2

a Enterec

ON PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

AD-A204 779

1.	2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER	
4. Ada Compiler Validation Summary Report: Honeywell Bull, GCOS 8 Ada Compiler, Version 2.1, DPS 8000, DPS 8/70, DPS 90 (Host) DPS 8000, 8/70, 90 (Target)	5. TYPE OF REPORT & PERIOD COVERED 8 June 1988 to 8 June 1988
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) National Bureau of Standards Gaithersburg, MD	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Bureau of Standards Gaithersburg, MD	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) National Bureau of Standards Gaithersburg, MD	15. SECURITY CLASS (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20 If different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

GCOS 8 Ada Compiler, Version 2.1, Honeywell Bull, NBS, DPS 8000, 8/70, 90 under GCOS 8, SR3000 (Host) to DPS 8000, 8/70, under GCOS 8, SR3000 (Target), ACVC 1.9

AVF Control Number: NBS88VHFS505

Ada Compiler
VALIDATION SUMMARY REPORT:

Certificate Number: 880608S1.09144
Honeywell Bull
GCOS 8 Ada Compiler, Version 2.1
DPS 8000, DPS 8/70, DPS 90

Completion of On-Site Testing:
8 June 1988

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

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: GCOS 8 Ada Compiler, Version 2.1

Certificate Number: 880608S1.09144

Host:	Target:
DPS 8000, 8/70, 90 under	DPS 8000, 8/70, 90 under
GCOS 8	GCOS 8
SR3000	SR3000

Testing Completed 8 June 1988 Using ACVC 1.9

This report has been reviewed and is approved.

[illegible]

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899

[illegible]

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

[illegible]

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

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

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

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

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

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

> To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
>
> To attempt to identify any unsupported language constructs required by the Ada Standard
>
> To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by the National Bureau of Standards according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was completed 8 June 1988, at Honeywell Bull Corporation, Phoenix, Arizona.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

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

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

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

or from:

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

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

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

1.3 REFERENCES

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

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

3. Ada Compiler Validation Capability Implementers' Guide., December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Language Maintenance	The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and Panel possible changes to the ANSI/MIL-STD for Ada.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	An Ada program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

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

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

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

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

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

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

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

Two library units, the package REPORT and the procedure CHECK FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of

REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: GCOS 8 Ada Compiler, Version 2.1

ACVC Version: 1.9

Certificate Number:		880608S1.09144
Host Computer:		
	Machine:	DPS 8000, 8/70, 90
	Operating System:	GCOS 8 SR3000
	Memory Size:	each machine has 32 Megawords
Target Computer:		
	Machine:	DPS 8000, 8/70, 90
	Operating System:	GCOS 8 SR3000
	Memory Size:	each machine has 32 Megawords
Communications Network:		none used

2.2 IMPLEMENTATION CHARACTERISTICS

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

- Capacities.

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

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types LONG_INTEGER, LONG_FLOAT, and in the package STANDARD. (See tests B86001BC and B86001D.)

- Based literals.

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

- Expression evaluation.

  Apparently all default initialization expressions or record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with less precision than the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Apparently NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

- Rounding.

The method used for rounding to integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

- Array types.

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

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises CONSTRAINT ERROR. (See test C36003A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

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

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

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

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

- Discriminated types.

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

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

- Aggregates.

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

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

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

- Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is not supported, then the implementation must reject it.

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

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

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

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

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

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

Length clauses with SMALL specifications are not supported. (See tests A39005E and C87B62C.)

Record representation clauses are not supported. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are not supported. (See test C87B62A.)

- Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

There are strings which are illegal external file names for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102C and CE2102H.)

Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D and CE2102E.)

Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I, and CE2102J.)

RESET and DELETE are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2106A and CE2106B.)

Overwriting to a sequential file truncates the file to last element written. (See test CE2208B.)

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

Temporary sequestial files are not given names. Temporary direct files are not given names. (See test CE2108A and CE2108C.)

- Generics.

Generic subprogram declarations and bodies cannot compiled in separate compilations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies cannot be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 28 tests had been withdrawn because of test errors. The AVF determined that 231 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing. Modifications to the code, processing, or grading for 68 tests (72 files) were required to successfully demonstrate the test objective. (See section 3.6.)

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

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	106	1046	1632	17	16	46	2863
Inapplicable	4	5	221	0	1	0	231
Withdrawn	3	2	21	0	2	0	28
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	192	503	561	245	165	98	140	326	135	36	232	3	227	2863
Inapplicable	12	69	113	3	0	0	3	1	2	0	2	0	26	231
Withdrawn	2	14	3	0	1	1	2	0	0	0	2	1	2	28
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 28 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C	C35904A
C35904B	C35A03E	C35A03R	C37213H	C37213J	C37215C
C37215E	C37215G	C37215H	C38102C	C41402A	C45332A
C45614C	E66001D	A74106C	C85018B	C87B04B	CC1311B
BC3105A	AD1A01A	CE2401H	CE3208A		

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

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 231 test were inapplicable for the reasons indicated:

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

C35702A uses SHORT_FLOAT which is not supported by this implementation.

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

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

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

The following (14) tests use SHORT_INTEGER, which is not supported by this compiler.

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	

C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.

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

B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

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

CA2009C requires a generic non-library package body be compiled as a subunit in a separate file from its specification. This implementation is not supported.

CA2009F requires a generic non-library subprogram body be compiled as a subunit in a separate file from its specification. This implementation is not supported.

BC3204C and BC3205D require a generic library package body be compiled in a different file from its specification. This implementation is not supported.

AE2101H and EE2401D use instantiations of package DIRECT_IO with unconstrained array types. These instantiations are rejected by this compiler.

CE2105A and CE3109A attempt to CREATE files with mode IN_FILE; this implementation does not support such an operation.

CE2107A..I (9 tests), CE2110B, CE2111D, CE2111H, CE3111A..E (5 tests), CE3114B, and CE3115A are inapplicable because multiple internal files cannot be associated with the same external file. The proper exception is raised when multiple access is attempted.

CE2108A, CE2108C and CE3112A require names for temport files; this implementation does not give temporary files names.

The following 173 tests require a floating-point accuracy that exceeds the maximum of 17 digits supported by this implementation:

C24113N..Y (12 tests)	C35705N..Y (12 tests)
C35706N..Y (12 tests)	C35707N..Y (12 tests)
C35708N..Y (12 tests)	C35802N..Z (13 tests)
C45241N..Y (12 tests)	C45321N..Y (12 tests)
C45421N..Y (12 tests)	C45521N..Z (13 tests)
C45524N..Z (13 tests)	C45621N..Z (13 tests)
C45641N..Y (12 tests)	C46012N..Z (13 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

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

Modifications were required for 68 Class B tests (72 files).

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

B22003A	B26001A	B26002A	B26005A	B28001D
B29001A	B2A003A	B2A003B	B2A003C	B33301A
B35101A	B37106A	B37301A	B37302A	B38003A
B38003B	B38009A	B38009B	B51001A	B53009A
B54A01C	B54A01J	B55A01A	B61001C	B61001D
B61001F	B61001H	B61001I	B61001M	B61001R
B61001W	B67001A	B67001C	B67001D	B91001A
B91002A..L (12 tests)			B95030A	B95061A
B95061F	B95061G	B95077A	B97101A	B97101E
B97102A	B97103E	B97104G	BA1101B0..4 (5 tests)	
BC1109A	BC1109C	BC1109D	BC1202A	BC1202B
BC1202E	BC1202F	BC1202G	BC2001D	BC2001E

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the GCOS 8 Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavicr on all inapplicable tests.

3.7.2 Test Method

Testing of the GCOS 8 Ada Compiler using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a DPS 8000, DPS 8/70 and DPS 90 host operating under GCOS 8, Version SR30000 and a DPS 8000, DPS 8/70 and DPS 90 target operating under GCOS 8, Version SR3000.

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on-site after the magnetic tape was loaded. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape. The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the DPS 8000, DPS 8/70 and DPS 90, and all executable tests were run on the same system. Object files were linked on the host computer, and executable images were run on the same system.

The compiler was tested using command scripts provided by Honeywell Bull Corporation and reviewed by the validation team. The compiler was tested using all default option settings without exception.

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

3.7.3 Test Site

Testing was conducted at Honeywell Bull, Phoenix, Arizona and was completed on 8 June 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

Compliance Statement

Base Configuration:

Compiler: GCOS 8 Ada Compiler, Version 2.1

Test Suite: Ada Compiler Validation Capability, Version 1.9

Host Computer:

Machines:	DPS 8000, DPS 8/70, DPS 90
Operating System:	GCOS 8, Version SR3000

Target Computer:

Machines:	DPS 8000, DPS 8/70, DPS 90
Operating System:	GCOS 8, Version SR3000

Honeywell Bull Inc. has made no deliberate extensions to the Ada Language Standard.

Honeywell Bull Inc. agrees to the public disclosure of this report.

Honeywell Bull Inc. agrees to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

R. Edward Kearns Date: Feb 19, 1988

Honeywell Bull Inc.
R. Edward Kearns, Manager
Advanced Compiler Development

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the GCOS 8 Ada Compiler, Version 2.1 are described in the following sections which discuss topics in Appendix F of the Ada Standard. Implementation- specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is


     type INTEGER is range -34359738368 .. 34359738367;

     type LONG_INTEGER is range -2361183241434822606848 ..
                                 2361183241434822606847;

     type FLOAT is digits 6 range -16#0.1#E128 ..
                                 16#0.FFFFFFE#E127;

     type LONG_FLOAT is digits 17 range -16#0.1#E128 ..
                                 16#0.FFFF_FFFF_FFFF_FFFE#E127;

     type DURATION is delta 0.000016  range -1099511627776.0 ..
                                 775.99999904632568359375;


end STANDARD;
```

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

This appendix describes the implementation-dependent characteristics of the GCOS 8 Ada Compiler. This is the Appendix F referred to in the *Ada Reference Manual*.

- o Implementation-Dependent Pragmas
- o Package SYSTEM
- o Restrictions on Representation Clauses
  - Type Representation Clauses
  - Address Clauses
- o Unchecked Conversion
- o Input/Output
  - Introduction
  - Implementation Choices
  - Form Parameter

Topics (1), (3), (4), (6), (7), and (8) given in the Appendix F frame of the *Ada Reference Manual* (ANSI/MIL-STD-1815A) are discussed below. Topics (2) and (5) are not relevant since implementation-dependent attributes and implementation-generated names for implementation-dependent components are not supported by the compiler.

Implementation-Dependent Pragmas

See Section 4 for a description of the implementation-dependent pragmas, SINGLE_SEGMENT_DATA, MULTI_SEGMENT_DATA, and INTERFACE_SPELLING.

Package SYSTEM

See Appendix E for a listing of the package specification for SYSTEM.

Restrictions on Representation Clauses

Type Representation Clauses

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

Length Clause

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

Enumeration Representation Clause

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

Record Representation Clause

Alignment clauses are not supported. A component clause is allowed if, and only if, either of these statements is true:

- o The component type is a discrete type different from LONG_INTEGER.
- o The component type is an array type with a discrete element type different from LONG_INTEGER.

No component clause is allowed if the component type is not covered by the above two inclusions. If the record type contains components not covered by a component clause, they are allocated consecutively after the component with the highest AT value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler. The size of a component (or the size of each element of a component if the component is an array) must be 1, 6, 9, 18, or 36 bits.

No names denoting implementation-dependent components are generated.

Address Clauses

Not supported.

Unchecked Conversion

Unchecked conversion is allowed only between values of the same size. In this context the size of an array is equal to that of two access values and the size of a packed array is equal to that of two access values and an integer.

Input/Output

Introduction

Although variations from the syntax and semantics of the Ada language are not permitted, certain choices made in the implementation of the basic input/output system are visible to the Ada programmer. These choices, including the operation of the FORM parameter, are documented here.

Other implementation choices affect the basic file mapping and the interfaces to the operating system and are documented in Section 4.

Implementation Choices

- An attempt to CREATE an IN_FILE will raise USE_ERROR.
- A RESET to OUT_FILE, on a sequential or text file, empties the file.
- Two internal files may not be associated with the same external file simultaneously.
- Temporary files accessed by a batch program are not named. Use of the function NAME results in a USE_ERROR.
- An attempt to open a 'busy' file will result in the I/O exception, STATUS_ERROR.

Form Parameter

The FORM parameter can be the concatenation of any of the following strings separated by spaces:

"-FILCOD XX"	This associates the internal file with the external file designated by file code XX where XX is a valid file code supplied by JCL. A file code takes precedence over a NAME string. File code P* is treated specially. It is written as media code 7 (ASCII print) and report code 73. Any horizontal tab character (HT, ASCII 9) is converted to a space. A single space control is appended to the end of every line[1]. A page eject is placed at the end of every page[2]. These control characters must be taken into account if a disk file is to be reread.
"-MEDCOD N"	This specifies the media code of the external file where N is a valid media code.
"-APPEND"	This is only applicable to sequential and text files; it results in a USE_ERROR if it is applied to direct files. On opening a file with mode OUT_FILE, it specifies positioning at end-of-file so that writes will append rather than overwrite the file. It has no effect on creates or on opening files with mode IN_FILE.

[1] The space control sequence is a vertical tab character (VT, ASCII 11) followed by a byte (character) containing a decimal 1.

[2] The page eject sequence is a form feed character (FF, ASCII 12) followed by a byte (character) containing a decimal 0.

Pragma CONTROLLED

This pragma has no effect, as no automatic storage reclamation is performed before the point allowed by the pragma.

Pragma ELABORATE

As in *Ada Reference Manual.*

Pragma INLINE

This pragma is obeyed by the compiler whenever possible. If the argument is an overloaded subprogram name, the INLINE pragma has an effect on all subprograms with the specified name that appear in the same declarative part as the pragma.

Pragma INLINE causes inline expansion, except in the following cases:

- o The whole body of the subprogram for which inline expansion is requested has not been found (this also covers recursive calls).
- o The subprogram call appears in an expression on which a conformance check may be applied; for example, in a subprogram specification, in a discriminant part, or in a formal part of an entry declaration or accept statement.

A warning is given if inline expansion is not achieved.

Pragma INTERFACE

This pragma is supported for subprograms written in other languages. The languages supported by this interface are COBOL-85, C, and GMAPV assembly language. There are two forms of the GMAPV interface: The first form follows the calling convention of COBOL-85 and C. The second form follows the calling convention of Ada. (Further information about the assembly language conventions can be found elsewhere in this chapter.) The language names used in the pragma are "COBOL_85", "C", "GMAPV", and "GMAPV_ADA", respectively.

The subprogram name must be a legal Ada identifier, and it represents the name of the foreign subprogram. If the foreign subprogram name is not a legal Ada identifier, the pragma INTERFACE_SPELLING (see definition below) must be used.

Example:

```
pragma INTERFACE( COBOL_85, sort);
```

This example designates a subprogram, whose externally-known name is "sort", which will be invoked via the COBOL-85 calling convention.

There are some restrictions which apply to foreign subprograms which use the COBOL-85 and C calling convention:

- The subprogram must be a procedure and not a function. Function return values are not supported.
- Neither COBOL-85 nor C subprograms support the Ada parameter mode OUT. To achieve the desired result, it is necessary to use properly-declared access types. Parameters to foreign subprograms in these languages should be of mode IN. It is possible for a GMAPV subprogram to handle OUT (or IN OUT) parameters (with careful coding).
- No conversion of parameters is performed by the Ada compiler. It is necessary that the parameter types be compatible between Ada and the other language. Integer and access types are strongly recommended.

Pragma INTERFACE_SPELLING

This implementation-defined pragma is used to specify the string literal which represents the name of an externally known subprogram. The name of a subprogram written in another language may be an illegal Ada identifier. In that case, this pragma can be used to establish a correspondence between the external name and the Ada identifier specified in a pragma INTERFACE. The form of this pragma is shown below:

```
    pragma INTERFACE_SPELLING( subprogram_name, string_literal);
```

Example:

```
    pragma INTERFACE_SPELLING( sort, "A.SORT");
```

The identifier "sort" will be used within the Ada program to refer to the external subprogram. The compiler, however, will generate external references to "A.SORT".

This pragma is allowed at the place of a declarative item and must apply to a subprogram declared by an earlier pragma INTERFACE.

Pragma LIST

As in *Ada Reference Manual.*

Pragma MEMORY_SIZE

Not supported.

Pragma MULTI_SEGMENT_DATA

This implementation-defined pragma affects the method of storage access, and hence, the amount of static storage which is accessible to this compilation unit. This pragma allows the compilation unit to access over 256K words of static storage. (The default limit on static storage is 32K words.) However, the access method is more costly when this pragma is specified. The form of the pragma is shown below:

```
pragma MULTI_SEGMENT_DATA;
```

This pragma is allowed anywhere a pragma is allowed. It will affect the current compilation unit and any subsequent units in the same compilation.

Pragma OPTIMIZE

This pragma has no effect.

Pragma PACK

This pragma affects only array types with a discrete type (except LONG_INTEGER) as component type. The components of packed arrays are packed into the smallest possible fraction of a word, where a fraction of a word consists of 1, 6, 9, or 18 bits.

Pragma PAGE

As in *Ada Reference Manual.*

Pragma PRIORITY

As in *Ada Reference Manual.*

Pragma SHARED

Not supported.

Pragma SINGLE_SEGMENT_DATA

This implementation-defined pragma affects the method of storage access, and hence, the amount of static storage which is accessible to this compilation unit. This pragma allows the compilation unit to access up to 256K words of static storage. (The default limit on static storage is 32K words.) However, the access method is more costly when this pragma is specified. The form of the pragma is shown below:

```
pragma SINGLE_SEGMENT_DATA;
```

This pragma is allowed anywhere a pragma is allowed. It will affect the current compilation unit and any subsequent units in the same compilation.

Pragma STORAGE_UNIT

Not supported.

Pragma SUPPRESS

The implementation supports only the following form of the pragma:

```
PRAGMA SUPPRESS (identifier);
```

Thus, it is not possible to restrict the omission of a certain check to a specified name.

Pragma SYSTEM_NAME

Not supported.

Address Clauses

Not supported.

Machine Code Insertions

Not supported.

SYSTEM

```
package SYSTEM is

  type ADDRESS is access INTEGER;

  subtype PRIORITY is INTEGER range 1..15;

  type NAME is (DPS8, DPS88, DPS8000, DPS90, DPS9000);

  SYSTEM_NAME  : constant NAME := DPS8;

  STORAGE_UNIT : constant     := 36;
  MEMORY_SIZE  : constant     := 256 * 1024;

  -- System-Dependent Named Numbers:

  MIN_INT      : constant     := long_integer'POS(long_integer'first);
  MAX_INT      : constant     := long_integer'POS(long_integer'last);
  MAX_DIGITS   : constant     := long_float'digits;
  MAX_MANTISSA : constant     := 60;
  FINE_DELTA   : constant     := 2.0 ** (-60);
  TICK         : constant     := 0.000016;

  type interface_language is (GMAPV, COBOL_85, PL_6, FORTRAN_77, C, GMAPV_ADA);


end SYSTEM;
```

```
-- 12-APR-1985 22:21:48.78  /ADA_MNT_JJ
-------------------------------------------------------------------------------
--
-- Date          29 july 1985
--
-- Programmer    Svedn Bodilsen
--
-- Project       DDC Ada Compiler System
--               GCOS-6 version
--               Source text for predefined package STANDARD
--
-- Module        STANDARDS.ADA
--
-- Description   This source text is read by the package STANDARD builder.
--
-- Changes       Initial version 29 March 1985
--
-- Copyright 1985 by Dansk Datamatik Center (DDC).
-- This program as well as any listing thereof may not
-- be reproduced in any form without prior permission
-- in writing from DDC.
--

------------------------------------------------------------------------------

FALSE                            -- SHORT_INTEGER DEFINED

TRUE                             -- INTEGER DEFINED
-34_359_738_368                  -- INTEGER LOWER BOUND
34_359_738_367                   -- INTEGER UPPER BOUND
36                               -- INTEGER BINARY DIGITS
                                 -- ( NUMBER OF BITS FOR EACH OBJECT )

TRUE                             -- LONG_INTEGER DEFINED
-2_361_183_241_434_822_606_848  -- LONG_INTEGER LOWER BOUND
2_361_183_241_434_822_606_847   -- LONG_INTEGER UPPER BOUND
72                               -- LONG_INTEGER BINARY DIGITS
                                 -- ( NUMBER OF BITS FOR EACH OBJECT )

FALSE                            -- SHORT_FLOAT DEFINED

TRUE                             -- FLOAT DEFINED
6                                -- FLOAT DIGITS
-16#0.1#E128                     -- FLOAT LOWER BOUND
16#0.FFFFFFE#E127                -- FLOAT UPPER BOUND
36                               -- FLOAT BINARY DIGITS

                                 -- ( NUMBER OF BITS FOR EACH OBJECT )
16#0.FFFFF8#E127                 -- FLOAT SAFE_LARGE
16#0.8#E-127                     -- FLOAT SAFE_SMALL
508                              -- FLOAT SAFE_EMAX
16                               -- FLOAT MACHINE_RADIX
6                                -- FLOAT MACHINE_MANTISSA
127                              -- FLOAT MACHINE_EMAX
-128                             -- FLOAT MACHINE_EMIN
```

```
TRUE                           -- FLOAT MACHINE_ROUNDS
TRUE                           -- FLOAT MACHINE_OVERFLOWS

TRUE                           -- LONG_FLOAT DEFINED
17                             -- LONG_FLOAT DIGITS
-16#0.1#E128                   -- LONG_FLOAT LOWER BOUND
16#0.FFFF_FFFF_FFFF_FFFE#E127    -- LONG_FLOAT UPPER BOUND
72                             -- LONG_FLOAT BINARY DIGITS
                               -- ( NUMBER OF BITS FOR EACH OBJECT )
16#0.FFFF_FFFF_FFFF_FFC#E127    -- LONG_FLOAT SAFE_LARGE
16#0.8#E-127                   -- LONG_FLOAT SAFE_SMALL
508                            -- LONG_FLOAT SAFE_EMAX
16                             -- LONG_FLOAT MACHINE_RADIX
15                             -- LONG_FLOAT MACHINE_MANTISSA
127                            -- LONG_FLOAT MACHINE_EMAX

-128                           -- LONG_FLOAT MACHINE_EMIN
TRUE                           -- LONG_FLOAT MACHINE_ROUNDS
TRUE                           -- LONG_FLOAT MACHINE_OVERFLOWS

FALSE                          -- SHORT_FIXED DEFINED

TRUE                           -- FIXED DEFINED
-34_359_738_368                -- FIXED LOWER BOUND
34_359_738_367                 -- FIXED UPPER BOUND
36                             -- FIXED BINARY DIGITS
                               -- ( NUMBER OF BITS FOR EACH OBJECT )
FALSE                          -- FIXED MACHINE_ROUNDS
TRUE                           -- FIXED MACHINE_OVERFLOWS

TRUE                           -- LONG_FIXED DEFINED
-1_152_921_504_606_846_976     -- LONG_FIXED LOWER BOUND
1_152_921_504_606_846_975      -- LONG_FIXED UPPER BOUND
72                             -- LONG_FIXED BINARY DIGITS
                               -- ( NUMBER OF BITS FOR EACH OBJECT )
FALSE                          -- LONG_FIXED MACHINE_ROUNDS
TRUE                           -- LONG_FIXED MACHINE_OVERFLOWS

TRUE                           -- DURATION DEFINED

-1_099_511_627_776.0           -- DURATION LOWER BOUND
1_099_511_627_775.99999904632568359375       -- DURATION UPPER BOUND
0.000016                       -- DURATION DELTA
2#1.0#E-20                     -- DURATION SMALL
-20                            -- DURATION SMALL_POWER
                               -- ( 2**SMALL_POWER = SMALL )
72                             -- DURATION BINARY DIGITS
                               -- ( NUMBER OF BITS FOR EACH OBJECT )
FALSE                          -- DURATION MACHINE_ROUNDS
TRUE                           -- DURATION MACHINE_OVERFLOWS

36                             -- BINARY DIGITS FOR ALL ENUMERATION TYPES
                               -- ( NUMBER OF BITS FOR EACH OBJECT )
```

APPENDIX C

TEST PARAMETERS

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

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	<1..248 => 'A', 249 => '1'>
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	<1..248 => 'A', 249 => '2'>
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	<1..124 => 'A', 125 => '3', 126..249 => 'A'>
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	<1..124 => 'A', 125 => '4', 126..249 => 'A'>
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	<1..246 => '0', 247..249 => '298'>
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	<1..244 => '0', 245..249 => '69.00'>

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	"(124)A"
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	<1 => '"', 2..125 => 'A', 126 => '"'>
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	<1..129 => ' '>
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	34359738367
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	75
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	F(*FILE
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	N234567890123
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	0.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	1100000000000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	F@LENAME

$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	NAMETOLONGFORFILENAME
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-34359738368
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	34359738367
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	34359738368
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	0.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-1100000000000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	17
$MAX_IN_LEN Maximum input line length permitted by the implementation.	249
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	23611832414348226068 47
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2361183241434822606848
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	<1..244 => '0', 245..249 => '2#11#'>

$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	<1..242 => '0', 243..249 => '16#F.E#'>
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	<1..247 => 'X'>
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_ INT.	-2361183241434822606848
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	no_such_type
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFFFFFFFFFFFF#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 28 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A: A basic declaration (line 36) wrongly follows a later declaration.

E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.

C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.

C35502P: Equality operators in lines 62 & 69 should be inequality operators.

A35902C: Line 17's assignment of the nominal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

C35A03E, & R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.

C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

C37215C, E, G, H:	Various discriminant constraints are wrongly expected to be incompatible with type CONS.
C38102C:	The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.
C41402A:	'STORAGE_SIZE is wrongly applied to an object of an access type.
C45332A:	The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.
C45614C:	REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).
E66001D:	This test wrongly allows either the acceptance or rejection of a parameterless function with the same identifier as an enumeration literal; the function must be rejected (see Commentary AI-00330).
A74106C, C85018B, C87B04B, CC1311B:	A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively (and possibly elsewhere).
BC3105A:	Lines 159..168 are wrongly expected to be illegal; they are legal.
AD1A01A:	The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.
CE2401H:	The record aggregates in lines 105 & 117 contain the wrong values.
CE3208A:	This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.