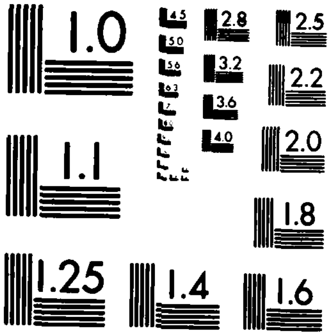
AD-A193 612   ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT   1/1
CONCURRENT COMPUTER (U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI   04 JUN 87
UNCLASSIFIED   AVF-VSR-75 0887   F/G 12/5   NL



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

AVF Control Number: AVF-VSR-75.0887
87-03-17-CCC

Ada® COMPILER
VALIDATION SUMMARY REPORT:
Concurrent Computer Corporation
C^3 Ada, Version R00-01.00
Concurrent Computer Corporation 3260 MPS

Completion of On-Site Testing:
4 June 1987

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

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

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

UNCLASSIFIED

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

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETEING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.
A192 612

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
Ada Compiler Validation Summary Report: Concurrent Computer Corporation C3 Ada, Ver. R00-01.00 Concurrent Computer Corporation 3260 MPS

5. TYPE OF REPORT & PERIOD COVERED
4 June 87 to 4 June 88

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Wright-Patterson AFB OH 45433-6503

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS
Wright-Patterson AFB OH 45433-6503

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

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

12. REPORT DATE
4 June 1987

13. NUMBER OF PAGES
43 p.

14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)*
Wright-Patterson AFB OH 45433-6503.

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)*

See Attached.

UNCLASSIFIED

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

```
++++++++++++++++++++++++
+                      +
+ Place NTIS form here +
+                      +
++++++++++++++++++++++++
```

Ada[®] Compiler Validation Summary Report:

Compiler Name: C^3 Ada, Version R00-01.00

Host:	Target:
Concurrent Computer Corporation 3260 MPS under OS/32, Version R08-02	Concurrent Computer Corporation 3260 MPS under OS/32, Version R08-02

Testing Completed 4 June 1987 Using ACVC 1.8

This report has been reviewed and is approved.

Steven P. Wilson by/ Georgeanne Chitwood

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

DTIC COPY INSPECTED 1

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

Virginia L. Castor

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

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

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the C^3 Ada® compiler, Version R00-01.00, using Version 1.8 of the Ada Compiler Validation Capability (ACVC). The C^3 Ada compiler is hosted on a Concurrent Computer Corporation 3260 MPS operating under OS/32, Version R08-02. Programs processed by this compiler may be executed on a Concurrent Computer Corporation 3260 MPS operating under OS/32, Version R08-02.

On-site testing was performed 29 May 1987 through 4 June 1987 at Concurrent Computer Corporation in Tinton Falls NJ, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2210 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 170 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2210 tests were processed, results for Class A, C, D, and E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 35 of the processed tests determined to be inapplicable. The remaining 2175 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	94	252	334	244	161	97	137	261	124	32	218	221	2175
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	22	73	86	3	0	0	2	1	6	0	0	12	205
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

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

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

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

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

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

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

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

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

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

- To attempt to identify any unsupported language constructs required by the Ada Standard

- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 29 May 1987 through 4 June 1987 at Concurrent Computer Corporation in Tinton Falls NJ.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

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

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

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

or from:

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

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

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

1.3 REFERENCES

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

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

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

1.4 DEFINITION OF TERMS

ACVC The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.

Ada Standard ANSI/MIL-STD-1815A, February 1983.

Applicant The agency requesting validation.

AVF The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.

AVO The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.

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

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

Host The computer on which the compiler resides.

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

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

Target — The computer for which a compiler generates code.

Test — A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn test — A test found to be incorrect and not used to check conformity to the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

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

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

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

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

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

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

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

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

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features 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: C^3 Ada, Version R00-01.00

ACVC Version: 1.8

Certificate Number: 870601W1.08061

Host Computer:

Machine:	Concurrent Computer Corporation 3260 MPS
Operating System:	OS/32, Version R08-02
Memory Size:	16 megabytes

Target Computer:

Machine:	Concurrent Computer Corporation 3260 MPS
Operating System:	OS/32, Version R08-02
Memory Size:	16 megabytes

2.2 IMPLEMENTATION CHARACTERISTICS

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

- Capacities.

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

- Universal integer calculations.

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

- Predefined types.

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

- Based literals.

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

- Array types.

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

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

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

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

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

- Discriminated types.

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

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

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

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

- Functions.

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

- Representation clauses.

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

- Pragmas.

The pragma INLINE is not supported for procedures or functions. (See tests CA3004E and CA3004F.)

- Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

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

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

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

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

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

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

- Generics.

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

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

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of the C^3 Ada compiler was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 205 tests were inapplicable to this implementation, and that the 2175 applicable tests were passed by the implementation.

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

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	67	865	1171	17	11	44	2175
Failed	0	0	0	0	0	0	0
Inapplicable	2	2	197	0	2	2	205
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	94	252	334	244	161	97	137	261	124	32	218	221	2175
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	22	73	86	3	0	0	2	1	6	0	0	12	205
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

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

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

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

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 205 tests were inapplicable for the reasons indicated:

- C24113D..K (8 tests) have line lengths greater than MAX IN LEN.
- C34001E, B52004D, B55B09C, and C55B07A use LONG INTEGER which is not supported by this compiler.
- C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.

- C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

- C87B62B uses the length clause 'STORAGE_SIZE for task types which is not supported by this compiler. The length clause is rejected during compilation.

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

- CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

- AE2101C, CE2201D, and CE2201E use an instantiation of package SEQUENTIAL_IO with unconstrained array types which is not supported by this compiler.

- AE2101H and CE2401D use an instantiation of package DIRECT_IO with unconstrained array types which is not supported by this compiler.

- CE2107C, CE2107D, CE2108A, CE2108C, and CE3112A are inapplicable because temporary files do not have names.

- CE3111B is inapplicable because the TEXT_IO.PUT operation does not output to the external file until a subsequent NEW_LINE, RESET, or CLOSE operation is executed.

- CE3114B attempts to delete an external file that is associated with multiple internal files. This implementation does not allow the external file to be deleted for TEXT_IO.

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

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

3.6 SPLIT TESTS

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

Splits were required for four Class B tests:

B23004A	B29001A	BC3204B	BC3205B

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by the C^3 Ada compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and that the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the C^3 Ada compiler using ACVC Version 1.8 was conducted on site by a validation team from the AVF. The configuration consisted of a Concurrent Computer Corporation 3260 MPS operating under OS/32, Version R08-02.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests BC3204B and BC3205B were included in their split form on the magnetic tape. Tests B23004A and B29001A were edited into their split form on site.

The contents of the magnetic tape were loaded to disk using a FORTRAN program developed by Concurrent Computer Corporation. This utility was used to truncate filenames to eight characters and to modify the files whose line lengths exceed Concurrent's 80-character record length. After the test files were loaded to disk, the full set of tests was compiled, linked, and executed as appropriate on the Concurrent Computer Corporation 3260 MPS. Results were printed from the Concurrent Computer Corporation 3260 MPS.

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

LIST The LIST option controls the generation of the source listing from the compiler. A listing of all source lines is generated.

OPTIMIZE This option controls the action of performing simple optimizations like constant folding, dead code elimination, and peephole optimization.

PAGE_SIZE This option specifies the number of significant lines per page on the listing file. The default is 60 lines per page.

SEGMENTED This option specifies that the code generated is to be segmented in PURE and IMPURE code.

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

3.7.3 Test Site

The validation team arrived at Concurrent Computer Corporation in Tinton Falls NJ on 29 May 1987, and departed after testing was completed on 4 June 1987.

APPENDIX A

DECLARATION OF CONFORMANCE

Concurrent Computer Corporation has submitted the following declaration of conformance concerning the C^3 Ada compiler.

DECLARATION OF CONFORMANCE

Compiler Implementor: Concurrent Computer Corporation
Ada® Validation Facility: ASD/SCOL, Wright-Patterson AFB, OH
Ada Compiler Validation Capability (ACVC) Version: 1.8

Base Configuration

Base Compiler Name: C^3 Ada Version: R00-01.00

Host Architecture ISA: Concurrent Computer Corporation 3260 MPS
OS&VER #: OS/32, Version R08-02
Target Architecture ISA: Concurrent Computer Corporation 3260 MPS
OS&VER #: OS/32, Version R08-02

Derived Compiler Registration

Derived Compiler Name: C^3 Ada Version: R00-01.00

Host Architecture ISA: Concurrent Computer Corporation Series 3200
3200MPS, 3203, 3205, 3210, 3230, 3250,
3230XP, 3250XP, 3230MPS, 3260MPS, 3280MPS
OS&VER #: OS/32, Version R08-02
Target Architecture ISA: All Hosts, Self Targeted
OS&VER #: OS/32, Version R08-02

Implementor's Declaration

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

______________________________ Date: ______________________________
Concurrent Computer Corporation
Seetharama Shastry
Manager, System Software Development

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

Owner's Declaration

I, the undersigned, representing Concurrent Computer Corporation, take full responsibility for implementation and maintenance of the Ada compilers 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:__________________________

Concurrent Computer Corporation
Seetharama Shastry
Manager, System Software 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 C^3 Ada compiler, Version R00-01.00, 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 TINY_INTEGER is range -128 .. 127;

        type FLOAT is digits 6 range -16#0.FFFF_FF#E63 .. 16#0.FFFF_FF#E63;
        type LONG_FLOAT is digits 15 range -16#FFFF_FFFF_FFFF_FF#E63 ..
                                            16#FFFF_FFFF_FFFF_FF#E63;

        type DURATION is delta 0.00006103515625 range -131072.0 ..
                                                       131071.99993896484375;

        ...

    end STANDARD;
```

APPENDIX F
IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 INTRODUCTION

The following sections provide all implementation-dependent characteristics of the C^3Ada Compiler.

F.2 IMPLEMENTATION-DEPENDENT PRAGMAS

The following is the syntax representation of a pragma:

```
pragma IDENTIFIER [(ARGUMENT {,ARGUMENT})];
```

Where:

IDENTIFIER is the name of the pragma.

ARGUMENT defines a parameter of the pragma. For example, the LIST pragma expects the arguments ON or OFF.

Table F-1 summarizes all of the recognized pragmas and whether they are implemented or not.

TABLE F-1. SUMMARY OF RECOGNIZED PRAGMAS

PRAGMA	IMPLEMENTED	COMMENTS
CONTROLLED	No	Automatic storage reclamation of unreferenced access objects is not applicable to the C^3Ada implementation.
ELABORATE	Yes	Is handled as defined by the Ada language.
INLINE	No	Subprogram bodies are not expanded inline at each call.
INTERFACE	Yes	Is implemented for ASSEMBLER and FORTRAN.
LIST	Yes	Is handled as defined by the Ada language.
MEMORY_SIZE	No	The user cannot specify the number of available storage units in the machine configuration which is defined in package SYSTEM.
OPTIMIZE	No	The user cannot specify either time or space as the primary optimization criterion.
PACK	Yes	The elements of an array or record are packed down to a minimal number of bytes.
PAGE	Yes	Is handled as defined by the Ada language.
PRIORITY	No	The task or main_program cannot have priority.
SHARED	No	Not applicable because every read or update of the variable declared by an object declaration and whose type is a scalar or access type is a synchronization point for that variable.
STORAGE_UNIT	No	The user cannot specify the number of bits per storage unit, which is defined in package SYSTEM.

TABLE F-1. SUMMARY OF RECOGNIZED PRAGMAS (Continued)

PRAGMA	IMPLEMENTED	COMMENTS
SUPPRESS	No	All run-time checks, such as ACCESS_CHECK, INDEX_CHECK, RANGE_CHECK, etc., cannot be suppressed for any specific type, object, subprogram etc., See the description of SUPPRESS_ALL.
SYSTEM_NAME	No	The user cannot specify the target system name, which is defined in package SYSTEM.
SUPPRESS_ALL	Yes	This pragma gives the compiler permission to omit all of the following run-time checks for all types and objects in the designated compilation units: ACCESS_CHECK, RANGE_CHECK, LENGTH_CHECK, INDEX_CHECK, DISCRIMINANT_CHECK and OVERFLOW_CHECK for all integer and fixed point calculations. The pragma must be placed before each compilation unit.
NO_STACK_CHECK	Yes	This pragma indicates to the compiler that there is enough space in the initial stack chunk for the activation record of all subroutines that may be active at any time. Therefore, no code is generated to check for providing additional space for the run-time stack of any task or of the main task. The pragma must be placed before the compilation, and applies to all compilation units.

F.3 LENGTH CLAUSES

A length clause specifies the amount of storage associated with a given type. The following is a list of the implementation-dependent attributes.

T'SIZE	must be a multiple of eight. Must be 32 for a type derived from FLOAT, and 64 for a type derived from LONG_FLOAT. For array and record types, only the size chosen by the compiler may be specified.
T'STORAGE_SIZE	is fully supported for collection size specification.
T'STORAGE_SIZE	is not supported for task activation. Task memory is limited by the work space for the program.
T'SMALL	must be a power of two for a fixed point type.

Size representation only applies to types - not to subtypes. In the following example, the size of T is 32, but the size of T1 is not necessarily 32.

```
type T is integer range  0..100;
subtype T1 is T range 0..10;
for T'SIZE use 32;
```

In the following example, the size of the subtype is the same as the size of the type (size of the type is applied to the subtype).

```
type T is integer range  0..100;
for T'SIZE use 32;
subtype T2 is T range 0..10;
```

F.4 REPRESENTATION ATTRIBUTES

The Representation attributes listed below are as described in the *Reference Manual for the Ada Programming Language*, Section 13.7.2.

X'ADDRESS - Note: Attribute ADDRESS is not supported for labels.
X'SIZE
R.C'POSITION
R.C'FIRST_BIT
R.C'LAST_BIT

T'STORAGE_SIZE for access types, returns the current amount of storage reserved for the type. If a T'STORAGE_SIZE representation clause has been specified, then the amount specified is returned. Otherwise the current amount allocated is returned.

T'STORAGE_SIZE for task types or objects is not implemented. It returns 0.

F.4.1 Representation Attributes of Real Types

P'DIGITS yields the number of decimal digits for the subtype P. This value is six for type FLOAT, and 15 for type LONG_FLOAT.

P'MANTISSA yields the number of binary digits in the mantissa of P. The value is 21 for type FLOAT, and 51 for type LONG_FLOAT.

DIGITS	MANTISSA	DIGITS	MANTISSA	DIGITS	MANTISSA
1	5	6	21	11	38
2	8	7	25	12	41
3	11	8	28	13	45
4	15	9	31	14	48
5	18	10	35	15	51

P'EMAX yields the largest exponent value of model numbers for the subtype P. The value is 84 for type FLOAT, and 204 for type LONG_FLOAT.

DIGITS	EMAX	DIGITS	EMAX	DIGITS	EMAX
1	20	6	84	11	152
2	32	7	100	12	164
3	44	8	112	13	180
4	60	9	124	14	192
5	72	10	140	15	204

P'EPSILON yields the absolute value of the difference between the model number 1.0 and the next model number above for the subtype P. The value is 16#0.00001# for type FLOAT, and 16#0.0000_0000_0000_4# for type LONG_FLOAT.

VALUES	EMAX	VALUES	EMAX	VALUES	EMAX
1	20	6	84	11	152
2	32	7	100	12	164
3	44	8	112	13	180
4	60	9	124	14	192
5	72	10	140	15	204

P'SMALL yields the smallest positive model number of the subtype P. The value is 16#0.8#E-21 for type FLOAT, and 16#0.8#E-51 for type LONG_FLOAT.

VALUES	SMALL	VALUES	SMALL	VALUES	SMALL
1	16#0.8#E-5	6	16#0.8#E-21	11	16#0.8#E-38
2	16#0.8#E-8	7	16#0.8#E-25	12	16#0.8#E-41
3	16#0.8#E-11	8	16#0.8#E-28	13	16#0.8#E-45
4	16#0.8#E-15	9	16#0.8#E-31	14	16#0.8#E-48
5	16#0.8#E-18	10	16#0.8#E-35	15	16#0.8#E-51

P'LARGE yields the largest positive model number of the subtype P. The value is 16#0.FFFFFF8#E21 for type FLOAT, and 16#0.FFFF_FFFF_FFFF_E#E51 for type LONG_FLOAT.

VALUES	LARGE
1	16#0.F8#E5
2	16#0.FF#E8
3	16#0.FFE#E11
4	16#0.FFFE#E15
5	16#0.FFFF_C#E18
6	16#0.FFFF_F8#E21
7	16#0.FFFF_FF8#E25
8	16#0.FFFF_FFF#E28
9	16#0.FFFF_FFFE#E31
10	16#0.FFFF_FFFF_E#E35
11	16#0.FFFF_FFFF_FC#E38
12	16#0.FFFF_FFFF_FF8#E41
13	16#0.FFFF_FFFF_FFF8#E45
14	16#0.FFFF_FFFF_FFFF#E48
15	16#0.FFFF_FFFF_FFFF_E#E51

P'SAFE_EMAX	yields the largest exponent value of safe numbers of type P. The value is 252 for types FLOAT and LONG_FLOAT.
P'SAFE_SMALL	yields the smallest positive safe number of type P. The value is 16#0.1#E-64 for types FLOAT and LONG_FLOAT.
P'SAFE_LARGE	yields the largest positive safe number of the type P. The value is 16#0.FFFF_FF#E63 for type FLOAT, and 16#0.FFFF_FFFF_FFFF_FF#E63 for type LONG_FLOAT.
P'MACHINE_ROUNDS	is true.
P'MACHINE_OVERFLOWS	is true.
P'MACHINE_RADIX	is 16.
P'MACHINE_MANTISSA	is six for types derived from FLOAT; else 14.
P'MACHINE_EMAX	is 63.
P'MACHINE_EMIN	is -64.

F.4.2 Representation Attributes of Fixed Point Types

For any fixed point type T, the representation attributes are:

T'MACHINE_ROUNDS	true
T'MACHINE_OVERFLOWS	true

F.4.3 Enumeration Representation Clauses

The maximum number of elements in an enumeration type is limited by the maximum size of the enumeration image table which cannot be greater than 65535 bytes. The enumeration table size is determined by the following function:

```
generic
   type ENUMERATION_TYPE IS (< >);
function ENUMERATION_TABLE_SIZE return NATURAL is
   Result : NATURAL :=0;
begin
   for I in ENUMERATION_TYPE 'FIRST..ENUMERATION_TYPE' LAST loop
       RESULT :=RESULT + 2 + I'WIDTH;
   End loop;
   return RESULT;
END ENUMERATION_TABLE_SIZE;
```

RESTRICTIONS - None.

F.4.4 Record Representation Clauses

The *Reference Manual for the Ada Programming Language* states that an implementation may generate names that denote implementation-dependent components. This is not present in this release of the C^3Ada Compiler.

RESTRICTIONS - Components must be placed at a storage position that is a multiple of eight. Floating point types must be fullword-aligned, that is, placed at a storage position that is a multiple of 32.

Record components of a private type cannot be included in a record representation specification.

Record clause alignment can only be 1, 2 or 4.

Component representations for access types must allow for at least 24 bits.

Component representations for scalar types other than for types derived from LONG_FLOAT must not specify more than 32 bits.

F.4.5 Type Duration

Duration'small equals 61.03515625 microseconds or 2^{-14} seconds. This number is the smallest power of two which can still represent the number of seconds in a day in a fullword fixed point number.

System.tick equals 10ms. The actual computer clock-tick is 1.0/120.0 seconds (or about 8.33333ms) in 60HZ areas and 1.0/100.0 seconds (or 10ms) in 50HZ areas. System.tick represents the greater of the actual clock-tick from both areas.

Duration'small is significantly smaller than the actual computer clock-tick Therefore, the least amount of delay possible is limited by the actual clock-tick. The delay of duration'small follows this formula:

<actual-clock-tick> ± <actual-clock-tick> + 4.45ms

The 4.45ms represents the overhead or the minimum delay possible on a Model 3250 or 3200MPS Family of Processors. For 60HZ areas, the range of delay is approximately from 4.45ms to 21.11666ms. For 50HZ areas, the range of delay is approximately from 4.45ms to 24.45ms. However, on the average, the delay is slightly greater than the actual clock-tick.

In general, the formula for finding the range of a delay value, *x*, is:

nearest_multiple(x,<actual-clock-tick>) ± <actual-clock-tick> + 4.45ms

where nearest_multiple rounds *x* up to the nearest multiple of the actual clock-tick.

TABLE F-2. TYPE DURATION

DURATION'DELTA	2#1.0#E-14	≃ 61μs
DURATION'SMALL	2#1.0#E-14	≃ 61μs
DURATION'FIRST	-131072.00	≃ 36 hrs
DURATION'LAST	131071.99993896484375	≃ 36 hrs
DURATION'SIZE	32	

F.5 ADDRESS CLAUSES

Address clauses are implemented for objects. No storage is allocated for objects with address clauses by the compiler. The user must guarantee the storage for these by some other means (e.g., through the use of the absolute instruction found in the *Common Assembly Language/32 (CAL/32) Reference Manual*). The exception PROGRAM_ERROR is raised upon reference to the object if the specified address is not in the program's address space or is not properly aligned.

RESTRICTIONS - Address clauses are not implemented for subprograms, packages or task units. In addition, address clauses are not available for use with task entries (i.e., interrupts).

Initialization of an object that has an address clause specified is not supported. Objects with address clauses may also be used to map objects into global task common (TCOM) areas. See Chapter 4 for more information regarding task common.

F.6 THE PACKAGE SYSTEM

The package SYSTEM, provided with C³Ada permits access to machine-dependent features. The specification of the package SYSTEM declares constant values dependent on the Series 3200 Processors. The following is a listing of the visible section of the package SYSTEM specification.

```
package SYSTEM is

   type ADDRESS is private;

   type NAME is (CCUR_3200);

   SYSTEM_NAME   : constant NAME := CCUR_3200,
   STORAGE_UNIT  : constant := 8;
   MEMORY_SIZE   : constant := 2 ** 24;
   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-30;
   TICK          : constant := 0.01;

   type UNSIGNED_SHORT_INTEGER is range 0 .. 65_535;

   type UNSIGNED_TINY_INTEGER is range 0 .. 255;

   for UNSIGNED_SHORT_INTEGER'SIZE use 16;

   for UNSIGNED_TINY_INTEGER'SIZE use 8;

   subtype PRIORITY      is INTEGER range 0 .. 255;

   subtype BYTE          is UNSIGNED_TINY_INTEGER;

   subtype ADDRESS_RANGE is INTEGER range 0 .. 2 ** 24 - 1;

   ADDRESS_NULL : constant ADDRESS;

   --These functions efficiently copy aligned elements of the specified size.
   --You can declare them locally using any scalar types with
   --PRAGMA interface(assembler,<Routine>);
   --WARNING: these routines work for scalar types only!!!!!!

   function  COPY_DOUBLEWORD (FROM : LONG_FLOAT) return LONG_FLOAT;
   pragma INTERFACE (ASSEMBLER, COPY_DOUBLEWORD);

   function  COPY_FULLWORD (FROM : INTEGER) return ADDRESS;
   function  COPY_FULLWORD (FROM : ADDRESS) return INTEGER;
   pragma INTERFACE (ASSEMBLER, COPY_FULLWORD);

   function  COPY_HALFWORD (FROM : SHORT_INTEGER) return SHORT_INTEGER;
   pragma INTERFACE (ASSEMBLER, COPY_HALFWORD);

   function  COPY_BYTE (FROM : TINY_INTEGER) return TINY_INTEGER;
   pragma INTERFACE (ASSEMBLER, COPY_BYTE);

   --Address conversion routines

   function  INTEGER_TO_ADDRESS (ADDR : ADDRESS_RANGE) return ADDRESS
      renames COPY_FULLWORD;

   function  ADDRESS_TO_INTEGER (ADDR : ADDRESS) return ADDRESS_RANGE
      renames COPY_FULLWORD;

   function  "+" (ADDR   : ADDRESS;
                  OFFSET : INTEGER) return ADDRESS;

   function  "-" (ADDR   : ADDRESS;
                  OFFSET : INTEGER) return ADDRESS;
   --This is a 32-bit type which is passed by value

   type EXCEPTION_ID is private;

   function  LAST_EXCEPTION_ID return EXCEPTION_ID;

private

   --Implementation defined

end SYSTEM;
```

F.7 INTERFACE TO OTHER LANGUAGES

Pragma INTERFACE is implemented for two languages, assembler and FORTRAN. The pragma can take one of three forms:

1. For any assembly language procedure or function:

   ```
   pragma INTERFACE (ASSEMBLER, ROUTINE_NAME);
   ```

2. For FORTRAN functions with only **in** parameters or procedures:

   ```
   pragma INTERFACE (FORTRAN, ROUTINE_NAME);
   ```

3. For FORTRAN functions that have **in out** or **out** parameters:

   ```
   pragma INTERFACE (FORTRAN, ROUTINE_NAME, IS_FUNCTION);
   ```

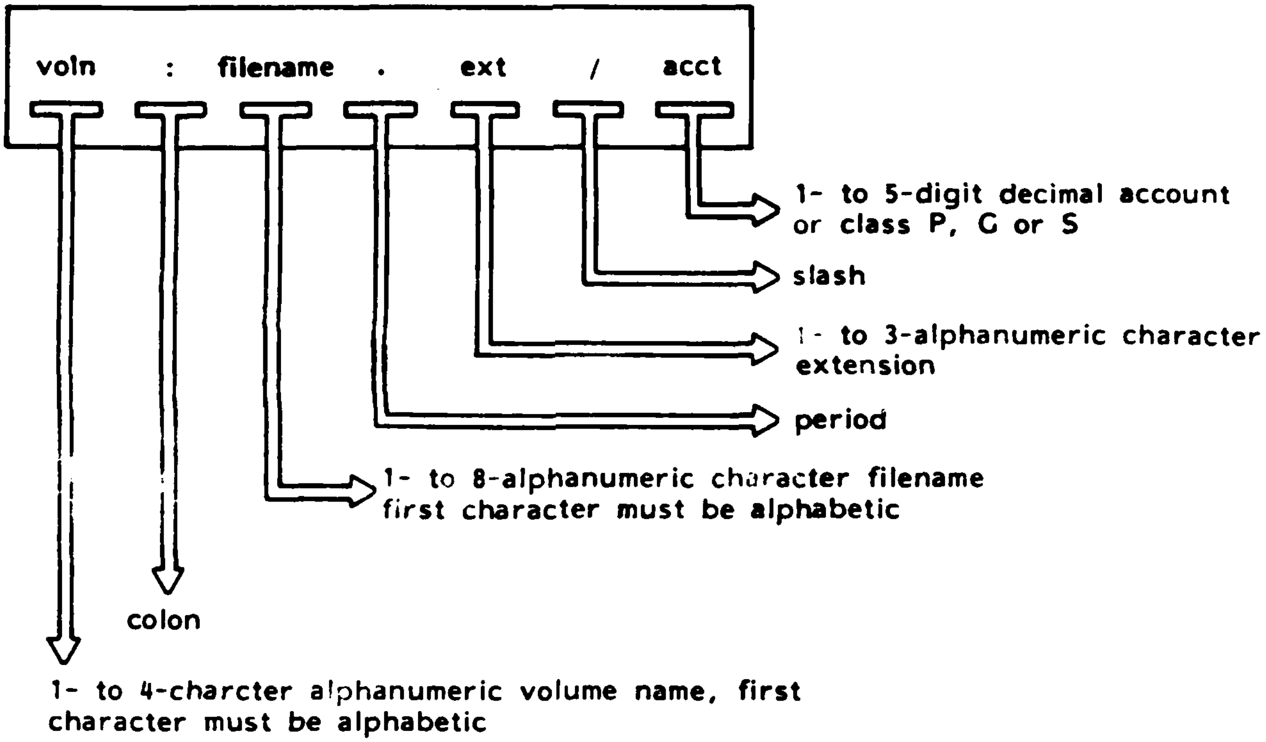
In C^3Ada functions cannot have **in out** or **out** parameters so the Ada specification for the function is written as a procedure with the first argument being the function return result. Then, the parameter "is_function" is specified to inform the compiler that it is, in reality, a FORTRAN function. Interface routine_names are truncated to an 8 character maximum length.

F.8 INPUT/OUTPUT (I/O) PACKAGES

The following two system-dependent parameters are used for the control of external files:

- NAME parameter
- FORM parameter

The NAME parameter must be an OS/32 file name string. OS/32 filenames are specified as follows:


voln
:
filename
.
ext
/
acct
1- to 5-digit decimal account or class P, G or S
slash
1- to 3-alphanumeric character extension
period
1- to 8-alphanumeric character filename first character must be alphabetic
colon
1- to 4-charcter alphanumeric volume name, first character must be alphabetic

The implementation-dependent values used for keywords in the FORM parameter are discussed below. The FORM parameter is a string that contains further system-dependent characteristics and attributes of an external file. The FORM parameter is able to convey to the file system information on the intended use of the associated external file. This parameter is used as one of the specifications for the CREATE procedure and the OPEN procedure. It specifies a number of system-dependent characteristics such as lu, file format, etc. It is returned by the FORM function.

The syntax of the FORM string, in our implementation, uses Ada syntax conventions and is as follows:

```
form_param  ::= [form_spec {, form_spec}]
form_spec   ::= lu_spec | fo_spec |
                rs_spec | dbf_spec |
                ibf_spec | al_spec |
                pr_spec | keys_spec |
                pad_spec | dc_spec |
                da_spec | ds_spec |
                ps_spec | ch_spec
lu_spec     ::= LU => lu
fo_spec     ::= FILE_ORGANIZATION => fo
rs_spec     ::= RECORD_SIZE => rs
dbf_spec    ::= DATA_BLOCKING_FACTOR => dbf
ibf_spec    ::= INDEX_BLOCKING_FACTOR => ibf
al_spec     ::= ALLOCATION => al
pr_spec     ::= PRIVILEGE => pr
keys_spec   ::= KEYS => keys
pad_spec    ::= PAD => pad
dc_spec     ::= DEVICE_CODE => dc
da_spec     ::= DEVICE_ATTRIBUTE => da
ds_spec     ::= DEVICE_STATUS => ds
ps_spec     ::= PROMPTING_STRING => ps
ch_spec     ::= CHARACTER_IO
```

The exception USE_ERROR is raised if a given FORM parameter string does not have the correct syntax or if certain conditions concerning the OPEN or CREATE statements are not fulfilled. Keywords that are listed above in upper-case letters are also recognized by the compiler in lower-case.

lu an integer in the range 0..254 specifying the logical unit (lu) number.

fo specifies legal OS/32 file formats (file organization). They are:

INDEX | IN
CONTIGUOUS | CO
NON_BUFFERED | NB
EXTENDABLE_CONTIGUOUS | EXTENDABLE_CONTIGUOUS | EC
LONG_RECORD | LR
ITAM
DEVICE

rs an integer in the range 1..65535 specifying the physical record size.

1. For INDEX, ITAM (inter telecommunications access method) and NON_BUFFERED files, this specifies the physical record size.
2. The physical record size for CONTIGUOUS and EXTENDABLE_CONTIGUOUS files is determined by rounding the element size up to the nearest 256-byte boundary. For such files, *rs* is ignored.
3. The physical record size for LONG_RECORD files is specified by the data blocking factor multiplied by 256 and *rs* is ignored.
4. For a DEVICE the physical record size always equals the element size and *rs* is ignored.

dbf Data_blocking_factor. An integer in the range 0..255 (as set up at OS/32 system generation (sysgen) time) that specifies the number of contiguous disk sectors (256 bytes) in a data block. It applies only to INDEX, NON_BUFFERED, EXTENDABLE_CONTIGUOUS and LONG_RECORD files. For other file organizations (see *file_organization* above), it is ignored. A value of 0 causes the data blocking factor to be set to the current OS/32 default.

ibf Index_blocking_factor. An integer in the range 0..255 (as set up at OS/32 sysgen time) specifying the number of contiguous disk sectors (256 bytes) in an index block of an INDEX, NON_BUFFERED, EXTENDABLE_CONTIGUOUS or LONG_RECORD file. For other file organizations (see *file_organization* above), it is ignored.

al Allocation. An integer in the range 1..2,147,483,647. For CONTIGUOUS files, it specifies the number of 256 byte sectors. For ITAM files, it specifies the physical block size in bytes associated with the buffered terminal. For other file organizations, (see *file_organization* above), it is ignored.

pr Privileges. Specifies OS/32 access privileges, e.g., shared read-only (SRO), exclusive read-only (ERO), shared write-only (SWO), exclusive write-only (EWO), shared read/write (SRW), shared read/exclusive write (SREW), exclusive read/shared write (ERSW) and exclusive read/write (ERW).

keys READ/WRITE keys. A decimal or hexadecimal integer specifying the OS/32 READ/WRITE keys, which range from 16#0000# to 16#FFFF#(0..65535). The left two hexadecimal digits signify the write protection key and the right two hexadecimal digits signify the read protection key. For more information on protection keys, see the *OS/32 Multi-Terminal Monitor (MTM) Primer.*

pad Pad character. Specifies the padding character used for READ and WRITE operations, the pad character is either NONE, BLANK or NUL. The default is NONE.

TABLE F-3. PAD CHARACTER OPTIONS

PAD CHARACTER	ACTION
NONE	Records are not padded. (Default.)
NUL	Records are padded with ASCII.NUL.
BLANK	Records are padded with blanks and OS/32 ASCII I/O operations are used.

dc Device code. An integer in the range 0..255 specifying the OS/32 device code of the external file. See the *System Generation/32 (SYSGEN/32) Reference Manual* for a list of all devices and their respective codes.

da Device attributes. An integer in the range 0..65535 specifying the OS/32 device attributes of the external file. See the *OS/32 Supervisor Call (SVC) Reference Manual* (Chapter 7, the table entitled Description and Mask Values of the Device Attributes Field) for all devices and their respective attributes.

ds Device status. An integer in the range 0..65535 specifying the status of the external file. A status of 0 means that the access to the file terminated with no errors, otherwise a device error has occurred. For errors occurring during READ and WRITE operations, the status values and their meanings are found in Chapter 2 (The tables on Device-Independent and Device-Dependent Status Codes) of the *OS/32 Supervisor Call (SVC) Reference Manual.*

ps Prompting string. This quoted string is output on the terminal before the GET operation only if the file is associated with a terminal; otherwise this FORM parameter is ignored. The default is the null string, in which case no string is output to the terminal.

character_io If character_io is specified in the FORM string, the only other allowable FORM *parameters* are LU => *lu*, FILE_ORGANIZATION => DEVICE and PRIVILEGE=> SRW. Furthermore, the NAME string must denote a terminal or interactive device. In order for character_io to work properly, the user must specify ENABLE TYPEAHEAD to MTM, to turn on BIOC's type ahead feature.

F.8.1 Text Input/Ouput (I/O)

There are two implementation-dependent types for TEXT_IO: COUNT and FIELD. Their declarations implemented for the C^{3}Ada Compiler are as follows:

```
type COUNT is range 0 .. INTEGER'LAST;
subtype FIELD is INTEGER range 0 .. 255;
```

F.8.1.1 End of File Markers

When working with text files, the following representations are used for end of file markers. A line terminator followed by a page terminator is represented by:

```
ASCII.FF  ASCII.CR
```

A line terminator followed by a page terminator, which is then followed by a file terminator is represented by:

```
ASCII.FF ASCII.EOT ASCII.CR
```

End of file may also be represented as the physical end of file. For input from a terminal, the combination above is represented by the control characters:

```
ASCII.FF ASCII.EOT ASCII.CR
```

or with BIOC:

```
ASCII.DC4 ASCII.EOT ASCII.CR, i.e., ^T ^D <cr>
```

F.8.2 Restrictions on ELEMENT_TYPE

The following are the restrictions concerning ELEMENT_TYPE:

1. I/O of access types is undefined, although allowable; i.e., the fundamental association between the access variable and its accessed type is ignored.
2. The maximum size of a variant data type is always used.
3. If the size of the element type is exceeded by the physical record length, then during a READ operation the extra data on the physical record is lost. The exception DATA_ERROR is not raised.
4. If the size of the element type exceeds the physical record length during a WRITE operation, the extra data in the element is not transferred to the external file and DATA_ERROR is not raised.
5. SEQUENTIAL_IO and DIRECT_IO cannot be instantiated with unconstrained types. An attempt will lead to a semantic error.
6. I/O operations or composite types containing dynamic array components will not transfer these components because they are not physically contained within the record itself.

F.8.3 TEXT Input/Output (I/O) on a Terminal

A line terminator is detected when either an ASCII.CR is input or output, or when the operating system detects a full buffer. No spanned records with ASCII.NUL are output.

A line terminator followed by a page terminator may be represented as:

```
ASCII.CR
ASCII.FF ASCII.CR
```

if they are issued separately by the user, e.g., NEW_LINE followed by a NEW_PAGE. The same reasoning applies for a line terminator followed by a page terminator, which is then followed by a file terminator.

All text I/O operations are buffered, unless for CHARACTER_IO is specified. This means that physical I/O operations are performed on a line by line basis, as opposed to a character by character basis. For example:

```
put ("Enter Data");
get_line (data, len):
```

will not output the string "Enter Data" until the next put_line or new_line operation is performed.

F.9 UNCHECKED PROGRAMMING

Unchecked programming gives the programmer the ability to circumvent some of the strong typing and elaboration rules of the Ada language. As such, it is the programmer's responsibility to ensure that the guidelines provided in the following sections are followed.

F.9.1 Unchecked Storage Deallocation

The unchecked storage deallocation generic procedure explicitly deallocates the space for a dynamically acquired object.

Restrictions

This procedure frees storage only if:

1. The object being deallocated was the last one allocated of all objects in a given declarative part.
2. All objects in a single chunk of the collection belonging to all access types declared in the same declarative part are deallocated.

F.9.2 Unchecked Type Conversions

The unchecked type conversion generic function permits the user to convert, without type checking, from one type to another. It is the user's responsibility to guarantee that such a conversion preserves the properties of the target type.

Restrictions

The object used as the parameter in the function may not have components which contain dynamic or unconstrained array types.

If the target'size is greater than the source'size, the resulting conversion is unpredictable. If the target'size is less than the source'size, the result is that the left-most bits of the source are placed in the target.

Since unchecked_conversion is implemented as an arbitrary block move, no alignment constraints are necessary on the source or the target operands.

F.10 IMPLEMENTATION-DEPENDENT RESTRICTIONS

1. The main procedure must be parameterless.
2. The source line length must be less than or equal to 80 characters.
3. Due to the source line length, the largest identifier is 80 characters.
4. No more than 9998 lines in a single compilation unit.
5. The maximum number of library units is 9999.
6. The maximum number of bits in an object is 2^{31} -1.
7. The maximum static nesting level is 63.
8. The maximum number of directly imported units of a single compilation unit must not exceed 255.
9. Recompilation of SYSTEM or CALENDAR specification is prohibited.
10. ENTRY'ADDRESS, PACKAGE'ADDRESS and LABEL'ADDRESS are not supported.

F.11 UNCONSTRAINED RECORD REPRESENTATIONS

Objects of an *unconstrained* record type with array components based on the discriminant are allocated with maximal size, based on discriminant'LAST. If this size is greater than 2GB, then the array is allocated with 1024 elements. For example:

```
type DYNAMIC_STRING( LENGTH : NATURAL )
     is record
     STR : STRING( 1 .. LENGTH );
     end record;
```

For this record, the compiler attempts to allocate NATURAL'LAST bytes for the record. Because this is greater than 2GB, the array is instead allocated with 1024 bytes, and a warning message is produced.

APPENDIX C

TEST PARAMETERS

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

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..79 => 'A', 80 => '1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..79 => 'A', 80 => '2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..40 | 42..80 => 'A', 41 => '3')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..40 | 42..80 => 'A', 41 => '4')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..77 => '0', 78..80 => "298")

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be the size of the maximum line length.	(1..74 => '0', 75..80 => '69.0E1")
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1..60 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz" & "$%?@[\]`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	255
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters, or is too long if no invalid characters exist.	"F_#$.BAD"
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character, or is too long if no wild card character exists.	"FILENAME2.BAD"
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in the range of DURATION.	100_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	4_294_967_295.0

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	"ILLEGAL_.FIL"
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	"ILLEGALFILE.NAM"
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2_147_483_648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2_147_483_647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-4_294_967_296.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	15
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	80
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647

TEST PARAMETERS

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

APPENDIX D

WITHDRAWN TESTS

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

- C32114A: An unterminated string literal occurs at line 62.

- B33203C: The reserved word "IS" is misspelled at line 45.

- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.

- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.

- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.

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

- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.

- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.

- B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.

- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

WITHDRAWN TESTS

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

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

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

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

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

- BC3204C: The body of BC3204C0 is missing.