RE AD-A249 488 PAGE

Public reporting burden for t ~~~ g the time for reviewing instructions, searching existing data sources gathering and maintaining the data needed, and reviewing the ~~~ ther aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Service, Dire ~~~ Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, ~~~

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final: 30 Nov 1990 to 01 Jun 1993

4. TITLE AND SUBTITLE

Validation Summary Report: Concurrent Computer Corporation, C3 Ada Version 1.1v, Concurrent Computer Corporation 6650 with MC6882 Floating Point under RTU Version 5.0C (Host & Target), 901130W1.11110

5. FUNDING NUMBERS

6. AUTHOR(S)

Wright-Patterson AFB, Dayton, OH
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ada Validation Facility, Language Control Facility ASD/SCEL
Bldg. 676, Rm 135
Wright-Patterson AFB, Dayton, OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER

AVF-VSR-417.0891

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Ada Joint Program Office
United States Department of Defense
Pentagon, Rm 3E114
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

DTIC
ELECTE
MAY 0 1 1992
S D
D

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Concurrent Computer Corporation, C3 Ada Version 1.1v, Wright-Patterson AFB, Concurrent Computer Corporation 6650 with MC6882 Floating Point under RTU Version 5.0C (Host & Target), ACVC 1.11.

92-11776

92 4 29 076

14. SUBJECT TERMS

Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFIED	

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 30 November 1990.

Compiler Name and Version: C3 Ada Version 1.1v

Host Computer System: Concurrent Computer Corporation 6650
with MC68882 Floating Point
under RTU Version 5.0C

Target Computer System: Concurrent Computer Corporation 6650
with MC68882 Floating Point
under RTU Version 5.0C

Customer Agreement Number: 90-10-08-CCC

See Section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 901130W1.11110 is awarded to Concurrent Computer Corporation. This certificate expires on 1 March 1993.

This report has been reviewed and is approved.

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH   45433-6503

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA   22311

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

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

Ada   COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 901130W1.11110
Concurrent Computer Corporation
C3 Ada Version 1.1v
Concurrent Computer Corporation 6650
with MC68882 Floating Point
under RTU Version 5.0C
(self-targeted)

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

DECLARATION OF CONFORMANCE

Customer: Concurrent Computer Corporation

Ada Validation Facility: Wright Patterson Air Force Base, Ohio.

ACVC Version: 1.11

Ada Implementation:

Compiler Name and Version: C^3Ada Version: 1.1v

Host Computer System: Concurrent Computer Corporation 6650
with MC68882 Floating point under RTU Version 5.0C

Target Computer System: Same as Host

Customer's Declaration

I, the undersigned, representing Concurrent Computer Corporation, declare that Concurrent Computer Corporation has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration. I declare that Concurrent Computer Corporation is the Implementor of the above implementation and the certificates shall be awarded in the name of Concurrent Computer Corporation's corporate name.

Seetharama Shastry (date) 11/29/90

Seetharama Shastry   (date)
Senior Manager, System Software Development

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada
Validation Procedures [Pro90] against the Ada Standard [Ada83] using the
current Ada Compiler Validation Capability (ACVC). This Validation Summary
Report (VSR) gives an account of the testing of this Ada implementation.
For any technical terms used in this report, the reader is referred to
[Pro90]. A detailed description of the ACVC may be found in the current
ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada
Certification Body 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 implementation has no
nonconformities to the Ada Standard other than those presented. Copies of
this report are available to the public from the AVF which performed this
validation or from:

> National Technical Information Service
> 5285 Port Royal Road
> Springfield VA 22161

Questions regarding this report or the validation test results should be
directed to the AVF which performed this validation or to:

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

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language,
ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program
Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC
contains a collection of test 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. Class B
and class L tests are expected to produce errors at compile time and link
time, respectively.

The executable tests are written in a self-checking manner and produce a
PASSED, FAILED, or NOT APPLICABLE message indicating the result when they
are executed. Three Ada library units, the packages REPORT and SPPRT13,
and the procedure CHECK_FILE are used for this purpose. The package REPORT
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 package SPPRT13 is used by many tests for Chapter 13 of the
Ada Standard. 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. If these units are not operating correctly, validation
testing is discontinued.

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 all violations of
the Ada Standard are detected. Some of the class B tests contain legal Ada
code which must not be flagged illegal by the compiler. This behavior is
also verified.

Class L tests check that an Ada implementation correctly detects violation
of the Ada Standard involving multiple, separately compiled units. Errors
are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by
implementation-specific values — for example, the largest integer. A list
of the values used for this implementation is provided in Appendix A. In
addition to these anticipated test modifications, additional changes may be
required to remove unforeseen conflicts between the tests and
implementation-dependent characteristics. The modifications required for
this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the
AVF. This customization consists of making the modifications described in
the preceding paragraph, removing withdrawn tests (see section 2.1) and,
possibly some inapplicable tests (see Section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of
the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler
: The software and any needed hardware that have to be added
to a given host and target computer system to allow
transformation of Ada programs into executable form and
execution thereof.

Ada Compiler
Validation
Capability
(ACVC)
: The means for testing compliance of Ada implementations,
consisting of the test suite, the support programs, the ACVC
user's guide and the template for the validation summary
report.

Ada
Implementation
: An Ada compiler with its host computer system and its
target computer system.

Ada Joint
Program
Office (AJPO)
: The part of the certification body which provides policy and
guidance for the Ada certification system.

Ada
Validation
Facility (AVF)
: The part of the certification body which carries out the
procedures required to establish the compliance of an Ada
implementation.

Ada
Validation
Organization
(AVO)
: The part of the certification body that provides technical
guidance for operations of the Ada certification system.

Compliance of
an Ada
Implementation
: The ability of the implementation to pass an ACVC version.

Computer
System
: A functional unit, consisting of one or more computers and
associated software, that uses common storage for all or
part of a program and also for all or part of the data
necessary for the execution of the program; executes
user-written or user-designated programs; performs
user-designated data manipulation, including arithmetic
operations and logic operations; and that can execute
programs that modify themselves during execution. A
computer system may be a stand-alone unit or may consist of
several inter-connected units.

Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for
withdrawing each test is available from either the AVO or the AVF. The
publication date for this list of withdrawn tests is 12 October 1990.

E28005C	B28006C	C34006D	B41308B	C43004A	C45114A
C45346A	C45612B	C45651A	C46022A	B49008A	A74006A
C74308A	B83022B	B83022H	B83025B	B83025D	B83026B
B85001L	C83026A	C83041A	C97116A	C98003B	BA2011A
CB7001A	CB7001B	CB7004A	CC1223A	BC1226A	CC1226B
BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A	CD2A21E
CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A	CD2B15C
BD3006A	BD4008A	CD4022A	CD4022D	CD4024B	CD4024C
CD4024D	CD4031A	CD4051D	CD5111A	CD7004C	ED7005D
CD7005E	AD7006A	CD7006E	AD7201A	AD7201E	CD7204B
BD8002A	BD8004C	CD9005A	CD9005B	CDA201E	CE2107I
CE2117A	CE2117B	CE2119B	CE2205B	CE2405A	CE3111C
CE3118A	CE3411B	CE3412B	CE3607B	CE3607C	CE3607D
CE3812A	CE3814A	CE3902B			

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant
for a given Ada implementation. Reasons for a test's inapplicability may
be supported by documents issued by ISO and the AJPO known as Ada
Commentaries and commonly referenced in the format AI-ddddd. For this
implementation, the following tests were determined to be inapplicable for
the reasons indicated; references to Ada Commentaries are included as
appropriate.

The following 159 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113O..Y (11 tests)	C35705O..Y (11 tests)
C35706O..Y (11 tests)	C35707O..Y (11 tests)
C35708O..Y (11 tests)	C35802O..Z (12 tests)
C45241O..Y (11 tests)	C45321O..Y (11 tests)
C45421O..Y (11 tests)	C45521O..Z (12 tests)
C45524O..Z (12 tests)	C45621O..Z (12 tests)
C45641O..Y (11 tests)	C46012O..Z (12 tests)

The following 21 tests check for the predefined type LONG_INTEGER:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45612C	C45613C	C45614C	C45631C	C45632C
B52004D	C55B07A	B55B09C	B86001W	C86006C
CD7101F				

C35702A, C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

C41401A checks that CONSTRAINT_ERROR is raised upon the evaluation of various attribute prefixes; this implementation derives the attribute values from the subtype of the prefix at compilation time, and thus does not evaluate the prefix or raise the exception. (See Section 2.3.)

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater.

C45624A checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 5. For this implementation, MACHINE_OVERFLOWS is TRUE.

C45624B checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 6. For this implementation, MACHINE_OVERFLOWS is TRUE.

C86001F recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete. For this implementation, the package TEXT_IO is dependent upon package SYSTEM.

B86001Y checks for a predefined fixed-point type other than DURATION.

C96005B checks for values of type DURATION'BASE that are outside the range of DURATION. There are no such values for this implementation.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use representation clauses specifying non-default sizes for access types.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions.

The tests listed in the following table are not applicable because the given file operations are not supported for the given combination of mode and file access method.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	————	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO

CE2107C..D (2 tests), CE2107H, and CE2107L apply function NAME to temporary sequential, direct, and text files in an attempt to associate multiple internal files with the same external file; USE_ERROR is raised because temporary files have no name.

CE2108B, CE2108D, and CE3112B use the names of temporary sequential, direct, and text files that were created in other tests in order to check that the temporary files are not accessible after the completion of those tests; for this implementation, temporary files have no name.

CE2203A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for SEQUENTIAL_IO. This implementation does not restrict file capacity.

EE2401D checks whether DIRECT_IO can be instantiated for an element type that is an unconstrained array type; this implementation raises USE_ERROR on the attempt to create a file, because the maximum potential element size exceeds the implementation limit of 2**31 - 1 bits.

CE2403A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for DIRECT_IO. This implementation does not restrict file capacity.

CE3111B and CE3115A associate multiple internal text files with the same external file and attempt to read from one file what was written to the other, which is assumed to be immediately available; this implementation buffers output. (See section 2.3.)

CE3202A expects that function NAME can be applied to the standard input and output files; in this implementation these files have no names, and USE_ERROR is raised. (See section 2.3.)

CE3304A checks that USE_ERROR is raised if a call to SET_LINE_LENGTH or SET_PAGE_LENGTH specifies a value that is inappropriate for the external file. This implementation does not have inappropriate values for either line length or page length.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST. For this implementation, the value of COUNT'LAST is greater than 150000 making the checking of this objective impractical.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 14 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests:

    B29001A   BC2001D   BC2001E   BC3204B   BC3205B   BC3205D

C34007P and C34007S were graded passed by Evaluation Modification as directed by the AVO. These tests include a check that the evaluation of the selector "all" raises CONSTRAINT_ERROR when the value of the object is null. This implementation determines the result of the equality tests at lines 207 and 223, respectively, bas d on the subtype of the object; thus, the selector is not evaluated and no exception is raised, as allowed by LRM 11.6(7). The tests were graded passed given that their only output from Report.Failed was the message "NO EXCEPTION FOR NULL.ALL - 2".

C35713D and B86001Z were processed incorrectly during validation testing: the AVF inadvertantly substituted "NO_SUCH_TYPE" for the macro FLOAT_NAME instead of "LONG_LONG_FLOAT", which is the name of a predefined floating=

point type in this implementation. This mistake in processing was noticed after testing had completed, and the AVF asked the customer to process the tests with the correct macro value; the customer complied with the request and provided the results of correct processing, which showed that the tests were passed instead of inapplicable.

C41401A was graded inapplicable by Evaluation Modification as directed by the AVO. This test checks that the evaluation of attribute prefixes that denote variables of an access type raises CONSTRAINT_ERROR when the value of the variable is null and the attribute is appropriate for an array or task type. This implementation derives the array attribute values from the subtype; thus, the prefix is not evaluated and no exception is raised as allowed by LRM 11.6(7), for the checks at lines 77, 87, 108, 121, 131, 141, 152, 165, and 175.

CE3111B and CE3115A were graded inapplicable by Evaluation Modification as directed by the AVO. The tests assume that output from one internal file is unbuffered and may be immediately read by another file that shares the same external file. This implementation raises END_ERROR on the attempts to read at lines 87 and 101, respectively.

CE3202A was graded inapplicable by Evaluation Modification as directed by the AVO. This test applies function NAME to the standard input file, which in this implementation has no name; USE_ERROR is raised but not handled, so the test is aborted. The AVO ruled that this behavior is acceptable pending any resolution of the issue by the ARG.

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described
adequately by the information given in the initial pages of this report.

For a point of contact for technical information about this Ada
implementation system, see:

> Michael Devlin
> 106 Apple Street
> Tinton Falls NJ 07724
> (201) 758-7531

For a point of contact for sales information about this Ada implementation
system, see:

> Michael Devlin
> 106 Apple Street
> Tinton Falls NJ 07724
> (201) 758-7531

Testing of this Ada implementation was conducted at the customer's site by
a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test
of the customized test suite in accordance with the Ada Programming
Language Standard, whether the test is applicable or inapplicable;
otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was
obtained that conforms to the Ada Programming Language Standard.

Total Number of Applicable Tests	3841
Total Number of Withdrawn Tests	81
Processed Inapplicable Tests	89
Non-Processed I/O Tests	0
Non-Processed Floating-Point Precision Tests	159
Total Number of Inapplicable Tests	248
Total Number of Tests for ACVC 1.11	4170

All I/O tests of the test suite were processed because this implementation supports a file system. The above number of floating-point tests were not processed because they used floating-point precision exceeding that supported by the implementation.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 249 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. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were configured on a 3280 MPS machine and transferred via tar tapes to the host machine.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

The tests were compiled and linked on the host computer system, as appropriate. The executable images were transferred to the target computer system by the communications link described above, and run. The results were captured on the host computer system.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

Option	Effect
-l	Generate a listing file.
-L	Specify the name of the file or a directory for the listing file.
-m	Specify the main program name.
-o	Specify the name of the executable image file.
-v	Cause the compiler to write a version identification and information messages to be displayed.

The listings were printed on a remote system via a remote shell call. Test output, compiler and linker 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.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC.
The meaning and purpose of these parameters are explained in [UG89]. The
parameter values are presented in two tables. The first table lists the
values that are defined in terms of the maximum input-line length, which is
the value for $MAX_IN_LEN—also listed here. These values are expressed
here as Ada string aggregates, where "V" represents the maximum input-line
length.

Macro Parameter	Macro Value
$MAX_IN_LEN	255
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"

$MAX_STRING_LITERAL   '"' & (1..V-2 => 'A') & '"'

The following table lists all of the other macro parameters and their respective values:

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2147283647
$DEFAULT_MEM_SIZE	2147283748
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	CCUR_MC68K
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	SYSTEM.signal_user_2_ref
$ENTRY_ADDRESS1	SYSTEM.signal_child_ref
$ENTRY_ADDRESS2	SYSTEM.signal_power_restore_ref
$FIELD_LAST	512
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_FLOAT_TYPE
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	0.0
$GREATER_THAN_DURATION_BASE_LAST	200000.0
$GREATER_THAN_FLOAT_BASE_LAST	16#1.0#+32
$GREATER_THAN_FLOAT_SAFE_LARGE	16#0.8#E+32

$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
                        0.0

$HIGH_PRIORITY   255

$ILLEGAL_EXTERNAL_FILE_NAME1
                        /nodir/file1

$ILLEGAL_EXTERNAL_FILE_NAME2
                        /wrongdir/file2

$INAPPROPRIATE_LINE_LENGTH
                        -1

$INAPPROPRIATE_PAGE_LENGTH
                        -1

$INCLUDE_PRAGMA1   PRAGMA INCLUDE ("A28006D1.TST")

$INCLUDE_PRAGMA2   PRAGMA INCLUDE ("B28006F1.TST")

$INTEGER_FIRST   -2147483648

$INTEGER_LAST   2147483647

$INTEGER_LAST_PLUS_1   2147483648

$INTERFACE_LANGUAGE   ASSEMBLER

$LESS_THAN_DURATION   -0.0

$LESS_THAN_DURATION_BASE_FIRST
                        -200000.0

$LINE_TERMINATOR   ASCII.LF

$LOW_PRIORITY   0

$MACHINE_CODE_STATEMENT
                        NULL

$MACHINE_CODE_TYPE   NO_SUCH_TYPE

$MANTISSA_DOC   31

$MAX_DIGITS   18

$MAX_INT   2147483647

$MAX_INT_PLUS_1   2147483648

$MIN_INT   -2147483638

MACRO PARAMETERS

$NAME	TINY_INTEGER
$NAME_LIST	CCUR_MC68K
$NAME_SPECIFICATION1	/ben1/mp183/acvc11/chape/X2120A
$NAME_SPECIFICATION2	/ben1/mp183/acvc11/chape/X2120B
$NAME_SPECIFICATION3	/ben1/mp183/acvc11/chape/X3119A
$NEG_BASED_INT	16#FFFFFFFE#
$NEW_MEM_SIZE	2147483648
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	CCUR_MC68K
$PAGE_TERMINATOR	ASCII.LF & ASCII.FF & ASCII.LF
$RECORD_DEFINITION	NEW INTEGER
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	10240
$TICK	1.0/60.0
$VARIABLE_ADDRESS	GET_VARIABLE_ADDRESS
$VARIABLE_ADDRESS1	GET_VARIABLE_ADDRESS1
$VARIABLE_ADDRESS2	GET_VARIABLE_ADDRESS2
$YOUR_PRAGMA	VOLATILE

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler and linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler and linker documentation and not to this report.

Option	Effect
-l	If specified, listing file is generated (listing is not generated by default)
-L	This option specifies the name of the file or a directory for the listing file. (default filename is source_file_name.l)
-m	This option will specify the main program name (a parameterless procedure).
-c	This option will specify the name of the library unit on which the completer should be run. This will cause the compilation of all generic instantiations. This option was not specified.
-o	This option specifies the name of the executable image file. This option is ignored if -m option is not given.
-A	This option instructs the compiler to generate assembly listing. No assembler listing was required. (not specified).
-C	This option instructs the compiler to copy the source file being compiled into the program library. (not specified)

COMPILATION SYSTEM OPTIONS

Option	Effect
-S	This option tells the compiler to suppress all run-time checks. (not specified)
-O	This option controls the optimization level of the compiler. (the compiler performs all optimizations by default)
-I	This option tells the compiler to obey pragma Inline. (default is to obey the pragma)
-s	This option tells the compiler to perform only the syntax analysis. (not specified)
-v	This option causes the compiler to write a version identification and information messages to be displayed. This option was specified. (the default is to suppress such information)

APPENDIX C

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 this Ada implementation, as described in this Appendix,
are provided by the customer. Unless specifically noted otherwise,
references in this Appendix are to compiler documentation and not to this
report. Implementation-specific portions of the package STANDARD, which
are not a part of Appendix F, are:

```
package STANDARD is
    ...

    type INTEGER is range -2147483648 .. 2147483647;

    type TINY_INTEGER is range -128 .. 127;

    type SHORT_INTEGER is range -32768 .. 32767;

    type FLOAT is digits 6 range -16#0.FFFFFF#E32 .. 16#0.FFFFFF#E32;

    type LONG_FLOAT is digits 15
        range -16#0.FFFFFFFFFFFF8#E256 ..
              16#0.FFFFFFFFFFFF8#E256;

    type LONG_LONG_FLOAT is digits 18
        range -16#0.FFFFFFFFFFFFFFFF#E4096 ..
              16#0.FFFFFFFFFFFFFFFF#E4096;

    type DURATION is delta 0.00006103515625
        range -131072.00 .. 131071.99993896484375;

    ...
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 containing all implementation-specific aspects of the compiler and the run-time system.

F.2 IMPLEMENTATION-DEPENDENT PRAGMAS

Table F-1 lists all pragmas in C^3Ada predefined as well as implementation-defined.

TABLE F-1. PRAGMA IMPLEMENTATION SUMMARY

PRAGMA	IMPLEMENTED	COMMENTS
BYTE_PACK	Yes	The elements of an array or record are packed down to a minimal number of bytes.
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	Defines the link-time name of a statically allocated object or of a subprogram.
INLINE	Yes	Is handled as defined by the Ada language, with the following restrictions: The subprogram to be expanded inline must not contain declarations of other subprograms, task bodies, generic units, or body stubs. If the subprogram is called recursively, only the outer call is expanded. The subprogra.. must be previously compiled, and if it is a generic instance, it must be previously completed.
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 adac command line.
PACK	Yes	The elements of an array or record are packed down to a minimal number of bits.
PAGE	Yes	Is handled as defined by the Ada language.
PRIORITY	Yes	Is handled as defined by the Ada language. Priorities in the range 0 .. 255 are supported.

TABLE F-1. PRAGMA IMPLEMENTATION SUMMARY (Continued)

PRAGMA	IMPLEMENTED	COMMENTS
SHARED	Yes	Is handled as defined by the Ada language.
STORAGE_UNIT	No	You cannot change the number of bits in a storage unit which is defined as 8 in package SYSTEM.
SUPPRESS	No	Different types of checks cannot be switched on or off for specific objects; however, see SUPPRESS_ALL.
SUPPRESS_ALL	Yes	This pragma allows the compiler to omit the generation of code to check for errors that may raise CONSTRAINT_ERROR or PROGRAM_ERROR that may be raised due to an elaboration order problem.
SYSTEM_NAME	No	You cannot change the system name which is defined as CCUR_MC68K in package SYSTEM.
VOLATILE	Yes	Specifies that every read or update of a variable causes a reference to the actual memory location for the variable. That is, a local copy of the variable is never made. This is similar to pragma SHARED, except that any variable may be specified. This pragma causes all optimizations on the specific variable to be suppressed.

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 outer call is expanded. The subprogram body 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 - ANSI/MIL-STD-1815A-1983.*

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'STORAGE_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'STORAGE_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 - ANSI/MIL-STD-1815A-1983*, 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 SYSTEM.ADDRESS_ZERO.
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'STORAGE_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'STORAGE_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 FLOAT, LONG_FLOAT, and LONG_LONG_FLOAT, respectively.
P'MANTISSA	yields the number of binary digits in the mantissa of P. Table F-2 shows the relationship between 'DIGITS and 'MANTISSA.

TABLE F-2. P'MANTISSA VALUES

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. Table F-3 summarizes the values for 'EPSILON.

TABLE F-3. P'EPSILON VALUES

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. The values of 'EMAX are given in Table F-4.

TABLE F-4. P'EMAX VALUES

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. The values of ALLaregiven

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. The values of 'LARGE are given in Table F-6.

TABLE F-6. P'LARGE VALUES

DIGITS	LARGE
1	16#0.F8#E05
2	16#0.FF#E08
3	16#0.FFE#E11
4	16#0.FFFE#E15
5	16#0.EEFF_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
16	16#0.FFFF_FFFF_FFFF_FE#E55
17	16#0.FFFF_FFFF_FFFF_FFC#E58
18	16#0.FFFF_FFFF_FFFF_FFF8#E61

The following attributes will return characteristics of the safe numbers and the implementation of the floating point types. For any floating point subtype P, the attributes below will yield the value of the predefined floating point type onto which type P is mapped. Therefore, only the values for the types FLOAT, LONG_FLOAT, and LONG_LONG_FLOAT are given in Table F-7.

TABLE F-7. IMPLEMENTATION-DEPENDENT ATTRIBUTES FOR FLOAT TYPES

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_FFFE#E4096
P'MACHINE_ROUNDS	TRUE	TRUE	TRUE
P'MACHINE_OVERFLOWS	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	is TRUE
T'MACHINE_OVERFLOWS	is TRUE
T'MANTISSA	is in the range 1 .. 31
T'SIZE	is in the range 2 .. 32

F.3.5 Enumeration Representation Clauses

The integer codes specified for each enumeration literal have to lie within the range of the largest integer type of the implementation (which is INTEGER). The maximum number of elements in an enumeration type is limited by the maximum size of the enumeration image table which cannot exceed 65535 bytes. 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'FIRST .. ENUMERATION_TYPE'LAST
    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, you 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 sizes 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.

F.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 pending interrupt 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 C³Ada 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 INTEGER;

    function SYMBOLIC_ADDRESS (SYMBOL : STRING) return ADDRESS;
        -- Returns the address of the external symbol supplied by
        -- SYMBOL, which must be a string literal. This value can
        -- be used in address clauses for objects, providing the
        -- capability of referring to objects declared in C or Assembler.

    type NAME is (CCUR_MC68K);
    SYSTEM_NAME  : constant NAME := CCUR_MC68K;

    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     PROCESS_ID    is new INTEGER;
subtype AST_NUMBER     is INTEGER range 65..128;
type     AST_PRIORITY is range 0..32767;
for      AST_PRIORITY'SIZE use 32;
   -- definitions for AST support.

function TO_C_ROUTINE (ADA_ROUTINE : ADDRESS) return ADDRESS;
   -- Converts an address of an Ada routine to an address
   -- of a C routine that can be used as an AST handler.
   -- This routine allocates some space on the heap
   -- and initializes it with code that changes the
   -- C calling conventions into Ada calling conventions
   -- so that the caller thinks that a C routine is being
   -- called.

type EXCEPTION_ID    is new INTEGER;

NO_EXCEPTION_ID       : constant EXCEPTION_ID := 0;

-- Coding of the predefined exceptions:

CONSTRAINT_ERROR_ID : constant EXCEPTION_ID := 16#0002_0000#;
NUMERIC_ERROR_ID    : constant EXCEPTION_ID := 16#0002_0001#;
PROGRAM_ERROR_ID    : constant EXCEPTION_ID := 16#0002_0002#;
STORAGE_ERROR_ID    : constant EXCEPTION_ID := 16#0002_0003#;
TASKING_ERROR_ID    : constant EXCEPTION_ID := 16#0002_0004#;

STATUS_ERROR_ID       : constant EXCEPTION_ID := 16#0002_0006#;
MODE_ERROR_ID         : constant EXCEPTION_ID := 16#0002_0007#;
NAME_ERROR_ID         : constant EXCEPTION_ID := 16#0002_0008#;
USE_ERROR_ID          : constant EXCEPTION_ID := 16#0002_0009#;
DEVICE_ERROR_ID       : constant EXCEPTION_ID := 16#0002_000A#;
END_ERROR_ID          : constant EXCEPTION_ID := 16#0002_000B#;
DATA_ERROR_ID         : constant EXCEPTION_ID := 16#0002_000C#;
LAYOUT_ERROR_ID       : constant EXCEPTION_ID := 16#0002_000D#;

TIME_ERROR_ID         : constant EXCEPTION_ID := 16#0002_000E#;

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 above.
      CODE_ADDR     : ADDRESS;
         -- Code address where the exception occurred. Depending
         -- on the kind of the exception, it may be the address of
         -- the instruction which caused the exception, or it
         -- may be 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);
   -- This subprogram must only be called from within an exception
   -- handler BEFORE ANY OTHER EXCEPTION IS RAISED. It then returns
   -- the information record about the actually handled exception.
   -- Otherwise, the result is undefined.
```

```
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. Table F-8 summarizes the characteristics of type DURATION:

TABLE F-8. TYPE DURATION

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

F.7 INTERFACE TO OTHER LANGUAGES

The **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 6 and 7.

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 for 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 others
8#002#	write	access all others
8#001#	execute	access all others

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

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 |
                 FIXED

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 C³Ada 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 the sequence ASCII.LF, ASCII.FF, ASCII.LF. 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

The generic function UNCHECKED_DEALLOCATION is supported as specified in the *Reference Manual for the Ada Programming Language - ANSI/MIL-STD-1815A-1983*, Section 13.10.

F.9.2 Unchecked Type Conversion

The generic function UNCHECKED_CONVERSION is supported as specified in the *Reference Manual for the Ada Programming Language - ANSI/MIL-STD-1815A-1983*, 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 implementation-dependent restrictions 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 255. Directly imported units are those referenced by with clauses.
- The maximum number of nested separates is 511.
- 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 pathname 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 255.
- The maximum total size for text of unique symbols per compilation is 300000 bytes.
- The maximum parser stack depth is 10000.
- The maximum depth of nested packages is 511.
- The maximum length of a program library name is 242 characters.

- The amount of statically allocated, non-initialized data in a compilation unit cannot exceed INTEGER'LAST bytes.

- The amount of statically allocated, initialized data in a compilation unit cannot exceed INTEGER'LAST bytes.

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_STRING ( 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 C³Ada 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 (the 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 C³Ada 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 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 are 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 C^3Ada task is not implemented as an independent operating system process; rather, the whole Ada program is one operating system process which does not use threads.