

UNCLASSIFIED

2

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER DTIC FILE COPY	12. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: SYSTEAM, C3 Ada Compiler Version R00-00, Concurrent Computer Corporation 5600 under RTU version 4.0A (host & target) 89110311.10199		5. TYPE OF REPORT & PERIOD COVERED 3 Nov 89 - 1 Dec 90
7. AUTHOR(s) IABG, Ottobrunn, Federal Republic of Germany.		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION AND ADDRESS IABG, Ottobrunn, Federal Republic of Germany.		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) IABG, Ottobrunn, Federal Republic of Germany.		12. REPORT DATE
		13. NUMBER OF PAGES
		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 DTIC ELECTE MAR 15 1990 S D D		
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) SYSTEAM, C3 Ada Compiler, Version R00-00, IABG, West Germany, Concurrent Computer Corporation 5600 under RTU Version 4.0A (host & target), ACVC 1.10		

AD-A219 494

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AVF Control Number: IABG-VSR-047

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: #891103I1.10199
SYSTEM
C3 Ada Compiler Version R00-00
Concurrent Computer Corporation 5600
under RTU version 4.0A

Completion of On-Site Testing:
3rd November 1989

Prepared By:
IABG mbH, Abt SZT
Einsteinstr 20
D8012 Ottobrunn
West Germany

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

90-03-14 036

Ada Compiler Validation Summary Report:

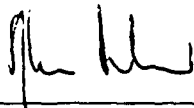
Compiler Name: C3 Ada Compiler
Version R00-00

Certificate Number: #891103I1.10199

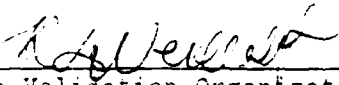
Host and Target: Concurrent Computer Corporation 5600
under RTU version 4.0A

Testing Completed Friday, 3rd November 1989 Using ACVC 1.10

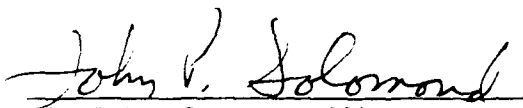
This report has been reviewed and is approved.



IABG mbH, Abt SZT
Dr S. Heilbrunner
Einsteinstr 20
D8012 Ottobrunn
West Germany



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



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

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	1
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1
1.2	USE OF THIS VALIDATION SUMMARY REPORT	2
1.3	REFERENCES	3
1.4	DEFINITION OF TERMS	3
1.5	ACVC TEST CLASSES	4
CHAPTER 2	CONFIGURATION INFORMATION	7
2.1	CONFIGURATION TESTED	7
2.2	IMPLEMENTATION CHARACTERISTICS	7
CHAPTER 3	TEST INFORMATION	14
3.1	TEST RESULTS	14
3.2	SUMMARY OF TEST RESULTS BY CLASS	14
3.3	SUMMARY OF TEST RESULTS BY CHAPTER	15
3.4	WITHDRAWN TESTS	15
3.5	INAPPLICABLE TESTS	15
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS	19
3.7	ADDITIONAL TESTING INFORMATION	
3.7.1	Prevalidation	19
3.7.2	Test Method	20
3.7.3	Test Site	21
APPENDIX A	DECLARATION OF CONFORMANCE	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	
APPENDIX D	COMPILER AND LINKER OPTIONS	

CHAPTER 1

INTRODUCTION

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

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

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

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

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

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but 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 IABG mbH, Abt SZT according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 3rd November 1989 at SYSTEAM KG, Karlsruhe.

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

IABG mbH, Abt. SZT
 Einsteinstr 20
 D8012 Ottobrunn

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

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. 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.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer which executes the code generated by the compiler.
Test	A 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 errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. 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 the run time system to ensure 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.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. 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. In some cases, an implementation may legitimately detect errors during compilation of the test.

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 each test in the ACVC follows 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

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 this 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: C3 Ada Compiler Version R00-00

ACVC Version: 1.10

Certificate Number: #891103I1.10199

Host and Target Computer:

Machine : Concurrent Computer Corporation 5600

Operating System : RTU Version 4.0A

Memory Size : 8 MB

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:

a. Capacities.

- 1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)
- 2) The compiler correctly processes tests containing loop

statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

- 3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)
- 4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

- 1) This implementation supports the additional predefined types `SHORT_INTEGER`, `LONG_FLOAT` and `LONG_LONG_FLOAT` in the package `STANDARD`. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

- 1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)
- 2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)
- 3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)
- 4) No exception is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)
- 5) No exception 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.)
- 6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

- 1) The method used for rounding to integer is round to even. (See tests C46012A..Z (26 tests).)
- 2) The method used for rounding to longest integer is round to even. (See tests C46012A..Z (26 tests).)
- 3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. 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:

This implementation evaluates the `'LENGTH` of each constrained array subtype during elaboration of the type declaration. This causes the declaration of a constrained array subtype with more than `INTEGER'LAST` (which is equal to `SYSTEM.MAX_INT` for this implementation) components to raise `CONSTRAINT_ERROR`. However the optimisation mechanism of this implementation suppresses the evaluation of `'LENGTH` if no object of the array type is declared depending on whether the bounds of the array are static, the visibility of the array type, and the presence of local subprograms. These general remarks apply to points (1) to (5), and (8).

- 1) Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises no exception if the bounds of the array are static. (See test C36003A.)
- 2) `CONSTRAINT_ERROR` is raised when `'LENGTH` is applied to an array type with `INTEGER'LAST + 2` components if the bounds of the array are not static and if the subprogram declaring the array type contains no local subprograms. (See test C36202A.)
- 3) `CONSTRAINT_ERROR` is raised when an array type with `INTEGER'LAST + 2` components is declared if the bounds of the

array are not static and if the subprogram declaring the array type contains a local subprogram. (See test C36202B.)

- 4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises CONSTRAINT_ERROR when the array type is declared if the bounds of the array are not static and if there are objects of the array type. (See test C52103X.)
- 5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the array type is declared if the bounds of the array are not static and if there are objects of the array type. (See test C52104Y.)
- 6) In assigning one-dimensional array types, the expression is not 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.)
- 7) In assigning two-dimensional array types, the expression is not 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.)
- 8) 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 CONSTRAINT_ERROR when the array type is declared if the bounds of the array are not static and if there are objects of the array type. (See test E52103Y.)

f. Discriminated types.

- 1) In assigning record types with discriminants, the expression is not 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.)

g. Aggregates.

- 1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)
- 2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)
- 3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragas.

- 1) The pragma INLINE is supported for functions and procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

- 1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)
- 2) Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)
- 3) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)
- 4) Generic non-library package bodies as subunits can be compiled in separate compilations. (See test CA2009C.)
- 5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs. (See test CA2009F.)
- 6) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)
- 7) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

- 8) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)
- 9) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output.

- 1) The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)
- 2) The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. However this implementation raises USE_ERROR upon creation of a file for unconstrained array types. (See tests AE2101H, EE2401D, and EE2401G.)
- 3) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)
- 4) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)
- 5) Modes IN_FILE, OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)
- 6) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)
- 7) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K AND CE2102Y.)
- 8) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)
- 9) Overwriting to a sequential file truncates the file to the last element written. (See test CE2208B.)
- 10) Temporary sequential files are given names and deleted when closed. (See test CE2108A.)
- 11) Temporary direct files are given names and deleted when closed. (See test CE2108C.)

- 12) Temporary text files are not given names. (See test CE3112A.)
- 13) More than one internal file can be associated with each external permanent (not temporary) file for sequential files when reading only or writing only. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)
- 14) More than one internal file can be associated with each external permanent (not temporary) file for direct files when reading only or writing only. (See tests CE2107F..H (3 tests), CE2110D AND CE2111H.)
- 15) More than one internal file can be associated with each external permanent (not temporary) file for text files when reading only or writing only. (See tests CE3111A..B (2 tests), CE3111D..E (2 tests), CE3114B, and CE3115A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 265 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 159 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 14 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	129	1132	2057	17	27	46	3408
Inapplicable	0	6	258	0	1	0	265
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

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	201	590	566	245	172	99	161	331	137	36	252	325	293	3408	
N/A	11	59	114	3	0	0	5	1	0	0	0	44	28	265	
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44	
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717	

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G	B97102E	C97116A	BC3009B	CD2A62D
CD2A63A	CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B
CD2A66C	CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D
CD2A76A	CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G
CD2A84N	CD2A84M	CD50110	CD2B15C	CD7205C	CD2D11B
CD5007B	ED7004B	ED7005C	ED7005D	ED7006C	ED7006D
CD7105A	CD7203B	CD7204B	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B				

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, 265 tests were inapplicable for the reasons indicated:

- a. The following 159 tests are not applicable because they have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

C241130..Y (11 tests)	C357050..Y (11 tests)
C357060..Y (11 tests)	C357070..Y (11 tests)
C357080..Y (11 tests)	C358020..Z (12 tests)
C452410..Y (11 tests)	C453210..Y (11 tests)
C454210..Y (11 tests)	C455210..Z (12 tests)
C455240..Z (12 tests)	C456210..Z (12 tests)
C456410..Y (11 tests)	C460120..Z (12 tests)

- b. C34007P and C34007S are expected to raise `CONSTRAINT_ERROR`. This implementation optimizes the code at compile time on lines 205 and 221 respectively, thus avoiding the operation which would raise `CONSTRAINT_ERROR` and so no exception is raised.
- c. C41401A is expected to raise `CONSTRAINT_ERROR` for the evaluation of certain attributes, however this implementation derives the values from the subtypes of the prefix at compile time as allowed by 11.6 (7) LRM. Therefore elaboration of the prefix is not involved and `CONSTRAINT_ERROR` is not raised.
- d. The following 16 tests are not applicable because this implementation does not support a predefined type `LONG_INTEGER`:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45614C	C45631C
C45632C	B52004D	C55B07A	B55B09C	B86001W
CD7101F				

- e. C45531M..P (4 tests) and C45532M..P (4 tests) are inapplicable because the value of `SYSTEM.MAX_MANTISSA` is less than 48.
- f. C47004A is expected to raise `CONSTRAINT_ERROR` whilst evaluating the comparison on line 51, but this compiler evaluates the result without invoking the basic operation qualification (as allowed by 11.6 (7) LRM) which would raise `CONSTRAINT_ERROR` and so no exception is raised.
- g. C86001F is not applicable because, for this implementation, the package `TEXT_IO` is dependent upon package `SYSTEM`. These tests recompile package `SYSTEM`, making package `TEXT_IO`, and hence package `REPORT`, obsolete.
- h. B86001X, C45231D, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than `INTEGER` or `SHORT_INTEGER`.
- i. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than `DURATION`.

- j. B86001T and C35702A are not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or LONG_LONG_FLOAT.
- k. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.
- l. CD1009C, CD2A41A, CD2A41B, CD2A41E and CD2A42A..J (10 tests) are not applicable because this implementation imposes restrictions on 'SIZE length clauses for floating point types.
- m. CD2A61I and CD2A61J are not applicable because this implementation imposes restrictions on 'SIZE length clauses for array types.
- n. CD2A71A..D (4 tests), CD2A72A..D (4 tests), CD2A74A..D (4 tests) and CD2A75A..D (4 tests) are not applicable because this implementation imposes restrictions on 'SIZE length clauses for record types.
- o. CD2A84B..I (8 tests), CD2A84K and CD2A84L are not applicable because this implementation imposes restrictions on 'SIZE length clauses for access types.
- p. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.
- q. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.
- r. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.
- s. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.
- t. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.
- u. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.
- v. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.
- w. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.
- x. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

- y. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.
- z. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.
- aa. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.
- ab. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.
- ac. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.
- ad. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.
- ae. CE2107C..D (2 tests) raise USE_ERROR when the function NAME is applied to temporary sequential files, which are not given names.
- af. CE3102E is inapplicable because text file CREATE with IN_FILE mode is supported by this implementation.
- ag. CE3102F is inapplicable because text file RESET is supported by this implementation.
- ah. CE3102G is inapplicable because text file deletion of an external file is supported by this implementation.
- ai. CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.
- aj. CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.
- ak. CE3102K is inapplicable because text file OPEN with OUT_FILE mode is not supported by this implementation.
- al. CE3111B and CE3115A are inapplicable because they assume that a PUT operation writes data to an external file immediately. This implementation uses line buffers; only complete lines are written to an external file by a PUT_LINE operation. Thus attempts to GET data before a PUT_LINE operation in these tests raise END_ERROR.
- am. CE3112B is inapplicable because, for this implementation, temporary text files are not given names.
- an. CE3202A is inapplicable because the underlying operating system does not allow this implementation to support the NAME operation for STANDARD_INPUT and STANDARD_OUTPUT. Thus the calls of the NAME

operation for the standard files in this test raise `USE_ERROR`.

- ao. EE2401D contains instantiations of package `DIRECT_IO` with unconstrained array types. This implementation raises `USE_ERROR` upon creation of such a file.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

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

Modifications were required for 14 tests.

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

B22003A	B24009A	B29001A	B38003A	B38009A	B38009B
B51001A	B91001H	BA1101E	BC2001D	BC2001E	BC3204B
BC3205B	BC3205D				

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the C3 Ada Compiler Version R00-00 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 C3 Ada Compiler Version R00-00 using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Compiler: C3 Ada Compiler Version R00-00

Host and Target computer:

Machine : Concurrent Computer Corporation 5600

Operating System: RTU Version 4.0A

The Concurrent Computer Corporation 5600 machine is based on a MC68020 CPU and uses a MC68881 floating point coprocessor.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded onto a SUN 3/60, and transferred via ETHERNET to the host computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the Concurrent Computer Corporation 5600. Results were transferred to a SUN 3/60, via ETHERNET, and then transferred to a VAX 8350, via DECNET, where they were printed and evaluated.

The compiler was tested using command scripts provided by SYSTEAM KG and reviewed by the validation team. Tests were compiled using the command

```
$mm181/adac -v -l <file name>
```

and linked with the command

```
$mm181/adac -v -m <test name> -o <test name>.OUT
```

Chapter B tests were compiled with the full listing option. A full description of compiler and linker options is given in Appendix E.

Tests were compiled, linked, and executed (as appropriate) using a single computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at SYSTEAM KG, Karlsruhe and was completed on Friday, 3rd November 1989.

DECLARATION OF CONFORMANCE

APPENDIX A

DECLARATION OF CONFORMANCE

SYSTEM KG has submitted the following Declaration
of Conformance concerning the C3 Ada Compiler,
Version R00-00.

DECLARATION OF CONFORMANCE

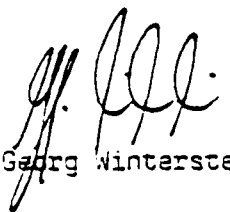
Compiler Implementor: Systeam KG
Ada Validation Facility:
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name: C³Ada Version: R00-00
Base Host Architecture ISA: Concurrent Computer Corporation 5600
Host Operating System: RTU Version 4.0A
Base Target Architecture ISA: Same as Host
Target Operating System: RTU Version 4.0A

Implementor's Declaration

I, the Undersigned, representing Systeam KG, 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 the said compilers in conformance to ANIS/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.


Dr. Georg Winterstein, President

20. October 1989

Date

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 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.


Seetharama Shastri
Senior Manager, System Software Development (Date)

5th Oct 1989

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the C3 Ada Compiler Version R00-00, as described in this Appendix, are provided by SYSTEAM KG. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD are also a part of this Appendix.

4 Predefined Language Environment

The predefined language environment comprises the package standard, the language-defined library units and the implementation-defined library units.

4.1 The Package STANDARD

The specification of the package standard is outlined here; it contains all predefined identifiers of the implementation.

The operations defined for the predefined types are not mentioned here, since they are implicitly declared according to the language rules. Anonymous types (such as *universal_integer*) are not mentioned either.

PACKAGE standard IS

TYPE boolean IS (false, true);

TYPE short_integer IS RANGE - 32_768 .. 32_767;

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

TYPE float IS DIGITS 6 RANGE
- 16#0.FFFF_FF#E32 .. 16#0.FFFF_FF#E32;

TYPE long_float IS DIGITS 15 RANGE
- 16#0.FFFF_FFFF_FFFF_F8#E256 .. 16#0.FFFF_FFFF_FFFF_F8#E256;

TYPE long_long_float IS DIGITS 18 RANGE
- 16#0.FFFF_FFFF_FFFF_FFFF#E4096 ..
16#0.FFFF_FFFF_FFFF_FFFF#E4096;

-- TYPE character IS ... as in [Ada,Appendix C]

-- FOR character USE ... as in [Ada,Appendix C]

-- PACKAGE ascii IS ... as in [Ada,Appendix C]

-- Predefined subtypes and string types ... as in [Ada,Appendix C]

TYPE duration IS DELTA 2#1.0#E-14 RANGE

- 131_072.0 .. 131_071.999_938_964_843_75;

-- The predefined exceptions are as in [Ada,Appendix C]

END standard;

APPENDIX F IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 INTRODUCTION

The Ada programming language definition requires every Ada compilation system to supply an appendix F which contains all implementation-specific aspects of the compiler and run-time system.

F.2 IMPLEMENTATION-DEPENDENT PRAGMAS

The following is a list of all pragmas in C²Ada, predefined as well as implementation-defined:

PRAGMA	IMPLEMENTED	COMMENTS
BYTE_PACK	Yes	Allows arrays and records to be packed to the byte level.
CONTROLLED	No	Not applicable, because no automatic storage reclamation of unreferenced access objects is performed. The complete storage requirement of a collection is released when it passes out of scope.
ELABORATE	Yes	Is handled as defined by the Ada language.
EXTERNAL_NAME	Yes	Allows the association of an external name with a subprogram or statically allocated object.
INLINE	Yes	Subprograms are expanded inline rather than called.
INTERFACE	Yes	Is implemented for the languages C and Assembler.
LIST	Yes	Is handled as defined by the Ada language.
MEMORY_SIZE	No	You cannot change the number of available storage units in the machine configuration, which is defined in package SYSTEM.
OPTIMIZE	No	Optimization of a compilation can only be controlled using the -O option on the compiler command line.
PACK	Yes	Allows arrays and records to be packed to the bit level.
PAGE	Yes	Is handled as defined by the Ada language.
PRIORITY	Yes	Is handled as defined by the Ada language. Priorities between 0 and 255 are supported.
VOLATILE	Yes	Prevents the optimizer from eliminating memory references.

PRAGMA	IMPLEMENTED	COMMENTS
SHARED	No	Not applicable, as every read or write operation on a scalar or access type is a synchronization point for that variable in the CAda implementation.
STORAGE_UNIT	No	You cannot change the number of bits in a storage unit as defined by package SYSTEM.
SUPPRESS	No	Different types of checks cannot be switched on and off for specific objects; however, see SUPPRESS_ALL.
SUPPRESS_ALL	Yes	This pragma prevents the compiler from generating any code to check for errors that may raise CONSTRAINT_ERROR or NUMERIC_ERROR.
SYSTEM_NAME	No	You cannot change the system name as defined in package SYSTEM.

F.2.1 Pragma INLINE Restrictions

Inline expansion of subprogram calls occurs only if the subprogram does not contain any declarations of subprograms, task bodies, generic units, or body stubs. For recursive calls, only the outermost call is expanded. The subprogram must be previously compiled, and if it is a generic instance, it must be previously completed. If for one or more of these reasons the inline expansion is rejected by the compiler, a corresponding warning message will be produced.

F.3 REPRESENTATION CLAUSES

The following subsections describe the restrictions on representation clauses as defined in Chapter 13 of the *Reference Manual for the Ada Programming Language*.

F.3.1 Length Clauses

A length clause specifies the amount of storage to be allocated for objects of a given type. The following is a list of the implementation-dependent attributes:

T'SIZE	must be ≤ 32 for any integer, fixed point, or enumeration type. For any type derived from FLOAT, LONG_FLOAT, or LONG_LONG_FLOAT, the size must be equal to the default value selected by the compiler. These are 32, 64, and 96, respectively. The only value allowed for access types is 32 (the default value). If any of these restrictions is violated, the compiler will report a RESTRICTION error.
T'SORAGE_SIZE	If this length clause is applied to a collection, the exact amount of space specified will be allocated. No dynamic extension of the collection will be performed. If the length clause is not specified, the collection will be extended automatically whenever the allocator new is executed and the collection is full.
T'SORAGE_SIZE	If this length clause is applied to a task type, the specified amount of stack space will be allocated for each task of corresponding type. The value supplied should not be less than 1400. If no length clause is specified for a task type, a default value of 10K bytes is supplied by the compiler. Stack space allocated for a task is never extended automatically at run-time.
T'SMALL	There is no implementation-dependent restriction. In particular, even values of SMALL which are not powers of 2 may be chosen.

F.3.2 Representation Attributes

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

X'ADDRESS	is only supported for objects, subprograms, and interrupt entries. Applied to any other entity, this attribute yields the value <code>SYSTEMADDRESS_ZERO</code> .
X'SIZE	is handled as defined by the Ada language.
R.C'POSITION	is handled as defined by the Ada language.
R.C'FIRST_BIT	is handled as defined by the Ada language.
R.C'LAST_BIT	is handled as defined by the Ada language.
T'SORAGE_SIZE	applied to an access type, this attribute will return the amount of storage currently allocated for the corresponding collection. The returned value may vary as collections are extended dynamically.
T'SORAGE_SIZE	for task types or task objects, this attribute is handled as defined by the Ada language.

F.3.3 Representation Attributes of Real Types

This subsection lists all representation attributes for the floating point types supported:

P'DIGITS	yields the number of decimal digits for the subtype P. The values for the predefined types are 6, 15, and 18 for the types <code>FLOAT</code> , <code>LONG_FLOAT</code> , and <code>LONG_LONG_FLOAT</code> , respectively.
P'MANTISSA	yields the number of binary digits in the mantissa of P. The following table shows the relationship between 'DIGITS and 'MANTISSA:

DIGITS	MANTISSA	DIGITS	MANTISSA
1	5	10	35
2	8	11	38
3	11	12	41
4	15	13	45
5	18	14	48
6	21	15	51
7	25	16	55
8	28	17	58
9	31	18	61

P'EPSILON

yields the absolute value of the difference between the model number 1.0 and the next model number above 1.0 of subtype P:

DIGITS	EPSILON	DIGITS	EPSILON
1	16#0.1#E0	10	16#0.4#E-08
2	16#0.2#E-01	11	16#0.8#E-09
3	16#0.4#E-02	12	16#0.1#E-09
4	16#0.4#E-03	13	16#0.1#E-10
5	16#0.8#E-04	14	16#0.2#E-11
6	16#0.1#E-04	15	16#0.4#E-12
7	16#0.1#E-05	16	16#0.4#E-13
8	16#0.2#E-06	17	16#0.8#E-14
9	16#0.4#E-07	18	16#0.1#E-14

P'EMAX

yields the largest exponent of model numbers for subtype P:

DIGITS	EMAX	DIGITS	EMAX
1	20	10	140
2	32	11	152
3	44	12	164
4	60	13	180
5	72	14	192
6	84	15	204
7	100	16	220
8	112	17	232
9	124	18	244

P'SMALL

yields the smallest model number of subtype P:

DIGITS	SMALL	DIGITS	SMALL
1	16#0.8#E-05	10	16#0.8#E-35
2	16#0.8#E-08	11	16#0.8#E-38
3	16#0.8#E-11	12	16#0.8#E-41
4	16#0.8#E-15	13	16#0.8#E-45
5	16#0.8#E-18	14	16#0.8#E-48
6	16#0.8#E-21	15	16#0.8#E-51
7	16#0.8#E-25	16	16#0.8#E-55
8	16#0.8#E-28	17	16#0.8#E-58
9	16#0.8#E-31	18	16#0.8#E-61

P' LARGE

yields the largest model number of the subtype P:

DIGITS	LARGE
1	16#0.F3#E05
2	16#0.FF#E08
3	16#0.FFE#E11
4	16#0.FFFE#E15
5	16#0.EFFF_C#E18
6	16#0.FFFF_F3#E21
7	16#0.FFFF_FF3#E25
8	16#0.FFFF_FFF#E29
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_FF3#E41
13	16#0.FFFF_FFFF_FFF3#E45
14	16#0.FFFF_FFFF_FFFF#E48
15	16#0.FFFF_FFFF_FFFF_E#E51
16	16#0.FFFF_FFFF_FFFF_FE#E55
17	16#0.FFFF_FFFF_FFFF_FFC#E59
18	16#0.FFFF_FFFF_FFFF_FFF3#E61

The following attributes will return characteristics of the safe numbers and the implementation of the floating point types. For any floating point type sub P, the attributes below will yield the value of the predefined floating point onto which the type P is mapped. For this reason, only the values for the types FLOAT, LONG_FLOAT, and LONG_LONG_FLOAT are given:

ATTRIBUTE	FLOAT	LONG_FLOAT	LONG_LONG_FLOAT
P'SAFE_EMAX	125	1021	16382
P'SAFE_SMALL	16#0.4#E -31	16#0.4#E- 255	16#0.2#E- 4095
P'SAFE_LARGE	16#0.1FFF_FF#E32	16#0.1FFF_FFFF_FFFF_FC#E256	16#0.3FFF_FFFF_FFFF_FFFF#E4096
P'MACHINE_ROUNDS	TRUE	TRUE	TRUE
P'MACHINE_OVERFLOW	TRUE	TRUE	TRUE
P'MACHINE_RADIX	2	2	2
P'MACHINE_MANTISSA	24	53	64
P'MACHINE_EMAX	128	1024	16384
P'MACHINE_EMIN	-125	-1021	-16382

F.3.4 Representation Attributes of Fixed Point Types

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

T'MACHINE_ROUNDS TRUE

T'MACHINE_OVERFLOW TRUE

F.3.5 Enumeration Representation Clauses

The integer codes specified for each enumeration literal has to lie within the range of the largest integer type of the implementation (which is INTEGER). The enumeration table size is determined by the following generic function:

```
generic
  type ENUMERATION_TYPE is (<>);
  function ENUMERATION_TABLE_SIZE return NATURAL;
  function ENUMERATION_TABLE_SIZE return NATURAL is
    RESULT : NATURAL := 0;
begin
  for I in ENUMERATION_TYPE'RANGE
    loop
      declare
        subtype E is ENUMERATION_TYPE range I .. I;
      begin
        RESULT := RESULT + 2 * E'WIDTH;
      end;
    end loop;
  return RESULT;
end ENUMERATION_TABLE_SIZE;
```

F.3.6 Record Representation Clauses

With a record representation clause, the programmer can define the exact layout of a record in memory. Two types of representation clauses are supported: alignment clauses and component clauses.

The value given for an alignment clause must be either 0, 1, 2, or 4. A record with an alignment of 0 may start anywhere in memory. Values other than 0 will force the record to start on a byte address which is a multiple of the specified value. If any value other than 0, 1, 2, or 4 is specified, the compiler will report a **RESTRICTION** error.

For component clauses, the specified range of bits for a component must not be greater than the amount of storage occupied by that component. Gaps within a record may be achieved by not using some bit ranges in the record. Violation of these restrictions will be flagged with a **RESTRICTION** error message by the compiler.

In some cases, the compiler will generate extra components for a record. These cases are:

- If the record contains a variant part and the difference between the smallest and the largest variant is greater than 32 bytes and
 - it has more than one discriminant or
 - the discriminant can hold more than 256 values.

In these cases, an extra component is generated which holds the actual size of the record.

- If the record contains array or record components whose size depend on discriminants. In this case, one extra component is generated for each such component holding its offset in the record relative to the component generated.

The compiler does not generate names for these extra components. Therefore, they cannot be accessed by the Ada program. Also, it is not possible to specify representation clauses for the components generated.

F.4 ADDRESS CLAUSES

Address clauses can be used to allocate an object at a specific location in the computer's address space or to associate a task entry with an interrupt.

Address clauses are supported for objects declared in an object declaration and for task entries. If an address clause is specified for a subprogram, package, or task unit, the compiler will report a **RESTRICTION** error.

For an object, an address clause causes the object to start at the specified location.

6.4.1 Interrupt Entries

Address clauses are supported for task entries. An address clause applied to a task entry enables an operating system signal to initiate an entry call to that entry. The address supplied in an address clause for a task entry must be one of the constants declared in package **SYSTEM** for this purpose.

The interrupt is mapped onto an ordinary entry call. The entry may also be called by an Ada entry call statement. However, it is assumed that there are no entry calls waiting for the same entry when an interrupt occurs. Otherwise, the program is erroneous and behaves as follows:

- If an entry call on behalf of an interrupt is pending, the interrupt pending is lost.
- If any entry call on behalf of an Ada entry call statement is pending, the interrupt entry call takes precedence. The rendezvous on behalf of the interrupt is performed before any other rendezvous.

F.5 PACKAGE SYSTEM

The Ada language definition requires every implementation to supply a package **SYSTEM**. In addition to the declarations required by the language, package **SYSTEM** includes definitions of certain configuration-dependent characteristics. The specification for the C Ada implementation is given below.

package **SYSTEM** is

type **ADDRESS** is private;

ADDRESS_NULL : constant **ADDRESS**;

ADDRESS_ZERO : constant **ADDRESS**;

function "+" (**LEFT** : **ADDRESS**; **RIGHT** : **INTEGER**) return **ADDRESS**;

function "+" (**LEFT** : **INTEGER**; **RIGHT** : **ADDRESS**) return **ADDRESS**;

function "-" (**LEFT** : **ADDRESS**; **RIGHT** : **INTEGER**) return **ADDRESS**;

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

function **SYMBOLIC_ADDRESS** (**SYMBOL** : **STRING**) return **ADDRESS**;

type **NAME** is (**CCUR_MC58K**);

SYSTEM_NAME : constant **NAME** := **CCUR_MC58K**;

STORAGE_UNIT : constant := 8;

MEMORY_SIZE : constant := 2 ** 31;

MIN_INT : constant := - 2 ** 31;

MAX_INT : constant := 2 ** 31 - 1;

MAX_DIGITS : constant := 18;

MAX_MANTISSA : constant := 31;

FINE_DELTA : constant := 2.0 ** (-31);

TICK : constant := 1.0 / 60.0;

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 BYTE is UNSIGNED_TINY_INTEGER;
subtype ADDRESS_RANGE is INTEGER;

subtype PRIORITY is INTEGER range 0 .. 255;

```

```

type SIGNAL is (
    SIGNAL_NULL,
    SIGNAL_HANGUP,
    SIGNAL_INTERRUPT,
    SIGNAL_QUIT,
    SIGNAL_ILLEGAL_INSTRUCTION,
    SIGNAL_TRACE_TRAP,
    SIGNAL_ABORT,
    SIGNAL_EMT_INSTRUCTION,
    SIGNAL_FLOATING_POINT_ERROR,
    SIGNAL_KILL,
    SIGNAL_BUS_ERROR,
    SIGNAL_SEGMENTATION_VIOLATION,
    SIGNAL_BAD_ARGUMENT_TO_SYSTEM_CALL,
    SIGNAL_PIPE_WRITE,
    SIGNAL_ALARM,
    SIGNAL_TERMINATE,
    SIGNAL_USER_1,
    SIGNAL_USER_2,
    SIGNAL_CHILD,
    SIGNAL_POWER_RESTORE,
    SIGNAL_STOP,
    SIGNAL_TERMINAL_STOP,
    SIGNAL_CONTINUE,
    SIGNAL_TERMINAL_INPUT,
    SIGNAL_TERMINAL_OUTPUT,
    SIGNAL_INPUT_CHARACTER,
    SIGNAL_CPU_TIME_LIMIT_EXCEEDED,
    SIGNAL_FILE_SIZE_LIMIT_EXCEEDED,
    SIGNAL_WINDOW_RESIZED,
    SIGNAL_OUT_OF_BAND_DATA_ON_SOCKET,
    SIGNAL_VIRTUAL_TIMER_ALARM,
    SIGNAL_PROFILING_TIMER_ALARM,
    SIGNAL_IO_IS_POSSIBLE);

```

```

-- SIGNAL_NULL_REF              intentionally omitted
SIGNAL_HANGUP_REF               : constant ADDRESS;
SIGNAL_INTERRUPT_REF           : constant ADDRESS;
SIGNAL_QUIT_REF                : constant ADDRESS;
-- SIGNAL_ILLEGAL_INSTRUCTION_REF intentionally omitted
SIGNAL_TRACE_TRAP_REF          : constant ADDRESS;
SIGNAL_ABORT_REF               : constant ADDRESS;
SIGNAL_EMT_INSTRUCTION_REF     : constant ADDRESS;
-- SIGNAL_FLOATING_POINT_ERROR_REF intentionally omitted
-- SIGNAL_KILL_REF              intentionally omitted
-- SIGNAL_BUS_ERROR_REF         intentionally omitted
-- SIGNAL_SEGMENTATION_VIOLATION_REF intentionally omitted
SIGNAL_BAD_ARGUMENT_TO_SYSTEM_CALL_REF : constant ADDRESS;
SIGNAL_PIPE_WRITE_REF          : constant ADDRESS;
-- SIGNAL_ALARM_REF            intentionally omitted
SIGNAL_TERMINATE_REF           : constant ADDRESS;
-- SIGNAL_USER_1_REF           intentionally omitted
SIGNAL_USER_2_REF              : constant ADDRESS;
SIGNAL_CHILD_REF               : constant ADDRESS;
SIGNAL_POWER_RESTORE_REF       : constant ADDRESS;
-- SIGNAL_STOP_REF             intentionally omitted
SIGNAL_TERMINAL_STOP_REF       : constant ADDRESS;

```

```

SIGNAL_CONTINUE_REF          : constant ADDRESS;
SIGNAL_TERMINAL_INPUT_REF    : constant ADDRESS;
SIGNAL_TERMINAL_OUTPUT_REF    : constant ADDRESS;
SIGNAL_INPUT_CHARACTER_REF    : constant ADDRESS;
SIGNAL_CPU_TIME_LIMIT_EXCEEDED_REF : constant ADDRESS;
SIGNAL_FILE_SIZE_LIMIT_EXCEEDED_REF : constant ADDRESS;
SIGNAL_WINDOW_RESIZED_REF     : constant ADDRESS;
SIGNAL_OUT_OF_BAND_DATA_ON_SOCKET_REF : constant ADDRESS;
SIGNAL_VIRTUAL_TIMER_ALARM_REF : constant ADDRESS;
SIGNAL_PROFILING_TIMER_ALARM_REF : constant ADDRESS;
SIGNAL_IO_IS_POSSIBLE_REF     : constant ADDRESS;

type EXCEPTION_ID is new INTEGER;

NO_EXCEPTION_ID      : constant EXCEPTION_ID := 0;

-- Coding of the predefined exceptions:

CONSTRAINT_ERROR_ID : constant EXCEPTION_ID := ... ;
NUMERIC_ERROR_ID    : constant EXCEPTION_ID := ... ;
PROGRAM_ERROR_ID     : constant EXCEPTION_ID := ... ;
STORAGE_ERROR_ID     : constant EXCEPTION_ID := ... ;
TASKING_ERROR_ID     : constant EXCEPTION_ID := ... ;

STATUS_ERROR_ID      : constant EXCEPTION_ID := ... ;
MODE_ERROR_ID        : constant EXCEPTION_ID := ... ;
NAME_ERROR_ID        : constant EXCEPTION_ID := ... ;
USE_ERROR_ID         : constant EXCEPTION_ID := ... ;
DEVICE_ERROR_ID      : constant EXCEPTION_ID := ... ;
END_ERROR_ID         : constant EXCEPTION_ID := ... ;
DATA_ERROR_ID        : constant EXCEPTION_ID := ... ;
LAYOUT_ERROR_ID      : constant EXCEPTION_ID := ... ;

TIME_ERROR_ID        : constant EXCEPTION_ID := ... ;

NO_ERROR_CODE        : constant := 0;

type EXCEPTION_INFORMATION
is record
    EXCP_ID      : EXCEPTION_ID;
    -- Identification of the exception. The codings of
    -- the predefined exceptions are given below.
    CODE_ADDR    : ADDRESS;
    -- Code address where the exception occurred. Depending
    -- on the kind of exception, it may be the address of
    -- the instruction which caused the exception, or
    -- the address of the instruction which would
    -- have been executed if the exception had not occurred.
    SIGNAL       : SYSTEM.SIGNAL;
    -- Signal that caused this exception to be raised,
    -- else signal_null.
end record;

procedure GET_EXCEPTION_INFORMATION
    (EXCP_INFO : out EXCEPTION_INFORMATION);

function INTEGER_TO_ADDRESS(ADDR : ADDRESS_RANGE) return ADDRESS;
function ADDRESS_TO_INTEGER(ADDR : ADDRESS) return ADDRESS_RANGE;
pragma INLINE(INTEGER_TO_ADDRESS, ADDRESS_TO_INTEGER);
-- Conversion between address and integer types.

type EXIT_STATUS is new INTEGER range 0 .. 2**8-1;
NORMAL_EXIT      : constant EXIT_STATUS := 0;

```

```

ERRNO : INTEGER;
for ERRNO use at SYMBOLIC_ADDRESS("_errno");
-- Allows access to the errno set by the last system call, C, or
-- assembler routine call that was made on behalf of the calling
-- task.

procedure EXIT_PROCESS(STATUS : in EXIT_STATUS := NORMAL_EXIT);
-- Terminates the Ada program with the following actions:
-- All Ada tasks are aborted, and the main program exits.
-- All I/O buffers are flushed, and all open files are closed.

private
-- Implementation Defined
end SYSTEM;

```

F.6 TYPE DURATION

DURATION'SMALL is 2^{-14} seconds. This number is the smallest power of two which can represent the number of seconds in a day in longword fixed point number representation.

SYSTEM.TICK is equal to $1.0 / 60.0$ seconds. DURATION'SMALL is significantly smaller than the actual computer clock-tick. Therefore, the accuracy with which you can specify a delay is limited by the actual clock-tick and not by DURATION'SMALL. The following table summarizes the characteristics of the type DURATION:

ATTRIBUTE	VALUE	APPROXIMATE VALUE
DURATION'DELTA	2×10^{-14}	~ 61 μ s
DURATION'SMALL	2×10^{-14}	~ 61 μ s
DURATION'FIRST	-131072.00	~ 36 hrs
DURATION'LAST	131071.99993896484375	~ 36 hrs
DURATION'SIZE	32	

F.7 INTERFACE TO OTHER LANGUAGES

Pragma INTERFACE is implemented for two programming languages: C and Assembler. The pragma has the form:

```
pragma INTERFACE (C, subprogram_name);
```

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

Here, *subprogram_name* is a subprogram declared in the same compilation unit before the pragma.

The only parameter mode supported for subprograms written in the C language is IN. The only types allowed for parameters to subprograms written in the C language are INTEGER, LONG_FLOAT, and SYSTEM_ADDRESS. These restrictions are not checked by the compiler.

Details on interfacing to other languages are given in Chapters 7 and 8.

F.8 INPUT/OUTPUT PACKAGES

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

- the NAME parameter
- the FORM parameter

The NAME parameter must be a legal RTU pathname conforming to the following syntax:

pathname ::= [/] [dirname {/ dirname }/] filename

dirname and *filename* are strings of up to 14 characters length. Any characters except ASCII.NUL, ' ' (blank), and '/' (slash) may be used.

The following is a list of all keywords and possible values for the FORM parameter in alphabetical order.

APPEND => FALSE | TRUE

Only applicable to sequential and text files. If TRUE is specified in an OPEN operation, the file pointer is positioned to the end of the file. This keyword is ignored in a CREATE operation. The file mode must be IN_FILE. The default is APPEND => FALSE.

MODE => numeric_literal

This value specifies the access permission of an external file. It only takes effect in a CREATE operation. It is ignored in an OPEN operation. Access rights can be specified for file owner, group members, and all users. The numeric_literal has to be a three digit octal number. The single bits of this number have the following meaning:

8#400#	read access owner
8#200#	write access owner
8#100#	execute access owner
8#040#	read access group
8#020#	write access group
8#010#	execute access group
8#004#	read access all
8#002#	write access all
8#001#	execute access all

You can specify any sum of the above. The default value is 8#666# which is the maximum access right.

Please note that the RTU operating system will subtract the process's file mode creation mask from the mode you have specified. You can change the file mode creation mask with the RTU command umask (see the *RTU Programming Manual*). For example, if your session has a file mode creation mask of 8#022# and you create a file with mode 8#666#, the file will actually be created with the privileges 8#644#.

RECORD_FORMAT => VARIABLE | FIXED

This parameter is only allowed for sequential files. The default value is VARIABLE.

RECORD_SIZE => numeric_literal

Only applicable to sequential and direct files. It specifies the number of bytes in one record. This parameter is only allowed for files with a fixed record length. When specified in an OPEN operation, it must agree with the corresponding value of the external file. If ELEMENT_TYPE is a constrained type, the

maximum size of ELEMENT_TYPE rounded up to the next byte boundary is selected by default. If ELEMENT_TYPE is an unconstrained array type and you want a fixed record length file, this parameter must be specified.

TRUNCATE => FALSE | TRUE

Only applicable to sequential files. The FILE_MODE must be OUT_FILE. When TRUE is specified in an OPEN operation, the file size is truncated to zero. The previous contents of the file is deleted. If FALSE is specified, the file is not changed initially. If less records than the initial file size are written, old records will remain unchanged in the file. This parameter is ignored for CREATE operations. The default value is TRUE.

F.8.1 Text Input/Output

There are two implementation-dependent types for TEXT_IO: COUNT and FIELD. In C¹Ada, they are implemented as:

```
type COUNT is range 0 .. INTEGER'LAST;  
subtype FIELD is INTEGER range 0 .. 512;
```

The line terminator is implemented by the character ASCII.LF, the page terminator by ASCII.FF. There is no character for the file terminator. End of file is deduced from the file size.

F.9 UNCHECKED PROGRAMMING

F.9.1 Unchecked Storage Deallocation

Unchecked storage deallocation is not supported.

F.9.2 Unchecked Type Conversion

The generic function UNCHECKED_CONVERSION is supported as specified in the *Reference Manual for the Ada Programming Language*, Section 13.10. However, the following restrictions apply:

The generic parameter TARGET must not be an unconstrained array type. If TARGET'SIZE > SOURCE'SIZE, the result of the conversion will be unpredictable. On the other hand, if TARGET'SIZE < SOURCE'SIZE, the left-most bits of the source will be copied to the target.

F.10 IMPLEMENTATION-DEPENDENT RESTRICTIONS

The following is a list of limitations of the compiler:

- The maximum length of a source line is 255 characters.
- A program library may contain no more than 16381 compilation units.
- A single compilation unit may not contain more than 65534 lines of Ada source text. (Depending on the complexity of the code, the actual number of lines acceptable may be considerably smaller than the upper limit.)
- The number of directly imported units for a single compilation unit may not exceed 63. Directly imported units are those referenced by `with` clauses.
- The maximum number of nested `separates` is 100.
- The main program must be a parameterless procedure.
- The maximum length of an identifier is 255 (maximum line length). All characters of an identifier are significant.
- The maximum number of bits of any object is $2^{31} - 1$.
- The maximum length of a file name is 255 characters.
- The maximum length of a listing line is 131 characters.
- The maximum number of errors handled is 1000.
- The maximum number of units that may be named in the pragma `ELABORATE` of a compilation unit is 63.
- The maximum total size for text of unique symbols per compilation is 80000 bytes.
- The maximum parser stack depth is 10000.
- The maximum depth of nested packages is 100.
- The maximum length of a program library name is 242 characters.
- The attribute `ADDRESS` is not supported for packages, labels, and entries that do not have an address clause applied to them.

F.11 UNCONSTRAINED RECORD REPRESENTATION

Objects of an unconstrained record type with array components based on the discriminant are allocated using the discriminant value supplied in the object declaration. However, if no discriminant is supplied in the object declaration, the compiler will choose the maximum possible size. For example:

```
type DYNAMIC_RECORD ( LENGTH : NATURAL := 10 ) is
  record
    STR : STRING ( 1 .. LENGTH );
  end record;

DSTR : DYNAMIC_STRING;
```

For the record DSTR, the Compiler would attempt to allocate NATURAL*LAST bytes. However, this is more than 2 GBytes. As a consequence, CONSTRAINT_ERROR would be raised. On the other hand, the declaration

```
CSTR : DYNAMIC_STRING (80);
```

causes no problems. The compiler would allocate 84 bytes for CSTR.

F.12 TASKING IMPLEMENTATION

The C¹Ada system implements fully pre-emptive and priority-driven tasking. Pre-emptive means that task switches may take place even when the currently running task does not voluntarily give up processor control. This may happen when a task with a high priority is waiting on an external event (time period specified in a delay statement expires). When this event occurs, processor control is passed to the waiting task immediately if it has the highest priority of the tasks ready to run.

The C¹Ada run-time system keeps track of all tasks in two categories: tasks which are ready to run and those that are suspended because they are waiting for something (e.g., a rendezvous to occur or waiting in a delay statement). The tasks ready to run are sorted in a queue by priority (high priorities first). Within one priority, they are sorted in the order in which they entered the "ready" state (tasks waiting longer are served first). Whenever the run-time system needs a task to schedule, the first task in the queue is selected and run.

The accuracy of delay statements is governed by the resolution of the operating system clock which is 1.0/60.0 seconds (SYSTEM.TICK). Although the resolution of the type DURATION is much higher (2⁻¹⁴ seconds), task switches caused by the expiration of a delay can only take place on a clock tick. A task waiting in a delay enters the "ready" state when the next clock tick after its delay period has expired.

Another implementation-dependent aspect of tasking is the stack size of each task. All task objects of a task type with a length clause and all tasks of an anonymous task type have a stack space of 10K bytes. For task types, a length clause may be given. The specified amount of storage space will be allocated for each task object of that type.

In addition to stack space, a task a control block is allocated for each task object. It occupies 250 + 20 * number_of_entries bytes. The task control block is deallocated when the task passes out of scope.

A program is erroneous if any of the following operations can be performed simultaneously by more than one task:

- The allocator new is evaluated for the same collection.
- Input-Output operations are performed on the same external file.

A C¹Ada task is not implemented as an independent operating system process; rather, the whole Ada program is one operating system process.

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
\$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
\$BIG_ID1 An identifier the size of the maximum input line length which is identical to \$BIG_ID2 except for the last character.	254 * 'A' & '1'
\$BIG_ID2 An identifier the size of the maximum input line length which is identical to \$BIG_ID1 except for the last character.	254 * 'A' & '2'
\$BIG_ID3 An identifier the size of the maximum input line length which is identical to \$BIG_ID4 except for a character near the middle.	127 * 'A' & '3' & 127 * 'A'

Name and Meaning	Value
\$BIG_ID4 An identifier the size of the maximum input line length which is identical to \$BIG_ID3 except for a character near the middle.	127 * 'A' & '4' & 127 * '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.	252 * '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.	250 * '0' & "690.0"
\$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	'"' & 127 * 'A' & '"'
\$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	'"' & 127 * 'A' & '1' & '"'
\$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	235 * ' '
\$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
\$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	2_147_483_648
\$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	8

Name and Meaning	Value
\$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	CCUR_MC68K
\$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	2#1.0#E-31
\$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	512
\$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_FIXED_TYPE
\$FLOAT_NAME The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	LONG_LONG_FLOAT
\$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	0.0
\$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	200_000.0
\$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	255
\$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	nodir/file1
\$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	wrondir/file2

TEST PARAMETERS

Name and Meaning	Value
\$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.	-0.0
\$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-200_000.0
\$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	0
\$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
\$MAX_DIGITS Maximum digits supported for floating-point types.	18
\$MAX_IN_LEN Maximum input line length permitted by the implementation.	255
\$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

TEST PARAMETERS

Name and Meaning	Value
\$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.	"2:" & 250 * '0' & "11:"
\$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	"16:" & 248 * '0' & "F.E:"
\$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	'"' & 253 * 'A' & '"'
\$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
\$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	32
\$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	NO_SUCH_TYPE
\$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	CCUR_MC68K
\$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

Name and Meaning	Value
\$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than \$DEFAULT_MEM_SIZE. If there is no other value, then use \$DEFAULT_MEM_SIZE.	2_147_483_648
\$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma STORAGE_UNIT, other than \$DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
\$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than \$DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	CCUR_MC68K
\$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.	32
\$TICK A real literal whose value is SYSTEM.TICK.	1.0/60.0

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 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.

- a. E28005C This test expects that the string "-- TOP OF PAGE. -- 63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.
- b. A39005G This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).
- c. B97102E This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).
- d. C97116A This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING-OF-THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.
- e. BC3009B This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).
- f. CD2A62D This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).
- g. CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests] These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the

parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

- h. CD2A81G, CD2A83G, CD2A84N & M, & CD50110 [5 tests] These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).
- i. CD2B15C & CD7205C These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.
- j. CD2D11B This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.
- k. CD5007B This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).
- l. ED7004B, ED7005C & D, ED7006C & D [5 tests] These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.
- m. CD7105A This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).
- n. CD7203B, & CD7204B These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.
- o. CD7205D This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.
- p. CE2107I This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

- q. CE3111C This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.
- r. CE3301A This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).
- s. CE3411B This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

APPENDIX E

COMPILER AND LINKER OPTIONS

This appendix contains information concerning the compilation and linkage commands used within the command scripts for this validation.

Default Values for Compiler Start Options

- l The option controls the generation of source listing. The default action is not to generate the complete source listing.
- L This option specifies the name of the file or the directory for the listing file. By default, the warning and error messages are directed to `stdout`.
- ld This option specifies `ld` options such as object files, archives. By default, no `ld` options are passed.
- A This option controls the generation of assembly listing for the source being compiled. By default, no assembly listing is generated.
- C This option controls the copying of source file being compiled into the library. By default, source file is not copied into the library.
- S This option causes the suppression of all run-time checks. The default action is to generate code for all run-time checks.
- O This option controls the optimization performed by the compiler. The default option (1) is to perform optimization.
- I This option controls the inline expansion of subprograms that have the `pragma INLINE` specified. The default action (1) is to perform the inline expansion of subprograms for which the `pragma INLINE` has been specified.
- s This option directs the compiler to only perform Syntax analysis. The default is to perform the complete compilation of the supplied source program.
- v This option causes the compiler to produce the compiler version and information messages to be displayed. The default action is to suppress the display of such options.