (FILE : in FILE_TYPE) return
                                            BOOLEAN;
function  END_OF_LINE                       return
                                            BOOLEAN;

procedure NEW_PAGE    (FILE : in FILE_TYPE);
procedure NEW_PAGE                         ;

procedure SKIP_PAGE   (FILE : in FILE_TYPE);
procedure SKIP_PAGE                        ;

function  END_OF_PAGE (FILE : in FILE_TYPE) return
                                            BOOLEAN;
function  END_OF_PAGE                       return
                                            BOOLEAN;

function  END_OF_FILE (FILE : in FILE_TYPE) return
                                            BOOLEAN;
```

```
function  END_OF_FILE                             return
                                                  BOOLEAN;

procedure SET_COL     (FILE : in FILE_TYPE;
                       TO   : in POSITIVE_COUNT);
procedure SET_COL     (TO   : in POSITIVE_COUNT);

procedure SET_LINE    (FILE : in FILE_TYPE;
                       TO   : in POSITIVE_COUNT);
procedure SET_LINE    (TO   : in POSITIVE_COUNT);

function  COL         (FILE : in FILE_TYPE) return
                                  POSITIVE_COUNT;
function  COL                                return
                                  POSITIVE_COUNT;

function  LINE        (FILE : in FILE_TYPE) return
                                  POSITIVE_COUNT;
function  LINE                               return
                                  POSITIVE_COUNT;

function  PAGE        (FILE : in FILE_TYPE) return
                                  POSITIVE_COUNT;
function  PAGE                               return
                                  POSITIVE_COUNT;

-- Character Input-Output

procedure GET  (FILE : in     FILE_TYPE;
                ITEM :    out CHARACTER);
procedure GET  (ITEM :    out CHARACTER);
procedure PUT  (FILE : in FILE_TYPE;
                ITEM : in CHARACTER);
procedure PUT  (ITEM : in CHARACTER);

-- String Input-Output

procedure GET  (FILE : in     FILE_TYPE;
                ITEM :    out STRING);
procedure GET  (ITEM :    out STRING);
procedure PUT  (FILE : in FILE_TYPE;
                ITEM : in STRING);
procedure PUT  (ITEM : in STRING);

procedure GET_LINE  (FILE : in     FILE_TYPE;
                     ITEM :    out STRING;
                     LAST :    out NATURAL);
procedure GET_LINE  (ITEM :    out STRING;
                     LAST :    out NATURAL);
procedure PUT_LINE  (FILE : in     FILE_TYPE;
                     ITEM : in     STRING);
procedure PUT_LINE  (ITEM : in     STRING);
```

```
 -- Generic Package for Input-Output of Integer Types

generic
    type NUM is range <>;
package INTEGER_IO is

   DEFAULT_WIDTH : FIELD        := NUM'WIDTH;
   DEFAULT_BASE  : NUMBER_BASE  :=        10;

   procedure GET  (FILE  : in     FILE_TYPE;
                   ITEM  :    out NUM;
                   WIDTH : in     FIELD := 0);
   procedure GET  (ITEM  :    out NUM;
                   WIDTH : in     FIELD := 0);

   procedure PUT  (FILE  : in FILE_TYPE;
                   ITEM  : in NUM;
                   WIDTH : in FIELD := DEFAULT_WIDTH;
                   BASE  : in NUMBER_BASE := DEFAULT_BASE);
   procedure PUT  (ITEM  : in NUM;
                   WIDTH : in FIELD := DEFAULT_WIDTH;
                   BASE  : in NUMBER_BASE := DEFAULT_BASE);

   procedure GET  (FROM  : in     STRING;
                   ITEM  :    out NUM;
                   LAST  :    out POSITIVE);
   procedure PUT  (TO    :    out STRING;
                   ITEM  : in     NUM;
                   BASE  : in     NUMBER_BASE :=
                                        DEFAULT_BASE);

end INTEGER_IO;
```


```
   -- Generic Packages for Input-Output of Real Types

generic
   type NUM is digits <>;
package FLOAT_IO is

   DEFAULT_FORE : FIELD :=              2;
   DEFAULT_AFT  : FIELD := NUM'digits - 1;
   DEFAULT_EXP  : FIELD :=              3;

   procedure GET  (FILE  : in     FILE_TYPE;
                   ITEM  :    out NUM;
                   WIDTH : in     FIELD := 0);
   procedure GET  (ITEM  :    out NUM;
                   WIDTH : in     FIELD := 0);

   procedure PUT  (FILE : in FILE_TYPE;
                   ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);
   procedure PUT  (ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);

   procedure GET  (FROM : in     STRING;
                   ITEM :    out NUM;
                   LAST :    out POSITIVE);
   procedure PUT  (TO   :    out STRING;
                   ITEM : in     NUM;
                   AFT  : in     FIELD := DEFAULT_AFT;
                   EXP  : in     FIELD := DEFAULT_EXP);

end FLOAT_IO;
```

```
generic
   type NUM is delta <>;
package FIXED_IO is

   DEFAULT_FORE : FIELD := NUM'FORE;
   DEFAULT_AFT  : FIELD := NUM'AFT;
   DEFAULT_EXP  : FIELD := 0;

   procedure GET  (FILE  : in     FILE_TYPE;
                   ITEM  :    out NUM;
                   WIDTH : in     FIELD := 0);
   procedure GET  (ITEM  :    out NUM;
                   WIDTH : in     FIELD := 0);

   procedure PUT  (FILE : in FILE_TYPE;
                   ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);

   procedure PUT  (ITEM : in NUM;
                   FORE : in FIELD := DEFAULT_FORE;
                   AFT  : in FIELD := DEFAULT_AFT;
                   EXP  : in FIELD := DEFAULT_EXP);

   procedure GET  (FROM : in     STRING;
                   ITEM :    out NUM;
                   LAST :    out POSITIVE);
   procedure PUT  (TO   :    out STRING;
                   ITEM : in     NUM;
                   AFT  : in     FIELD := DEFAULT_AFT;
                   EXP  : in     FIELD := DEFAULT_EXP);

end FIXED_IO;

   -- Generic Package for Input-Output of Enumeration Types
```

```
generic
   type ENUM is (<>);
package ENUMERATION_IO is

   DEFAULT_WIDTH   : FIELD    := 0;
   DEFAULT_SETTING : TYPE_SET := UPPER_CASE;

   procedure GET  (FILE : in     FILE_TYPE;
                   ITEM :    out ENUM);
   procedure GET  (ITEM :    out ENUM);

   procedure PUT  (FILE  : in FILE_TYPE;
                   ITEM  : in ENUM;
                   WIDTH : in FIELD       := DEFAULT_WIDTH;
                   SET   : in TYPE_SET    := DEFAULT_SETTING);

   procedure PUT  (ITEM  : in ENUM;
                   WIDTH : in FIELD       := DEFAULT_WIDTH;
                   SET   : in TYPE_SET    := DEFAULT_SETTING);

   procedure GET  (FROM : in     STRING;
                   ITEM :    out ENUM;
                   LAST :    out POSITIVE);
   procedure PUT  (TO   :    out STRING;
                   ITEM : in     ENUM;
                   SET  : in     TYPE_SET := DEFAULT_SETTING);

end ENUMERATION_IO;

   -- Exceptions

   STATUS_ERROR : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR   : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR   : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR    : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR    : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR   : exception renames IO_EXCEPTIONS.DATA_ERROR;
   LAYOUT_ERROR : exception renames IO_EXCEPTIONS.LAYOUT_ERROR;

private

  type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end TEXT_IO;
```

F.8.6 Low Level Input-Output

The package LOW_LEVEL_IO is empty.

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.	<125 X "A">1
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	<125 X "A">2
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	<63 X "A">3<62 X "A">
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	<63 X "A">4<62 X "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.	<123 X "0">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.	<120 X "0">69.0E1

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	<63 X "A">
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	<62 X "A">1
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	<106 X " ">
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	35
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	string'(1..255=>'A')
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	string'(1..255=>'B')
$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
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	200_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	string'(1..255=>'A')

$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	string'(1..255=>'B')
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2147483648
$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.	-200_000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	126
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$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.	<121 X "0"> 2:11:

$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:<119 X "0">F.E
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	"<124 X "A">"
$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.	SHORT_SHORT_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#ffffffff#

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, Various discriminant constraints are wrongly expected
E, G, H: 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: AI-330 states this test is to be changed from an "E" test to a "B" test during the next version of the ACVC. AI-330 was approved in July 1986 6 months before the initial version of ACVC Version 1.9 was released and a nearly a full year before the final version of ACVC Version 1.9 was released. This test is withdrawn pending further comment from AJPO regarding issue of the test being a B Test rather than an E Test.

A74106C, A bound specified in a fixed-point subtype declaration
C85018B, lies outside of that calculated for the base type, raising
C87B04B, CONSTRAINT_ERROR. Errors of this sort occur re lines 37 & 59,
CC1311B: 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.