R AD-A242 285

N PAGE

Form Approved
OPM No. 0704-0188

Public reporting burden f
needed, and reviewing th
Headquarters Service, D
Management and Budget

luding the time for reviewing instructions, searching existing data sources gathering and maintaining the data
ny other aspect of this collection of information, including suggestions for reducing this burden, to Washington
ay, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final: 29 Nov 1990 to 01 Jun 1993

4. TITLE AND SUBTITLE
R.R. Software, Inc., IntegrAda 5.1.0 POSIX, Unisys PW/2 386 SCO Unix 3.2 (Host & Target)901129W1.11086

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-434-0891

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
This software available for distribution per Michelle Kay ADA, telecon 11/4/91 mgm

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
R.R. Software, Inc., IntegrAda 5.1.0 POSIX, Wright-Patterson AFB, OH, Unisys PW/2 386 SCO Unix 3.2 (Host & Target),ACVC 1.11

91-15047

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	

NSN 7540-01-280-550

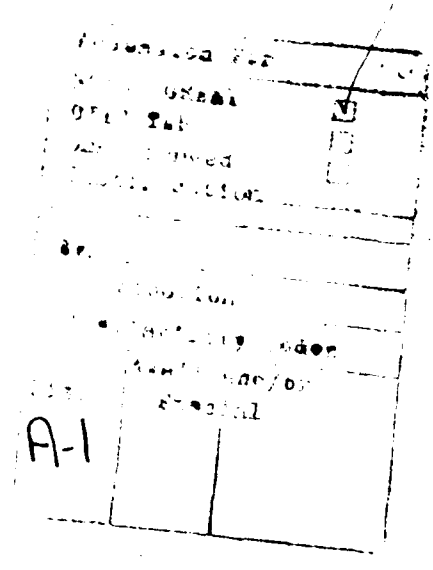
Standard Form 298, (Rev. 2-89)
Prescribed by ANSI Std. 239-128

18

AVF Control Number:AVF-VSR-434-0891
1 August 1991
90-08-02-RRS

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 901129W1.11086
R.R. Software, Inc.
IntegrAda 5.1.0 POSIX
Unisys PW/2 386 SCO Unix 3.2 => Unisys PW/2 386 SCO Unix 3.2

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



Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 29 November 1990.

Compiler Name and Version: IntegrAda 5.1.0 POSIX

Host Computer System: Unisys PW/2 386 (under SCO Unix 3.2)

Target Computer System: Unisys PW/2 386 (under SCO Unix 3.2)

Customer Agreement Number: 90-08-02-RRS

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

As a result of this validation effort, Validation Certificate 901129W1.11086 is awarded to R.R. Software, Inc. This certificate expires on 1 June 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

Declaration of Conformance

Compiler Implementor : R.R. Software, Inc.
Ada Validation Facility : Wright-Patterson AFB, Ohio 45433-6503
Ada Compiler Validation Capability (ACVC) Version : 1.11

Base Configuration

Ada Compiler Name : IntegrAda Version : 5.1.0 POSIX
Host Architecture: Unisys PW/2 386 Host OS & Ver.:SCO Unix 3.2
Target Architecture: Unisys PW/2 386 Target OS & Ver.:SCO Unix3.2

Implementor's Declaration

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

James A. Stewart
Vice President
R.R. Software, Inc.

Nov. 16, 1990
Date

Owner's Declaration

I, the undersigned, representing AETECH, Inc. take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the ANSI/MIL-STD-1815A.

James T. Thomes
President,
AETECH, Inc.

16 Nov. '90
Date

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

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

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

Ada Compiler Validation Capability User's Guide, [UG89] 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 21 November 1990.

E28005C	B28006C	C34006D	C35702A	B41308B	C43004A
C45114A	C45346A	C45612B	C45651A	C46022A	B49008A
A74006A	C74308A	B83022B	B83022H	B83025B	B83025D
B83026B	C83026A	C83041A	B85001L	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	CE3116A	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 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

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

The following 21 tests check for the predefined type SHORT_INTEGER:

C35404B	B36105C	C45231B	C45304B	C45411B
C45412B	C45502B	C45503B	C45504B	C45504E
C45611B	C45613B	C45614B	C45631B	C45632B
B52004E	C55B07B	B55B09D	B86001V	C86006D
CD7101E				

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

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.

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 48 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.

D55A03E..H (4 tests) use 31 or more levels of loop nesting which exceeds the capacity of the compiler.

D56001B uses 65 levels of block nesting which exceeds the capacity of the compiler.

D64005F..G (2 tests) use 10 or more levels of recursive procedure calls nesting which exceeds the capacity of the compiler.

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.

LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F check for pragma INLINE for procedures and functions.

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.

AD9004A uses pragma INTERFACE for overloaded subprograms; this implementation rejects this use due to calling conventions. (See section 2.3.)

CDA201C instantiates Unchecked_Conversion with an array type with a non-static index constraint; this implementation does not support Unchecked_Conversion for types with non-static constraints.

The tests listed in the following table are not applicable because the given file operations are 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

The following 16 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file and one or more are open for writing; USE_ERROR is raised when this association is attempted.

CE2107B..E	CE2107G..H	CE2107L	CD2110B	CE2110D
CE2111D	CE2111H	CE3111B	CE3111D..E	CE3114B
CE3115A				

EE2201D uses instantiations of package SEQUENTIAL_IO with unconstrained array types; this implementation raises USE_ERROR on the attempt to create a file of such type.

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 uses instantiations of package DIRECT_IO with unconstrained array types; this implementation raises USE_ERROR on the attempt to create a file of such type.

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.

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.

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 80 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.

B22003A	B24007A	B24009A	B29001A	B37106A	B51001A
B53003A	B55A01A	B63001A	B63001B	B73004B	B83003B
B83004B	B83004C	B83004D	B83004F	B83030D	B83E01C
B83E01D	B83E01E	B83E01F	B91001H	BA1001A	BA1001B
BA1001C	BA1010A	BA1010D	BA1101A	BA1101E	BA3006A
BA3006B	BA3007B	BA3008A	BA3008B	BA3013A	BC2001D
BC2001E	BC3005B	BD2B03A	BD2D03A	BD4003A	

C85006A..E (5 tests) were graded passed by Test Modification as directed by the AVO. This implementation generates more object code for these tests than it can contain in a single compilation unit. Each of these tests was split into five equivalent subtests.

The tests below were graded passed by Test Modification as directed by the AVO. These tests all use one of the generic support procedures, Length_Check or Enum_Check (in support files LENCHECK.ADA & ENUMCHEK.ADA), which use the generic procedure Unchecked_Conversion. This implementation rejects instantiations of Unchecked_Conversion with array types that have non-static index ranges. The AVO ruled that since this issue was not addressed by AI-00590, which addresses required support for Unchecked_Conversion, and since AI-00590 is considered not binding under ACVC 1.11, the support procedures could be modified to remove the use of Unchecked_Conversion. Lines 40..43, 50, and 56..58 in LENCHECK and lines 42, 43, and 58..63 in ENUMCHEK were commented out.

CD1009A	CD1009I	CD1009M	CD1009V	CD1009W	CD1C03A
CD1C04D	CD2A21A..C	CD2A22J	CD2A23A..B	CD2A24A	CD2A31A..C
CD2A81A	CD3014C	CD3014F	CD3015C	CD3015E..F	CD3015H
CD3015K	CD3022A	CD4061A			

BD4006A was graded inapplicable by Evaluation Modification as directed by the AVO. This test checks that non-static values in component and alignment clauses are rejected; but static alignment values of 8, 16, & 32 are assumed to be supported. This implementation supports only values 1 & 2; it rejects the clauses at lines 42, 48, 58, and 63, which are not marked as errors.

AD9001B was graded passed by Processing Modification as directed by the AVO. This test checks that, if pragma INTERFACE is supported, no bodies are required for interfaced subprograms. This implementation requires that some foreign bodies exist, even if the subprograms are not called. This test was processed in an environment in which implementor-supplied foreign bodies were present.

AD9004A was graded inapplicable by Evaluation Modification as directed by the AVO. This test uses a single INTERFACE pragma for several overloaded procedure and function subprograms; this implementation does not support the pragma in such circumstances due to the calling conventions of the interfaced language, and thus rejects the pragma.

CDA201C was graded inapplicable by Evaluation Modification as directed by the AVO. This test instantiates Unchecked_Conversion with an array type with a non-static index constraint; this implementation does not support Unchecked_Conversion for unconstrained types and so rejects the instantiation. The AVO ruled that various restrictions on Unchecked_Conversion may be accepted for validation under ACVC 1.11, because AI-00590, which addresses Unchecked_Conversion, did not show an ARG consensus at the time of ACVC 1.11's release.

CE2108B, CE2108D, and CE3112B were graded passed by Test Modification as directed by the AVO. These tests, respectively, check that temporary files that were created by (earlier-processed) CE2108A, CE2108C, and CE3112A are not accessible after the completion of those tests. However, these tests also create temporary files. This implementation gives the same names to the temporary files in both the earlier- and later-processed tests of each pair; thus, CE2108B, CE2108D, and CE3112B report failed, as though they have accessed the earlier-created files. The tests were modified to remove the code that created the (later) temporary file; these modified tests were passed. Lines 45..64 were commented out in CE2108B and CE2108D; lines 40..48 were commented out in CE3112B.

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:

Isaac Pentinmaki
R.R. Software, Inc.
P.O. Box 1512
Madison, WI 53701

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

Jim Stewart
R.R. Software, Inc.
P.O. Box 1512
Madison, WI 53701

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].

PROCESSING INFORMATION

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

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system — if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

a) Total Number of Applicable Tests	3773
b) Total Number of Withdrawn Tests	83
c) Processed Inapplicable Tests	113
d) Non-Processed I/O Tests	0
e) Non-Processed Floating-Point Precision Tests	201
f) Total Number of Inapplicable Tests	314
g) Total Number of Tests for ACVC 1.11	4170

3.3 TEST EXECUTION

The diskettes containing the customized test suite (see section 1.3) were taken on-site by the validation team for processing. The contents of the diskettes were installed onto a Northgate 386 with DOS 3.30 and then archived for installation on the actual host computer. The files were restored onto a Unisys 386 with SCO Unix.

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:

The options used for IntegrAda are:

-Q - Quiet error messages - suppresses user prompting on errors. Necessary for running B-Tests; otherwise every error would have to be responded to.

-W - Warnings off - warnings were suppressed mainly because of the many confusing warnings the validation tests produce. Many validation tests have intentional errors (such as an expression which always raises an exception, use of null ranges, unreachable code, etc.). The large volume of warnings produced made it difficult to grade the B-Tests in particular, so they were suppressed.

-BS - Brief Statistics. This was also used to cut the amount of output produced by the compiler during compile time.

All other options used their default values.

Then, all of the non-B-Tests were linked with the options:

-Q - Quiet error messages - suppresses user prompting on errors. Necessary for running L-Tests; otherwise every error would have to be responded to.

-T - Trim unused code - this option directs the linker to remove unused subroutines from the result file. This can make as much as a 30K space saving in the result file.

-B - Brief Statistics. This was also used to cut the amount of output produced by the Linker.

All other options used their default values.

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	200
$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	2
$COUNT_LAST	32_767
$DEFAULT_MEM_SIZE	65536
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	UNIX
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	(0, 16#40#)
$ENTRY_ADDRESS1	(0, 16#05#)
$ENTRY_ADDRESS2	(0, 16#01#)
$FIELD_LAST	32_767
$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	300_000.0
$GREATER_THAN_DURATION_BASE_LAST	1.0E6
$GREATER_THAN_FLOAT_BASE_LAST	1.0E+40
$GREATER_THAN_FLOAT_SAFE_LARGE	1.0E38

```
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
                        1.0E308

$HIGH_PRIORITY          0

$ILLEGAL_EXTERNAL_FILE_NAME1
                        /NODIRECTORY/FILENAME

$ILLEGAL_EXTERNAL_FILE_NAME2
                        <BAD/^^>

$INAPPROPRIATE_LINE_LENGTH
                        -1

$INAPPROPRIATE_PAGE_LENGTH
                        -1

$INCLUDE_PRAGMA1        PRAGMA INCLUDE ("A28006D1.ADA")

$INCLUDE_PRAGMA2        PRAGMA INCLUDE ("B28006E1.ADA")

$INTEGER_FIRST          -32768

$INTEGER_LAST           32767

$INTEGER_LAST_PLUS_1    32768

$INTERFACE_LANGUAGE     C

$LESS_THAN_DURATION     -305_000.0

$LESS_THAN_DURATION_BASE_FIRST
                        -1.0E6

$LINE_TERMINATOR        ASCII.LF

$LOW_PRIORITY           0

$MACHINE_CODE_STATEMENT
                        NULL;

$MACHINE_CODE_TYPE      NO_SUCH_TYPE

$MANTISSA_DOC           31

$MAX_DIGITS             15

$MAX_INT                2147483647

$MAX_INT_PLUS_1         2147483648

$MIN_INT                -214783648
```

MACRO PARAMETERS

$NAME	NO_SUCH_INTEGER_TYPE
$NAME_LIST	UNIX
$NAME_SPECIFICATION1	/usr/ike/x2120a
$NAME_SPECIFICATION2	/usr/ike/x2120b
$NAME_SPECIFICATION3	/usr/ike/x3119a
$NEG_BASED_INT	16#FFFF_FFFF#
$NEW_MEM_SIZE	65536
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	UNIX
$PAGE_TERMINATOR	ASCII.FF
$RECORD_DEFINITION	RECORD NULL; END RECORD;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	512
$TICK	0.01
$VARIABLE_ADDRESS	FCNDECL.Some_Var'Address
$VARIABLE_ADDRESS1	FCNDECL.Some_Var2'Address
$VARIABLE_ADDRESS2	FCNDECL.Some_Var3'Address
$YOUR_PRAGMA	ALL_CHECKS

APPENDIX B

COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation are provided by the customer and can be found in Appendix F, section F.9, page F-14.

LINKER OPTIONS

The linker options of this Ada implementation are provided by the customer and can be found in Appendix F, section F.9, page F-14.

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 -32768 .. 32767;

   type LONG_INTEGER is range -21474838648 .. 2147483647;

   type FLOAT is digits 6 range -((2.0 ** 128) - (2.0 ** 104)) ..
                                ((2.0 ** 128) - (2.0 ** 104);

   type LONG_FLOAT is digits 15 range -((2.0 ** 1024) - (2.0 ** 971)) ..
                                      ((2.0 ** 1024) - (2.0 ** 971));

   type DURATION is delta 0.00025 range -((2.0 ** 31) - 1)/4096.0 ..
                                         ((2.0 ** 31) - 1)/4096.0;

    ..........

end STANDARD;
```

2F Implementation Dependencies

This appendix specifies certain system-dependent characteristics of the IntegrAda, version 5.1.0 386 SCO Unix compiler.

F.1 Implementation Dependent Pragmas

In addition to the required Ada pragmas, IntegrAda also provides several others. Some of these pragmas have a textual range. Such pragmas set some value of importance to the compiler, usually a flag that may be On or Off. The value to be used by the compiler at a given point in a program depends on the parameter of the most recent relevant pragma in the text of the program. For flags, if the parameter is the identifier On, then the flag is on; if the parameter is the identifier Off, then the flag is off; if no such pragma has occurred, then a default value is used.

The range of a pragma - even a pragma that usually has a textual range - may vary if the pragma is not inside a compilation unit. This matters only if you put multiple compilation units in a file. The following rules apply:

1) If a pragma is inside a compilation unit, it affects only that unit.
2) If a pragma is outside a compilation unit, it affects all following compilation units in the compilation.

Certain required Ada pragmas, such as INLINE, would follow different rules; however, as it turns out, IntegrAda ignores all pragmas that would follow different rules.

The following system-dependent pragmas are defined by IntegrAda. Unless otherwise stated, they may occur anywhere that a pragma may occur.

ALL_CHECKS — Takes one of two identifiers On or Off as its argument, and has a textual range. If the argument is Off, then this pragma causes suppression of arithmetic checking (like pragma ARITHCHECK - see below), range checking (like pragma RANGECHECK - see below), storage error checking, and elaboration checking. If the argument is On, then these checks are all performed as usual. Note that pragma ALL_CHECKS does not affect the status of the DEBUG pragma; for the fastest run time code (and the worst run time checking), both ALL_CHECKS and DEBUG should be turned Off and the pragma OPTIMIZE (Time) should be used. Note also that ALL_CHECKS does not affect the status of the

ENUMTAB pragma. Combining check suppression using the pragma ALL_CHECKS and using the pragma SUPPRESS may cause unexpected results; it should not be done. However, ALL_CHECKS may be combined with the IntegrAda pragmas ARITHCHECK and RANGECHECK; whichever relevant pragma has occurred most recently will determine whether a given check is performed. ALL_CHECKS is on by default. Turning any checks off may cause unpredictable results if execution would have caused the corresponding assumption to be violated. Checks should be off only in fully debugged and tested programs. After checks are turned off, full testing should again be done, since any program that handles an exception may expect results that will not occur if no checking is done.

ARITHCHECK — Takes one of the two identifiers On or Off as its argument, and has a textual range. Where ARITHCHECK is on, the compiler is permitted to (and generally does) not generate checks for situations where it is permitted to raise NUMERIC_ERROR; these checks include overflow checking and checking for division by zero. Combining check suppression using the pragma ARITHCHECK and using the pragma SUPPRESS may cause unexpected results; it should not be done. However, ARITHCHECK may be combined with the IntegrAda pragma ALL_CHECKS; whichever pragma has occurred most recently will be effective. ARITHCHECK is on by default. Turning any checks off may cause unpredictable results if execution would have caused the corresponding assumption to be violated. Checks should be off only in fully debugged and tested programs. After checks are turned off, full testing should again be done, since any program that handles an exception may expect results that will not occur if no checking is done.

CLEANUP — Takes an integer literal in the range 0..3 as its argument, and has a textual range. Using this pragma allows the IntegrAda run-time system to be less than meticulous about recovering temporary memory space it uses. This pragma can allow for smaller and faster code, but can be dangerous; certain constructs can cause memory to be used up very quickly. The smaller the parameter, the more danger is permitted. A value of 3 -the default

value-causes the run-time system to be its usual immaculate self. A value of 0 causes no reclamation of temporary space. Values of 1 and 2 allow compromising between "cleanliness" and speed. Using values other than 3 adds some risk of your program running out of memory, especially in loops which contain certain constructs.

DEBUG

Takes one of the two identifiers On or Off as its argument, and has a textual range. This pragma controls the generation of line number code and procedure name code. When DEBUG is on, such code is generated. When DEBUG is off, no line number code or procedure names are generated. This information is used by the walkback which is generated after a run-time error (e.g., an unhandled exception). The walkback is still generated when DEBUG is off, but the line numbers will be incorrect, and no subprogram names will be printed. DEBUG's initial state can be set by the command line; if no explicit option is given, then DEBUG is initially on. Turning DEBUG off saves space, but causes the loss of much of IntegrAda's power in describing run time errors.

Notes:
DEBUG should only be turned off when the program has no errors. The information provided on an error when DEBUG is off is not very useful.

If DEBUG is on at the beginning of a subprogram or package specification, then it must be on at the end of the specification. Conversely, if DEBUG is off at the beginning of such a specification, it must be off at the end. If you want DEBUG to be off for an entire compilation, then you can either put a DEBUG pragma in the context clause of the compilation or you can use the appropriate compiler option.

ENUMTAB

Takes one of the two identifiers On or Off as its argument, and has a textual range. This pragma controls the generation of enumeration tables. Enumeration tables are used for the attributes IMAGE, VALUE, and WIDTH, and hence to input and output enumeration values. The tables are generated when ENUMTAB is on. The state of the ENUMTAB flag is significant only at enumeration type definitions. If this pragma is used to

prevent generation of a type's enumeration tables, then using the three mentioned attributes causes an erroneous program, with unpredictable results; furthermore, the type should not be used as a generic actual discrete type, and in particular TEXT_IO.ENUMERATION_IO should not be instantiated for the type. If the enumeration type is not needed for any of these purposes, the tables, which use a lot of space, are unnecessary. ENUMTAB is on by default.

PAGE_LENGTH — This pragma takes a single integer literal as its argument. It says that a page break should be added to the listing after each occurrence of the given number of lines. The default page length is 32000, so that no page breaks are generated for most programs. Each page starts with a header that looks like the following:

IntegrAda Version 5.1.0 compiling file on date at time

RANGECHECK — Takes one of the two identifiers On or Off as its argument, and has a textual range. Where RANGECHECK is off, the compiler is permitted to (and generally does) not generate checks for situations where it is expected to raise CONSTRAINT_ERROR; these checks include null pointer checking, discriminant checking, index checking, array length checking, and range checking. Combining check suppression using the pragma RANGECHECK and using the pragma SUPPRESS may cause unexpected results; it should not be done. However, RANGECHECK may be combined with the IntegrAda pragma ALL_CHECKS; whichever pragma has occurred most recently will be effective. RANGECHECK is on by default. Turning any checks off may cause unpredictable results if execution would have caused the corresponding assumption to be violated. Checks should be off only in fully debugged and tested programs. After checks are turned off, full testing should again be done, since any program that handles an exception may expect results that will not occur if no checking is done.

OPTIMIZER — Takes one of the identifiers On or Off, or an integer literal, as an argument. This pragma turns optimization on or off, either totally or partially. It has a textual range, except that if

the global optimizer is turned on for any part of a compilation unit, then it is on for the entire compilation unit. If the identifier is On or Off, then IntegrAda's optimizers are turned totally on or totally off, as appropriate. An integer literal as an arguement causes optimization to be turned partially on or off.

The following integer literals are meaningful as an argument to this pragma:

1) Turns check elimination optimizations on.
2) Turns the basic block optimizer on.
3) Turns the global optimizer on. If this is on anywhere in a compilation unit, it will be on everywhere in that unit.
4) Turns peephole optimizations on.
5) Puts the optimizer in 'Space' optimization mode (the default).
6) Puts the optimizer in 'Careful' optimization mode. The can take much longer than 'Quick' optimization, but will find more optimizations.
7) Puts the compiler in 'Fastest alignment' mode. Data objects will be aligned for the fastest performance on the target (unless overridden by rep. clauses). This takes more data space.
51) Turns check elimination optimizations off. Useful for finding uninitialized variables.
52) Turns the basic block optimizer off.
53) Turns the global optimizer off.
54) Turns peephole optimizations off.
55) Puts the optimizer in 'Time' optimization mode.
56) Puts the optimizer in 'Quick' optimization mode. This is faster than 'Careful' optimizations, and often will generate nearly the same code.
57) Put the compiler in 'Smallest alignment' mode. Data is only aligned when required or when the performance penalty is severe. Takes less data space.

Other integer literals will be ignored. In general, this pragma should not be mixed with the OPTIMIZE pragma, since one has a textual arange and the other does not; this can lead to surprising situations. However, the OPTIMIZE

pragma may be used inside a compilation unit for which pragma OPTIMIZER(On) has been listed before the start of the compilation unit.

SYSLIB — This pragma tells the compiler that the current unit is one of the standard IntegrAda system libraries. It takes as a parameter an integer literal in the range 1 .. 15; only the values 1 through 4 are currently used. For example, system library number 2 provides floating point support. Do not use this pragma unless you are writing a package to replace one of the standard IntegrAda system libraries.

VERBOSE — Takes On or Off as its argument, and has a textual range. VERBOSE controls the amount of output on an error. If VERBOSE is on, the two lines preceding the error are printed, with an arrow pointing at the error. If VERBOSE is off, only the line number is printed.

VERBOSE(Off):

```
Line 16 at Position 5
*ERROR* Identifier is not defined
```

VERBOSE(On):

```
  15: if X = 10 then
  16:    Z := 10;
---------^
*ERROR* Identifier is not defined
```

The reason for this option is that an error message with VERBOSE on can take a long time to be generated, especially in a large program. VERBOSE's initial condition can be set by the compiler command line.

Pragma INTERFACE is supported for the language "C". Pragma INTERFACE_NAME can be used to specify a name other than the Ada one as the name of the C function called. INTERFACE_NAME takes two parameters, the Ada subprogram name, and a string representing the C name for the function. Pragma INTERFACE_NAME is provided so that convienient Ada names can be used as appropriate, including operator symbols, and so that foreign language names which are not legal Ada identifiers can be interfaced to. If pragma INTERFACE is used in a program,

Jbind must be used to link it, and it must be linked with the Interface run-time.

Several required Ada pragmas may have surprising effects in IntegrAda. The PRIORITY pragma may only take the value 0, since that is the only value in the range System.Priority. Specifying any OPTIMIZE pragma turns on optimization; otherwise, optimization is only done if specified on the compiler's command line. The SUPPRESS pragma is ignored unless it only has one parameter. Also, the following pragmas are always ignored: CONTROLLED, INLINE, MEMORY_SIZE, PACK, SHARED, STORAGE_UNIT, and SYSTEM_NAME. Pragma CONTROLLED is always ignored because IntegrAda does no automatic garbage collection; thus, the effect of pragma CONTROLLED already applies to all access types. Pragma SHARED is similarly ignored: IntegrAda's non-preemptive task scheduling gives the appropriate effect to all variables. The pragmas INLINE, PACK, and SUPPRESS (with two parameters) all provide recommendations to the compiler; as Ada allows, the recommendations are ignored. The pragmas MEMORY_SIZE, STORAGE_UNIT, and SYSTEM_NAME all attempt to make changes to constants in the System package; in each case, IntegrAda allows only one value, so that the pragma is ignored.

F.2 Implementation Dependent Attributes

IntegrAda does not provide any attributes other than the required Ada attributes.

F.3 Specification of the Package SYSTEM

The package System for IntegrAda has the following definition.

```
package System is

    -- System package for IntegrAda

    -- Types to define type Address.
    type Offset_Type is new Long_Integer;
    type Word is range 0 .. 65536;
    for Word'Size use 16;
    type Address is record
        Offset : Offset_Type;
        Segment : Word;
    end record;
    Function "+" (Left : Address; Right : Offset_Type) Return
                    Address;
    Function "+" (Left : Offset_Type; Right : Address) Return
                    Address;
    Function "-" (Left : Address; Right : Offset_Type) Return
```



```
                    Address;
    Function "-" (Left, Right : Address) Return Offset_Type;

    type Name is (UNIX);

    System_Name : constant Name := UNIX;

    Storage_Unit : constant := 8;
    Memory_Size : constant := 65536;
        -- Note: The actual memory size of a program is
          -- determined dynamically; this is the maximum number
          -- of bytes in the data segment.

    -- System Dependent Named Numbers:
    Min_Int : constant := -2_147_483_648;
    Max_Int : constant :=  2_147_483_647;
    Max_Digits : constant := 15;
    Max_Mantissa : constant := 31;
    Fine_Delta : constant := 2#1.0#E-31;
        -- equivalently, 4.656612873077392578125E-10
    Tick : constant := 0.01; -- Some machines have less
          -- accuracy; for example, the IBM PC actually ticks
          -- about every 0.06 seconds.

    -- Other System Dependent Declarations
    subtype Priority is Integer range 0..0;

    type Byte is range 0 .. 255;
    for Byte'Size use 8;

  end System;
```

The type Byte in the System package corresponds to the 8-bit machine byte. The type Word is a 16-bit Unsigned Integer type, corresponding to a machine word.

F.4 Restrictions on Representation Clauses

A length clause that specifies T'SIZE has the following restrictions:

> If T is a discrete type, or a fixed point type, then the size expression can given any value between 1 and 32 bits (subject, of course, to allowing enough bits for every possible value). Signed and unsigned representations are supported.

If T is a floating point type, sizes of 32 and 64 bits are supported (corresponding to Float and Long_Float respectively).

If T is an array or record type, the expression must give enough room to represent all of the components of the type in their object representation. This can be smaller than the default size of the type.

If T is an access type or task type, the expression must give the default size for T.

A length clause that specifies T'STORAGE_SIZE for an access type is supported.

Any integer value can be specified. STORAGE_ERROR will be raised if the value is larger than available memory; no space will be allocated if the value is less than or equal to zero.

A length clause that specifies T'STORAGE_SIZE for a task type T is supported. Any integer value can be specified. Values smaller than 256 will be rounded up to 256 (the minimum T'Storage_Size), as the Ada standard does not allow raising an exception in this case.

A length clause that specifies T'SMALL for a fixed point type must give a value (subject to the Ada restrictions) in the range

```
2.0 ** (-99) .. 2.0 ** 99,
```

inclusive.

An enumeration representation clause for a type T may give any integer values within the range System.Min_Int .. System.Max_Int. If a size length clause is not given for the type, the type's size is determined from the literals given. (If all of the literals fit in a byte, then Byte'Size is used; similarly for Integer and Long_Integer).

The expression in an alignment clause in a record representation clause must equal 1 or 2 (to specify Byte or Word alignment respectively). The alignment value is respected for all object creations unless another representation clause explicitly overrides it. (By placing a component at a non-aligned address, for example).

A component clause may give any desired storage location. The size of the record is adjusted upward if no representation clause

has been given, and more space is needed for the specified storage location to be obeyed.

The range for specifying the bits may specify any values within the following limitations (assuming enough bits are allowed for any value of the subtype):

> If the component type is a discrete or fixed point type, any value may be specified for the lower bound. The upper bound must satisfy the equation
>
> ```
> UB - (LB - (LB Mod System.STORAGE_UNIT_SIZE)) <= 32.
> ```
>
> If the component type is any other type, the lower bound must satisfy
>
> ```
> LB Mod System.STORAGE_UNIT_SIZE = 0.
> ```
>
> The upper bound must be
>
> ```
> UB := LB + T'Size - 1;
> ```

IntegrAda supports address clauses on most objects. Address clauses are not allowed on parameters, generic formal parameters, and renamed objects. The address given for an object address clause may be any legal value of type System.Address. It will be interpreted as an absolute machine address, using the segment part as a selector if in the protected mode. It is the user's responsibility to ensure that the value given makes sense (i.e., points at memory, does not overlay other objects, etc.) No other address clauses are supported.

F.5 Implementation Defined Names

IntegrAda uses no implementation generated names.

F.6 Address Clause Expressions

The address given for an object address clause may be any legal value of type System.Address. It will be interpreted as an absolute machine address, using the segment part as a selector if in the protected mode. It is the user's responsibility to ensure that the value given makes sense (i.e., points at memory, does not overlay other objects, etc.)

F.7 Unchecked_Conversion Restrictions

We first make the following definitions:

A type or subtype is said to be a simple type or a simple subtype (respectively) if it is a scalar (sub)type, an access (sub)type, a task (sub)type, or if it satisfies the following two conditions:

1) If it is an array type or subtype, then it is constrained and its index constraint is static; and
2) If it is a composite type or subtype, then all of its subcomponents have a simple subtype.

A (sub)type which does not meet these conditions is called non-simple. Discriminated records can be simple; variant records can be simple. However, constraints which depend on discriminants are non-simple (because they are non-static).

IntegrAda imposes the following restriction on instantiations of Unchecked_Conversion: for such an instantiation to be legal, both the source actual subtype and the target actual subtype must be simple subtypes, and they must have the same size.

F.8 Implementation Dependencies of I/O

The syntax of an external file name depends on the operating system being used. Some external files do not really specify disk files; these are called devices. Devices are specified by special file names, and are treated specially by some of the I/O routines.

The syntax of an UNIX filename is:

[path]filename

where "path" is an optional path consisting of directory names, each followed by a foreslash; "filename" is the filename (maximum 14 characters). See your UNIX manual for a complete description. In addition, the following special device names are recognized:

/dev/sti	UNIX standard input. The same as Standard_Input. Input is buffered by lines, and all UNIX line editing characters may be used. Can only be read.
/dev/sto	UNIX standard output. The same as Standard_Output. Can only be written.
/dev/err	UNIX standard error. The output to this device cannot be redirected. Can only be written.
/dev/ekbd	The current terminal input device. Single character input with echoing. Due to the

design of UNIX, this device can be redirected. Can be read and written.

/dev/kbd — The current terminal input device. No character interpretation is performed, and there is no character echo. Again, the input to this device can be redirected, so it does not always refer to the physical keyboard.

The UNIX device files may also be used.

The UNIX I/O system will do a search of the default search path (set by the environment PATH variable) if the following conditions are met:

1) No path is present in the file name; and

2) The name is not that of a device.

Alternatively, you may think of the search being done if the file name does not contain any of the characters ':' or '/'.

The default search path cannot be changed while the program is running, as the path is copied by the IntegrAda program when it starts running.

Note:
Creates will never cause a path search as they must work in the current directory.

Upon normal completion of a program, any open external files are closed. Nevertheless, to provide portability, we recommend explicitly closing any files that are used.

Sharing external files between multiple file objects causes the corresponding external file to be opened multiple times by the operating system. The effects of this are defined by your operating system. This external file sharing is only allowed if all internal files associated with a single external file are opened only for reading (mode In_File), and no internal file is Created. Use_Error is raised if these requirements are violated. A Reset to a writing mode of a file already opened for reading also raise Use_Error if the external file also is shared by another internal file.

Binary I/O of values of access types will give meaningless results and should not be done. Binary I/O of types which are not simple types (see definition in Section F.7, above) will raise Use_Error when the file is opened. Such types require

specification of the block size in the form, a capability which is not yet supported.

The form parameter for Sequential_IO and Direct_IO is always expected to be the null string.

The type Count in the generic package Direct_IO is defined to have the range 0 .. 2_147_483_647.

Ada specifies the existence of special markers called terminators in a text file. IntegrAda defines the line terminator to be <LF> (line feed), with or without an additional <CR> (carriage return). The page terminator is the <FF> (form feed) character; if it is not preceded by a <LF>, a line terminator is also assumed.

The file terminator is the end-of-file returned by the host operating system. If no line and/or page terminator directly precedes the file terminator, they are assumed. The only legal form for text files is "" (the null string). All other forms raise USE_ERROR.

Output of control characters does not affect the layout that Text_IO generates. In particular, output of a <LF> before a New_Page does not suppress the New_Line caused by the New_Page.

The character <LF> is written to represent the line terminator.

The type Text_IO.Count has the range 0 .. 32767; the type Text_IO.Field also has the range 0 .. 32767.

IO_Exceptions.USE_ERROR is raised if something cannot be done because of the external file system; such situations arise when one attempts:

- to create or open an external file for writing when the external file is already open (via a different internal file).
- to create or open an external file when the external file is already open for writing (via a different internal file).
- to reset a file to a writing mode when the external file is already open (via a different internal file).
- to write to a full device (Write, Close);
- to create a file in a full directory (Create);
- to have more files open than the OS allows (Open, Create);
- to open a device with an illegal mode;
- to create, reset, or delete a device;

- to create a file where a protected file (i.e., a directory or read-only file) already exists;
- to delete a protected file;
- to use an illegal form (Open, Create); or
- to open a file for a non-simple type without specifying the block size;
- to open a device for direct I/O.

IO_Exceptions.DEVICE_ERROR is raised if a hardware error other than those covered by USE_ERROR occurs. These situations should never occur, but may on rare occasions. For example, DEVICE_ERROR is raised when:

- a file is not found in a Close or a Delete;
- a seek error occurs on a direct Read or Write; or
- a seek error occurs on a sequential End_Of_File.

The subtypes Standard.Positive and Standard.Natural, used by some I/O routines, have the maximum value 32767.

No package Low_Level_IO is provided.

F.9 Running the compiler and linker

The IntegrAda compiler is invoked using the following format:

```
iada filename {-option}
```

where filename is an UNIX file name (including path) with optional compiler options {-option}.

The compiler options are:

B — Brief error messages. The line in error is not printed (equivalent to turning off pragma VERBOSE).

BS — Brief statistics. Few compiler statistics are printed.

D — Don't generate debugging code (equivalent to turning off pragma DEBUG)

F — Use in-line 80387 instructions for Floating point operations. By default the compiler generates library calls for floating point operations. The 80387 may be used to execute the library calls. A floating point support library is still required, even though this option is used.

L — Create a listing file with name filename.PRN on the same disk as filename. The listing file will be a listing of only the last compilation unit in a file.

Ox — Object code memory model. X is 0 for the 80386 system. Other memory models are not supported. (Since this model 'limits' a program to 4 Gigabytes of Code and 4 Gigabytes of Data, this is not a concern). Memory model 0 is assumed if this option is not given.

Q — Quiet error messages. This option causes the compiler not to wait for the user to interact after an error. In the usual mode, the compiler will prompt the user after each error to ask if the compilation should be aborted. This option is useful if the user wants to take a coffee break while the compiler is working, since all user prompts are suppressed. The errors (if any) will not stay on the screen when this option is used; therefore, the console traffic should be sent to the printer or to a file. Be warned that certain syntax errors can cause the compiler to print many error messages for each and every line in the program.

Rpath — Route the SYM, SRL, and JRL files produced by the compiler to the specified path 'path'. The default is the same path as filename.

Spath — Route Scratch files to specified path.

T — Generate information which allows trimming unused subprograms from the code. This option tells the compiler to generate information which can be used by the remove subprograms from the final code. This option increases the size of the .JRL files produced. We recommend that it be used on reusable libraries of code (like trig. libraries or stack packages) - that is those compilations for which it is likely that some subprograms are not called.

W — Don't print any warning messages. For more control of warning messages, use the following option form (Wx).

Wx — Print only warnings of level less than the specified digit 'x'. The given value of x may be from 1 to 9. The more warnings you are willing to see, the higher the number you should give.

X — Handle eXtra symbol table information. This is for the use of debuggers and other future tools. This option

requires large quantities of memory and disk space, and thus should be avoided if possible.

Z Turn on optimization. This has the same effect as if the pragma OPTIMIZE were set to SPACE throughout your compilation.

The default values for the command line options are:

B Error messages are verbose.
BS Statistics are verbose.
D Debug code is generated.
F Library calls are generated for floating point operations.
L No listing file is generated.
O Memory model 0 is used.
Q The compiler prompts for abort after every error.
T No trimming code is produced.
W All warnings are printed.
X Extra symbol table information is not generated.
Z Optimization is done only where so specified by pragmas.

Leading spaces are disregarded between the filename and the call to Iada. Spaces are otherwise not recommended on the command line. The presence of blanks to separate the options will be ignored.

Examples:

```
iada test-Q-L
iada test.run-W4
iada test
iada test   .run -B -W-L
```

The compiler produces a SYM (SYMbol table information) file when a specification is compiled, and a SRL or JRL (Specification ReLocatable or Janus ReLocatable) file when a body is compiled. To make an executable program, the appropriate SRL and JRL files must be linked (combined) with the run-time libraries. This is accomplished by running IntegrAda binder, JBIND.

The IntegrAda binder is invoked using the following format:

```
jbind filename {-option}
```

Here "filename" is the name of the SRL or JRL file created when the main program was compiled (without the .SRL or .JRL extension) with optional linker options {-option}. The filename usually corresponds to the first ten letters of the name of your main program. See the linker/binder manual for more detailed

directions. We summarize here, however, a few of the most commonly used linking options:

F0 Use software floating point (the default).
F2 Use hardware (80387) floating point.
L Display lots of information about the loading process.
O0 Use memory model 0 (the default); see the description of the /O option in the compiler, above.
Q Use quiet error messages; i.e., don't wait for the user to interact after an error.
B Use brief statistics.
T Trim unused subprograms from the code. This option tells the linker to remove subprograms which are never called from the final output file. This option reduces space usage of the final file by as much as 30K.

Examples:

```
jbind test
jbind test -Q-L
jbind test-L-F2
```

Note that if you do not have a hardware floating point chip, then you generally will not need to use any binder options.

The output of Jbind is a standard UNIX .o file. This file must be linked with the standard UNIX libraries using ld; see your UNIX manual for details.