LEVEL II

1

AD A095699

INTERMETRICS

81 2 27 005

LEVEL

(1)

(14) IR-663

(9) FINAL REPORT.

ON

(6) ADA TEST AND EVALUATION.

(11) 6 FEBRUARY 1981

(12) 167

(15) MDA903-79-C-0477, ARPA Order-3341

APPROVED FOR PUBLIC RELEASE
DISTRIBUTION UNLIMITED

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

38x 6

303

INTERMETRICS INCORPORATED • 733 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840
TELEX NO. 710 320 7523

| | |
|---|---|
| Submitted to: | DARPA/IPTO<br>1400 Wilson Blvd.<br>Arlington, VA 22209 |
| | Contract: MDA 903-79-C-0497 |
| Submitted from: | Intermetrics, Inc.<br>733 Concord Ave.<br>Cambridge, MA 02138 |

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| | | |
|---|---|---|
| **REPORT DOCUMENTATION PAGE** | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
| 1. REPORT NUMBER IR#663 | 2. GOVT ACCESSION NO. AD-A095 699 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) FINAL REPORT ON ADA TEST AND EVALUATION | | 5. TYPE OF REPORT & PERIOD COVERED Final Report |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) NA | | 8. CONTRACT OR GRANT NUMBER(s) MDA90379C0497 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Intermetrics, Inc. 733 Concord Avenue Cambridge, Mass. 02138 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS AO 3341 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209 | | 12. REPORT DATE 6 February 1981 |
| | | 13. NUMBER OF PAGES 66 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report) Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

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)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

ADA
Programming Language
Computers
FORTRAN

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

(U) In June of 1979, following an extensive process of selection and revision, the Preliminary Ada language definition was published. As a means of further refining the language, it was decided to approach the prospective user community and solicit their comments and reactions. This report describes the methods used to gather and evaluate the many responses received, and discusses the more prominent issues raised.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## Table Of Contents

# Report on Ada Test and Evaluation

## 1. Introduction

In June of 1979, following an extensive process of selection and revision, the Preliminary Ada language definition was published. As a means of further refining the langurge, it was decided to approach the prospective user community and solicit their comments and reactions. This report describes the methods used to gather and evaluate the many responses received, and discusses the more prominent issues raised.

## 2. Background

There were two major avenues used to solicit reactions. The publication of the Preliminary Ada Reference Manual and Rationale, and Ada newsletters in ACM Sigplan Notices, the major informal journal on programming languages, assured wide circulation of the language definition and requests for comments.

The other major and more formal source of comments was the Test and Evaluation reports. Military organizations, defense contractors, and the computer industry, were asked to analyze existing applications programs, possibly reprogram them in Ada, and report their experience. A few outside the military and its contractors also submitted such reports.

The High Order Language Working Group (HOLWG) appointed a .ol of experts on programming languages, termed the Ada Distinguished Reviewers, to oversee the review of these comments and further discuss language issues in order to assist the language design team at CII-Honeywell-Bull in its refinement effort. Intermetrics, Inc. was contracted to coordinate comment processing and to support the Distinguished Reviewers administratively. This program was Intermetrics Test and Evaluation.

The functions of the Intermetrics Test and Evaluation program have changed with the needs of the Distinguished Reviewers. Initially, it was thought to be most productive for Intermetrics to prepare position papers (Draft Change Requests), discussing controversial issues and proposing changes which would then be discussed by the Distinguished Reviewers. If accepted by the Reviewers, the proposals would be passed along to HOLWG for possible approval as Language Change Requests.

This approach proved too rigid, as it put the language design team and the Distinguished Reviewers in an adversary relationship. In subsequent Reviewers' meetings, Intermetrics did not present position papers, but instead continued to prepare documents useful for the Reviewers. These documents form the core of this report.

In addition to technical and administrative support, Intermetrics performed some general analyses on the Test and Evaluation documents. The results of the analyses were presented at the Paris meeting to the Distinguished Reviewers; their revised and updated versions are found within.

Intermetrics also provided continuing support for the Test and Evaluation process by serving as the clearinghouse for information and documents. Numerous inquiries concerning certain aspects of the language--from details of the syntax to questions about its future importance to computer science--were received by traditional and Arpanet mail, telephone, personal visits, and chance meetings. These were answered and documents were sent as appropriate.

## 3. Ada Test and Evaluation Reports

In addition to input from language designers and theoreticians, the Test and Evaluation process considered the opinions of system and application software developers.

Defense Department sites and contractors were asked to study their existing applications and select one to reprogram in Ada. The results of their experience are found in the Test and Evaluation Reports. The eighty-two TER's represent a broad spectrum of experience on the part of the participants.

A TER comprises two parts. The primary section contains a questionnaire, composed by DoD, which addresses the participant's experience with and reaction to using Ada in an applications context. Additionally, the primary section often contains an algorithm written in both a language normally used by the participant, and in Ada. Along with these comparison programs are usually more extensive comments dealing with issues that were beyond the scope of the questionnaire.

The supplementary section varies in nature and contents with each report. Many participants included system and language reference manuals in addition to the source code in order to fully present their applications.

To systematize their dissemination, the TER's were numbered and separated into their primary and supplementary sections. The primary section carries a TER number only; the supplementary sections carry the TER number and a suffixed section letter. The supplementary material was further separated into original code sections (containing code written in the original language), and Ada code sections (which contained the Ada translation) in order to facilitate distribution of Ada code samples to those wishing to analyze them. Appendix E indicates the languages used in these code comparisons.

Since the Test and Evaluation Reports were not submitted in machine-readable form, and their volume precludes entry, they are not on line. The derived files produced during the Intermetrics analysis, however, are all on line, and are described in the Appendixes.

## 4. Analytical Methods

Given the form and content of the TER's, some method was needed to summarize the useful information they contained. One possible technique would have been to summarize each TER separately and report its contents without any evaluation. This would faithfully preserve the contents of the individual TER's and would be the manner least influenced by the summarization. However, it would not help the language review process since it does not address the difficulties in understanding Ada, and the differences in training and experience among the Test and Evaluation participants.

Alternatively, the TER's could be analyzed into issues, as were the LIR's. This would be unsatisfactory as the TER's take quite a different approach to problems than do the LIR's: rather than study language issues, the TER's deal with language applications.

In order to avoid the problems described above, verbatim extracts, a topical cross-reference, and a presentation of the conclusions drawn by the TER analysis were prepared.

## 5. Findings

### 5.1 Extracts

So that the language team and outside evaluators might gain insight into the participants individual reactions to Ada, verbatim extracts from the TER's were compiled to preserve the original phrasing and tone, which would otherwise not be accessible to those unable to sift through the thousands of pages which comprise the original reports.

The extracts were chosen and abridged to present a balanced view of each report. Comparisons in the extracts refer to the original implementation language, so that the phrase "Ada is more debuggable" means "Ada is more debuggable than our current language". The extracts are not intended to reflect difficulties in understanding the preliminary manual or the language: these issues, are of central concern to writers of manuals and expository treatments, difficulties in understanding the language will also manifest themselves in the detailed technical summaries. Extracts were not taken from sections of the TER's where the participants indicated that they did not fully understand the language or the manual, as it is not clear how to interpret such answers in the context of language changes. Comments relating to concerns of compiler quality (i.e. degree of optimization, speed, size, etc.), development environment, organizational problems, and the like were similarly not included in this sample. (The extracts used in the analysis are found in Appendix F.)

## 5.2 Language Comparison

A review of the TER's shows that many conclusions drawn by participants are heavily influenced by their experience. In light of this fact, some general responses to Ada are categorized by the respondent's previous language.

### 5.2.1 Assembly language

Not surprisingly, assembly language programmers emphasize control over object program and data. Many praise such facilities as tasking and abstraction mechanisms in one breath, only to criticize them as inefficiently implemented in the next.

Specific features desired by this group are static allocation of data, unsafe pointers, and representation specifications.

The most important factor in the acceptance of Ada by this group will be the availability of compilers which use time and space efficiently. The assembly language programmers do not believe it can be done.

### 5.2.2 Fortran

Fortran programmers comment favorably on the presence of structured programming constructs such as IF...THEN...ELSE and loops. There is a definite split in opinion about GOTO's: some maintain they are important; others would like to eliminate them from the language entirely in order to encourage better programming style. It is not clear whether this argument is based on experience or current trends.

One Fortran feature missed is formatted I/O. If a formatted I/O facility is not standard, will there exist such a capability for every machine? Will the facilities for various machines be compatible? Can a formatted I/O facility be written efficiently using an Ada package?

### 5.2.3 PL/I

PL/I programmers prefer PL/I's model of storage allocation types in which there is an explicit choice of allocation method at the allocation of each variable. The parallel "based variable" facility is also missed.

Source inclusion is sometimes preferred to the package and separate compilation facility. Of course, source inclusion does exist in Ada through the Include pragma.

### 5.2.4 Algol-like languages

Among the Test and Evaluation participants, there is a fairly sizable contingent, primarily from England, using Algol-like languages in embedded applications. This group misses high-level Algol constructs more than lower level constructs. Additionally, conditional expressions and functional arguments are repeatedly mentioned as useful and efficient. It is not clear from the TER comments whether the revised Ada generics would satisfy the request for functional arguments.

## 5.3 Major Issues

This section identifies the major issues which have been raised by Ada Test and Evaluation participants. For ease of exposition, the issues have been grouped into seven categories:

1. Tasking
2. Program Structure, Name Resolution and Separate Compilation
3. Predictability and Efficiency of Object Code
4. Values and Expressions
5. Abstraction and Extensibility
6. Language Phase-In
7. Syntax

### 5.3.1 Tasking Issues

The TER reports present a variety of real-time applications requiring tasking. While many participants implemented their real-time applications successfully in Ada, others were unable to do so. The crucial question is whether this inability was due to shortcomings in the language, or to inadequate education in the use of Ada for such applications.

The difficulties which Test and Evaluation participants had in applying the tasking model to applications prompted both the Distinguished Reviewers and the Language Design Team to examine the existing model. The Revised Ada tasking model will be substantially the same, but with some important extensions and changes resulting from the Test and Evaluation process.

The Ada tasking model differs radically from many previous applications languages in that it recognizes tasks explicitly and has a well-defined notion of task communication and synchronization through the concept of the rendezvous. Careful education in the use of Ada tasking and re-analysis of applications will be needed.

It should be noted that although features of Ada tasking were rather extensively criticized, and that many of these criticisms led to language changes, the Ada tasking mechanism as a whole was often mentioned as a particularly strong feature of the language. Fully fourteen of the TER's praised tasking explicitly.

The following is a detailed account of the major concerns expressed:

5.3.1.1 --Ada tends to encourage and sometimes require programmers to define more tasks than they would in other languages, and there is a concern that this will result in excessive overhead, specifically in the areas of scheduling and context switching. T'CLOCK is a minor issue in this discussion some say it must be supported since it cannot be implemented within the language, and others that it incurs unavoidable and unacceptable overhead. The ability to pass arguments to tasks at creation is another specific request related to tasking efficiency. It has been asserted that the Habermann-Nassi optimization answers the efficiency concern. However, there are questions about how to tell when it can, should, or will be used, so some reviewers remain skeptical.

5.3.1.2 --Preliminary Ada does not at this point provide sufficient control of scheduling decisions, specifically the assignment of tasks to available physical processors (or virtual processors in a time-shared environment). This area will have a major impact on Ada's use in real-time systems and for systems programming. Some want the scheduling points to be precisely defined, to be able to explicitly suspend and resume tasks, or to be assured that scheduling is "fair", or both.

5.3.1.3 --The scheduling rules do not guarantee that hardware interrupts will cause the timely execution of the corresponding interrupt handler.

5.3.1.4 --The mechanisms for sharing data between tasks seem overly-involved to people who are used to having a mechanism for representing synchronization information as data.

5.3.1.5 --Many people want to be able to determine task names during program execution. For example, a server task might be doing work for several other tasks and need to determine which one has just been the partner in a rendezvous. People have also wondered how to terminate an errant task if they can't name it. There is substantial support for the idea of having TASK as a built-in type in the language as a solution to these problems.

5.3.1.6 --Some people have asked for a capability to dynamically create and delete tasks at run-time.

5.3.1.7 --Some people are concerned about the asymmetry in the CALL-ACCEPT model for invoking task entries. The writer of a task can prevent it from waiting indefinitely to be called by using the SELECT and DELAY statement; but the writer of the task making the entry call has no equivalent capability. Conditional and timed entry calls are desired.

5.3.1.8 --There may be problems associated with the handling of possible error conditions that can arise when one of the partners to a rendezvous has died.

5.3.1.9 --The requirement for hierarchies of tasks as provided by current Ada has not been demonstrated. Both users and implementors have expressed a desire for restrictions in this area to reduce complexity. Task hierarchies also may create problems in the interaction between task termination and scope exit.

5.3.1.10 --The requirement for one task to be able to change another task's priority has not been justified. Some people have pointed out that the ability to change priorities can be misused as a synchronization mechanism and should be eliminated. The ability of a task to terminate its parent has also been objected to.

5.3.1.11 --Although the tasking model itself is clean and simple, it is not always obvious how to apply it to problems which have been previously solved in other ways. This proved to be a difficulty expressed in several TER's. The concept of buffer task, for instance, although explained in the language documentation, is not easy to grasp--indeed, it appears to be inefficient at first glance. However, Habermann and Nassi have shown that the use of buffer tasks does not imply that they must be scheduled separately from the tasks they service.

5.3.1.12 --Many TER's requested that task priorities be strictly enforced, and that the scheduling algorithm be well-defined. The lack of definition of scheduling strategy left many participants unable to define solutions to their applications problems. One difficulty with strict priorities is the possibility that they might be used as synchronization mechanisms, defeating many of the advantages of the tasking model. Although forbidding this is unenforceable, on the balance it was decided that strict priorities were in fact needed in the language. As for the scheduling algorithm, the decision to adopt strict priorities partially defines it; the definition of scheduling points in revised Ada further defines it.

5.3.1.13 --The semantics of Preliminary Ada interrupts were often mentioned as inadequate, as they implied queuing of interrupts rather than providing immediate service. Interrupts in Revised Ada will correspond closely to the traditional notion.

5.3.1.14 --Where previous tasking models deal in low-level operations and explicit suspension and resumption of tasks, applications programs written in those terms cannot be easily translated into Ada. Some TER's request these facilities in Ada, but it is not clear whether they are functionally necessary.

## 5.3.2 Program Structure, Name Resolution, Separate Compilation, and Related Issues

Another area of concern is a perception that the mechanisms Preliminary Ada provides for structuring programs are unnecessarily rich, and hence, complex from the perspective of both users and implementors.

5.3.2.1 --Some rules in the manual (e.g., no aliasing) would require a complete pass through the entire system (a transitive closure) to check. Such a check could make separate compilation impractical.

5.3.2.2 --Some people are concerned about the complexity of the overloading resolution rules. For example, the interaction of renames with the ability to change discriminants has been mentioned. The design team has already decided that parameter modes are not a sufficiently strong criterion for overloading resolution, and that parameter names are not an appropriate criterion for overloading operators. It has been argued (persuasively) that overloading resolution with optional named parameters is a computation exponential in the number of parameters.

5.3.2.3 --Users have difficulty understanding the multiple mechanisms for structuring a program.

5.3.2.4 --Some users have difficulty understanding the combination of USE, RENAMES, and RESTRICTED. Examples (both good and bad) used in these debates are often library packages which contain large name spaces.

5.3.2.5 --Problems can arise when packages call other packages during initialization. It is not clear how the compiler must determine a workable initialization order (it could be expensive) or how the user can specify the order.

5.3.2.6 --Some people have objected to the private part of the module specification on both methodological and practical grounds.

5.3.2.7 --The language currently does not require compile time evaluation of static expressions. This means that some compilers will detect errors at compile time and others at run-time.

5.3.2.8 --Several people have asked that a conditional compilation capability be added to the language.

5.3.2.9 --Many important optimizations only work in the absence of aliasing, but aliasing can only be detected with a transitive closure computation. The language definition should take a position on the validity of optimizations which depend on the absence of aliasing. Also, some kinds of aliasing are useful (and safe). A mechanism is under consideration to allow the programmer to specify as part of a procedure or entry declaration that the procedure has been written to produce a correct result even if actuals are aliased through binding to formals. Aliasing of globals passed as parameters would still be an error in all cases.

### 5.3.3 Predictability and Efficiency of Object Code

One goal of Ada optimizing compilers is to generate code that is competitive with machine code handwritten by a skilled system programmer. This section discusses several efficiency-related issues for sequential Ada.

5.3.3.1 --Preliminary Ada specifies efficient parameter passing with some sacrifice of safety and portability in three special cases: variables shared between tasks, exceptions, and aliasing. Explicit copies must be made to prevent variables from being shared between tasks, the state of OUT and INOUT actuals is indeterminate if the exit is caused by an exception, and aliasing is illegal.

A typical implementation allows the calling program to pass/return small objects in registers or on the stack and to pass reference pointers to larger objects. Great care must be taken in reference implementations for INOUT and OUT parameters when the actuals are more tightly constrained than the formals; an incorrect implementation could result in assigning an illegal value to the actual which overwrites adjacent memory. An incorrect implementation could also leave an incorrect value in the actual after an exception.

Some of the criticism of preliminary Ada's parameter passing mechanisms comes from a mistaken belief that it is less efficient than reference passing.

Other criticism comes from people who object to the fact that a programmer can determine whether the compiler which implements a particular call be reference or by copy, and exploit that fact to write nonportable programs. Most people who insist on precisely defined semantics want pure copy semantics; they have not been able to convince the people who are primarily concerned with efficiency that large objects can be safely passed by reference while guaranteeing copy semantics and without a transitive closure analysis.

5.3.3.2 --Users are confused by the distinction between functions and value returning procedures. The current definition of function seems not to allow desired optimizations, seems to outlaw "benevolent" side-effects (e.g., garbage collection, instrumentation), and requires a transitive closure computation to check.

A popular proposal is to allow value returning procedures full functional notation and be usable wherever a value of the type is required. Under this scheme, it would be acceptable to eliminate pure functions. A declaration might be provided as part of the value returning procedure declaration to specify that calls may optimized under the assumption of abstract functionality.

5.3.3.3 --Some people are concerned that Ada will force programmers to use dynamic storage allocation and require the run-time system to do garbage collection. Garbage collection is unacceptable in many real-time systems. Examples of capabilities which have been requested to overcome these inefficiencies are explicit Allocate and Free mechanisms and pointers to static data.

5.3.3.4 --Some participants in the T&E analysis have asked that the language provide an explicit overlay capability. There does not seem to be any way to write such a facility in Ada without a primitive operation which says "execute this data as code".

5.3.3.5 --A much more robust set of standard implementation parameters is needed. For example, memory size, storage, remaining stack storage, target machine, word size.

5.3.3.6 --Many people have criticized the UNSAFE_PROGRAMMING feature of the language. In part, the problem is the name, which implies that the use of the facility is inappropriate, whereas it is in fact the only way to implement some very important operations such as the mapping of input data into a typed variable. One reviewer has stated that the language should have exactly one such feature, and UNSAFE_PROGRAMMING is the correct one. Others have suggested that there are degrees of unsafeness, and that really dangerous operations such as turning an integer into a pointer should be distinguished from safer kinds of type conversion.

### 5.3.4 Values and Expressions

A variety of issues have been raised with respect to variables, values, expressions and the initial values of variables. Some of these are minor, and have already been addressed in language changes under consideration. For example, NO_VALUE_ERROR will be eliminated, a private type will be defined for time, Overlap_error will be eliminated, the Underflow exception will be eliminated, exceptions occurring during the elaboration of declarations will be passed to the containing scope, the MOD function will have the conventional definition, and qualification will be required for one - component aggregates.

5.3.4.1 --Several LIR's request the capability to initialize parts of aggregates. With the deletion of NO_VALUE_ERROR, this capability seems reasonable and desirable.

5.3.4.2 --Some people want pointers initialized to NULL, others want them initialized to an illegal value (i.e. not NULL). Others point out that such automatic initialization would introduce a non-uniformity into the language. The language change under consideration suggests initialization to NULL.

5.3.4.3 --People have had trouble understanding type derivation and type conversion.

5.3.4.4 --Many LIR's suggest changes to the built-in numeric types, especially fixed point.

5.3.4.5 --The language does not contain a built-in "SET" type as required by Steelman. The notation for bit string constants is presently somewhat awkward.

5.3.4.6 --Many people have requested more convenient capabilities for handling variable length strings.

5.3.4.7 --Some people have asked to be able to specify default initial values with type definitions.

5.3.4.8 --Implementation dependencies can arise in certain cases because the language does not define a semantic order of evaluation for expressions, and does not specify the mathematical properties of operators which can be assumed.

## 5.3.5 Abstraction and Extensibility

Ada has advanced capabilities for defining new data types and operations and for defining generic procedures. These abstraction and extensibility features set Ada apart from existing programming languages for embedded systems. They are also the focus of some of the most active debates in language issue and TER's. This section summarizes some of the main topics of debate.

5.3.5.1 --In preliminary Ada, the equality operation for record and array types is defined in terms of the predefined equality of the component types. When the user has the possibility of redefining equality, this may lead to strange anomalies. The problem of assignment for composite types is akin to that of equality, though less acute because assignments are not user definable.

5.3.5.2 --It has been suggested that parameters should be named when overloading a function, and that the same overloading rules should be used as for generics.

5.3.5.3 --There is no way to set discriminants (e.g. array bounds or variant record selectors) of private types at run-time, because the component names are not visible. A "limited window into private types" mechanism is under consideration which would allow the programmer to specify that some of the discriminants embedded in a private type are externally settable at initialization time.

## 5.3.6 Language Phase-In

A variety of issues have been raised regarding interfaces to other languages, interfaces to existing operating systems, representation of external interfaces, and input/output packages. While these will have a major impact on the acceptance of the language and, in particular, on how quickly it will come into widespread use, they are the responsibility of the environment rather than the language. The one language change currently identified as addressing the above issues is to have the visible part of a module specify the relevant language processor if the body consists of foreign code.

## 5.3.7 Syntax

A large number of comments on the language syntax have been received, most frequently making mention of such issues as name or keyword, the position of semi-colon, parameter association, and the like. It is recommended that these be studied carefully after the language semantics have been finalized. The goal of standard Ada syntax should be readability for documentation purposes and for use in publishing algorithms. It is assumed that software tools such as language oriented text editors will be used to simplify the writing and entry of Ada programs.

## 6. Difficulties in Interpretating the TERs

Some of the material in the TER's was discounted because of admitted or obvious misunderstandings of the language. It was not always possible to consult individually with participants when they had problems, misunderstandings or had missed points. The root of the problem may lie in the fact that the preliminary language manual was not a tutorial document, and that those tutorial documents which were available did not examine all aspects of the language. Visibility, for example, was one area widely misunderstood in the LRM. However, TER's which reflected some confusion or misunderstanding ultimately played an important role in evaluating Ada, as they helped pinpoint areas of ambiguity.

Many of these problems in understanding the language arose when the participant was required to use mutually interactive features. However, this is precisely the sort of problem covered rather extensively in the LIR's; thus the TER's and LIR's complement one another.

Other material which had to be discounted in the TER's was the body of comments pertaining to the efficiency of various language features. Although it is certainly true that certain constructs can be shown to have intrinsic inefficiencies, and that efficiency is essential in many embedded applications, the TER's often do not identify what function must be performed efficiently, but rather indicate that a particular implementation of that function might be expected to compile poorly using the compilers familiar to the TER writer.

The underlying application requirements in a TER often become difficult to interpret when the writer presents his or her own language solutions rather than working within the framework of the desired functionality. For instance, many TER's request static allocation of variables as a language feature, apparently for time efficiency and ease during debugging. It is possible that with modern compiler technology, non-static allocation could be more efficient without compromising debugging. This is a language and compiler design matter. In this case, the functionality desired is fairly clear, and the matter has been discussed at meetings of the Distinguished Reviewers.

## 7. Mandates for Change

In some areas, there was great unanimity of opinion about necessary language changes. These are presented below with indications of design team actions.

7.1 --Some way of guaranteeing exact fixed-point representation is desired. The approximate fixed-point system of Preliminary Ada does not suffice for many applications. Exact fixed-point arithmetic also is desired. These concerns might be answered either through a change to fixed point or a demonstration that the applications requirements can be met through the writing of packages. The Language Design Team is revising fixed-point.

7.2 --The syntax of the case statement, "Case ... of when ..." is widely disliked as not resembling English. The semantics appear to be satisfactory, but the syntax will be changed.

7.3 --Variable-length strings are needed, again either through the language or through definable libraries. The Distinguished Reviewers and the Language Design Team have studied this matter carefully, and will meet the need.

7.4 --A more complete I/O package is desired which would include multiple data types per file, Fortran-like formatting functions, and more functions in the standard package. This requirement can currently be met with the package mechanism. To what extent a larger standard I/O package should be part of the language and not the environment is still an open issue.

7.5 --True interrupts are needed. Revised Ada will have them.

7.6 --Many TER's request "bitstrings". This is a very widespread demand, but it is not clear what functionality is desired in using bitstrings The Preliminary Ada Unsafe_Conversion function (now renamed to Unchecked_Conversion) can certainly convert between integers and packed arrays of booleans. Packed arrays of booleans themselves can represent sets. Thus the bitstring representation of sets is easily captured by Ada. LIR's mention the lack of set notation for this kind of set and the awkwardness of the aggregate notation.

## 8. Conclusion: The Overall Response to Ada

The T&E Reports show an extremely favorable attitude and a great deal of acceptance for Ada among the prospective users. Repeatedly, Test and Evaluation participants mention the advantages of coding in Ada, maintaining systems written in Ada, transporting Ada programs to other target machines, and so on.

Twenty-three TER's explicitly favored strong typing; the strongest comment on any one feature. Other features with strong appeal were enumeration types, overloading, packages, the separation of specifications from bodies, restricted visibility, tasking, separate compilation, exception handling, and generics.

There are consistently strong complaints about functionality only in one area: tasking and interrupts. There is a great deal of concern that the tasking and interrupt constructs cannot handle the requirements of embedded applications. There are two sides to this concern: one, semantic functionality, the other, performance requirements.

Many reviewers indicated that they liked some other language better. Yet, there was virtually no agreement on which language was preferred. It is clear from the results that Ada is the most viable candidate for standardization of any present language. Almost everyone praises some features of Ada. There are groups of people who say that, for example, PASCAL is just a toy, FORTRAN is hopelessly backward, LISP is no good for "real" projects, etc. The reactions to Ada are more along the lines of "If only they would change one little thing...". It is expected that final Ada will meet the very ambitious objectives of the DoD common high order language project.

Another major theme apparent throughout the T&E analysis was the need for better manuals and tutorial materials.

There is a sense of optimism that the issues which have been identified by the T&E analysis can be resolved, and that the result of the design refinement process will be a polished and effective tool which fully meets the objective of the Common High Order Language Program.

## APPENDIX A: TER Topic Index

The TER Topic Index cross-references specific technical concerns mentioned in the TER's with LRM chapters.

The index basically serves two major functions: it reflects a general sense of the technical opinions of the Test and Evaluation participants, and it may bring up or emphasize topics which might otherwise not be considered.

The TER questionnaire contains several sections which ask Test and Evaluation participants to list which language features they liked, which they thought ought to be changed, and which they thought were redundant. Since many of the proposed changes were in fact proposed additions, the responses to these questionnaire sections are divided into categories labelled Add, Change, Like, and Redundant.

Although this Topic Index certainly does not represent all the concerns of Test and Evaluation participants, it represents those issues which they considered most important. The questionnaire answers were put into uniform nomenclature, and similar answers were merged.

The index entries were categorized by LRM chapter number rather than LRM section number, as most replies were not specific enough to be related to a particular section. After each topic entry are listed the TER's mentioning it. Some groups have submitted more than one TER; some TER's are more extensive in their coverage than others; some TER's are more carefully considered than others; some topics are closely related to others. For these reasons, it was considered unwise to take a count of the number of TER's mentioning a topic. It would be even less wise to base decisions about the language on such a count, since the varying importance and expertise of submitters of TER's are nowhere accounted for.

1: A Language subsets: 25,
1: C Make declaration syntax more uniform: 30,
1: C Improve syntax: 4,
1: C Require blocks rather than sequence of statements: 30,
2: A Abbreviations for keywords: 3, 30,
2: A Imbedded comments: 30, 72,
2: A Alternate character set support: 13,
2: A Bit string constants: 13, 41, 44, 51, 59,
2: C Make "_" non-significant: 30, 40,
2: L "_" in identifiers: 19,
2: L Long identifiers: 19, 37, 75,
2: R Bases other than 2, 8, 10, and 16: 18,
2: R Significance of "_" in tokens: 7,
3: A Bit handling: 26, 71, 77,
3: A Function as data: 7,
3: A Multi-level structures: 3,
3: A Implicit conversion of numeric types (when no loss of precision): 30,
3: A Reference variables: 7, 19, 30,
3: A Simula classes: 7,
3: A Static allocation of access objects: 13,
3: A Unsafe pointers: 14,
3: A Strings: 29, 35, 30, 45, 59, 61, 63, 72,
3: A Variable declarations after local program bodies: 84,
3: A Statis variables: 84,
3: C "=>" has two meanings: 19, 30,
3: C Ranges should not have to be contiguous: 30,
3: C Delta is poor keyword: 19,
3: C Expressions in range constraints(?): 8,

```
3: C  Fixed-point delta should be exact: 27, 28,
3: C  Require specification of maximum size of strings: 2,
3: C  Store matrices by column: 18,
3: C  Types too restrictive: 15,
3: C  Allow anonymous types in record fields: 28,
3: C  Use structure equivalence for arrays: 30,
3: C  Guaranteed one-step conversion between derived types: 30,
3: L  Aggregate syntax: 7,
3: L  Aggregates: 29, 40,
3: L  Arrays: 13,
3: L  Enumeration types: 7, 34, 35, 37, 38, 58, 68, 75, 88,
3: L  Derived types: 88,
3: L  Machine-independent data definition: 2,
3: L  Overloading: 2, 7, 35, 37, 42, 61,
3: L  Precision specification: 13,
3: L  Record syntax: 19,
3: L  Record variant semantics: 29,
3: L  Initialization in declarations: 86,
3: L  Strong typing: 2, 3, 10, 16, 18, 26, 29, 31, 46, 48, 50, 52, 54, 58,
3: L  Variant arrays in records: 86,
3: L  Arrays with unspecified index range: 86,
3: L  Type constraints: 1, 20, 49,
3: L  User-defined types: 5, 17, 26,
3: L  Scope for access types: 29,
3: R  Subtypes: 87,
3: R  Either subtypes or derived types: 19,
3: R  Derived types: 29,
3: R  Named components in aggregates: 25,
4: A  Conditional expressions: 7, 28, 30,
4: A  Multiple assignments: 30,
4: A  Method of expressing parallelism in expression evaluation: 21,
4: A  'Free' operation: 29, 69,
4: A  Standard built-in math library: 19,
4: A  Standard built-in array operations: 16, 19,
4: C  Accurate fixed point arithmetic (specification, coercion): 8, 85, 88,
4: C  Define mathematical properties of user-defined operators: 1,
4: C  More control over allocation: 13, 15,
4: C  Qualified expression syntax: 13,
4: C  Time should not be floating point: 19, 86,
4: C  User type names should be overloadable as conversion functions: 83,
4: L  Attributes: 20, 21, 29,
4: L  Expression structure: 19,
4: L  Array slicing: 29, 88,
4: L  No automatic type conversion: 14,
4: R  Allocators for access types: 1,
4: R  Array slicing: 18,
5: A  Combined For and While statements: 16,
5: A  Compound statements: 7,
5: A  Loop failure exits: 17,
5: A  More loop constructs: 13, 16, 27, 87
5: A  Block exits: 83,
5: A  Exit from named block: 30,
5: C  Remove mandatory semicolon before end, elsif, etc.: 30,
5: C  Allow mixing of "and then" and "or else": 28,
5: C  Use "do" not "loop" as keyword: 19,
5: C  Allow VRP's as conditions: 30,
5: C  Overloading rules too complicated w.r.t. parameters: 86,
5: L  Recursion: 20, 21, 22,
5: L  Structured programming constructs: 2, 5, 10, 13,
5: R  Elsif: 18, 19,
5: R  Exit when: 8, 54, 85,
5: R  Exit: 4,
5: R  Function call syntax: 20, 86,
5: R  Keyword parameter-association syntax (:=:, etc.): 7, 19, 27,
5: R  Assert: 30, 54, 64,
5: R  Labels and gotos: 1, 4, 30, 84,
5: R  Short circuit conditions: 18, 54, 88,
5: R  VRP's: 29,
6: A  Functional arguments: 8, 20, 21, 28, 40, 74, 85,
6: A  Intermixed declarations: 7,
6: A  Generalize initialization in type declarations: 30,
6: A  Not recursive/reentrant declaration: 2,
6: A  Variable number of parameters: 29,
6: A  Guaranteed by-value calls: 84,
6: C  Define parameter passing: 18,
6: C  Reference passing preferred: 14,
6: C  Functionality should not be compiler-verified: 86,
6: L  Initialization in declarations: 29,
6: L  Default parameters: 7, 29, 35,
6: L  Functions and VRP's: 21, 22,
6: L  Parameter modes: 86,
6: R  Declarations in blocks: 7,
6: R  Default parameters: 25,
6: R  Initial values in declarations: 25,
6: R  Optionality of block declarations: 1,
6: R  Recursion support: 13,
6: R  Tasks and Procedures should be merged: 5,
6: R  VRP's: 26,
7: C  Allow representations in private part: 28,
7: L  Information hiding/data abstraction in general: 10, 14, 20, 34,
7: L  Packages: 4, 8, 10, 16, 29, 40, 46, 50, 52, 56, 58, 61, 68, 73,
7: L  Private types, parts: 2, 8,
7: L  Separate specifications: 1, 2, 13, 19, 38, 47, 50, 58, 69,
7: R  Nested packages: 15,
7: R  Scoping hierarchy: 13,
7: R  Separate specifications: 18,
8: C  Clarification of separate compilation and visibility: 1,
8: C  Loop index should be valid beyond end of loop: 27,
8: C  Restricted is poor keyword: 19,
8: C  Visibility rules disliked: 13, 46, 49,
8: L  Logical scope rules: 16, 18,
8: L  Restricted visibility: 4, 22, 23, 55, 87,
8: L  Private types: 87,
8: R  Use clause: 25,
9: A  Background tasks: 13,
9: A  Initiate parameters: 11,
9: A  Mutual exclusion to data access: 22, 23,
9: A  Timed-out entry calls: 30, 86, 87,
```

| | | |
|---|---|---|
| 9: | A | Suspend and resume of tasks: 82, |
| 9: | A | Easier cyclic scheduling: 88, |
| 9: | C | Disallow data shared between tasks: 20, 21, |
| 9: | C | Forbid aborting or changing priority of parent tasks: 8, 85, |
| 9: | C | Interrupt semantics: 13, 26, 82, |
| 9: | C | More control over scheduling: 13, 26, 82, |
| 9: | C | Preemptive priorities: 27, |
| 9: | C | Rendezvous too restrictive: 15, |
| 9: | C | Static priority: 1, |
| 9: | L | Tasking: 4, 10, 20, 21, 27, 29, 33, 71, 75, 77, 83, 85, 86, 88, |
| 9: | L | Task families: 88, |
| 9: | L | Rendezvous arguments: 29, 38, |
| 9: | R | Tasking too complex: 15, |
| 9: | R | Signals and semaphores: 30, |
| 10: | C | Allowing deferred constants to be set in a separate compilation unit: |
| 10: | C | Have different visibility rules for separate compilation: 30, |
| 10: | C | Separate units should have full upward visibility: 87, |
| 10: | L | Program structure: 16, |
| 10: | L | Separate compilation: 10, 19, 26, 54, 68, 72, 73, 87 |
| 11: | L | Exception handling: 7, 18, 20, 29, 33, 38, 58, 86, |
| 11: | C | Exceptions in declarative parts should propagate up: 86, |
| 12: | A | Type restrictions for generic parameters: 8, 85, |
| 12: | A | Component names as generic parameters: 29, |
| 12: | C | Generics: 20, |
| 12: | L | Generics: 2, 18, 38, 58, 68, 86, 87, 88, |
| 12: | R | Generics: 3, 69, |
| 13: | A | Overlays: 1, 26, |
| 13: | A | Representation of integers as bit fields: 16, |
| 13: | A | Records with overlapping fields: 29, |
| 13: | A | Representation specification of fixed point binary point: 18, 19, |
| 13: | A | Better Fortran interface: 87, |
| 13: | C | Improve alignment specifications: 13, |
| 13: | C | Machine code inserts clumsy: 15, 41, 47, |
| 13: | C | Incorporate representations into type definitions: 27, |
| 13: | L | Record representation: 19, 38, 44, |
| 13: | L | Representation specifications: 19, 27, 56, 88, |
| 13: | L | Machine-code insertions: 27, |
| 13: | L | Unsafe conversion: 88, |
| 14: | A | Timeout on I/O: 11, |
| 14: | A | Fortran-like Formats: 0, |
| 14: | A | Mixed-mode files: 82, |
| 14: | A | A high-level real-time I/O mechanism: 82, |
| 14: | C | EOF not exception: 14, 62, |
| 14: | C | I/O incomplete: 13, |
| 14: | C | Operating system assumed too big: 13, |
| 14: | C | Extend Text_IO: 0, |
| 14: | L | I/O as package: 1, 7, |
| 14: | R | Send_control, Receive_control (in Low_level_IO): 1, |
| 2 : | C | Keywords are overloaded: 87, |
| 2 : | L | Matching keywords (e.g. if -- endif): 87, |

## APPENDIX B: Issues File

Just as the Topics Index cross references TER issues with the Ada Language Reference Manual, the issues file cross references concern found in the LIR's with the LRM.

The issues file is organized by section number of the Preliminary Ada LRM. Under each section number are grouped abstracts of comments relating to that section. The comments are numbered by section number with a serial letter following. Thus, "2.3.A" is the first comment on section 2.3.

The relation of comments to sections is at best approximate, since many issues cross section boundaries. In order, therefore, to make the document more useful, cross-references to other sections are entered under comments. Text processing tools can extract these cross-references and place them under the sections cross-referenced.

The content of an issue abstract is intended to reflect the intent of the comment writer; no evaluation of its substance is intended. Several comments which make the same general point are indexed under one issue abstract; they may nonetheless differ in detail. Although the abstracts are intended to be informative and useful apart from the comments, in general it is necessary to read the comment itself in order to understand the analysis, justifications, and suggestions contained in it.

The abstracts are generally self-explanatory. In order to keep them concise, they are often presented as statements of fact even though the point may be debatable (e.g. "tasking is inflexible"). Syntactic terms and reserved words are capitalized (e.g. Exponentiating_operator, Begin). "Presumably" means that the comment writer felt the manual was incomplete (e.g. "labels are presumably in a different name space"). An absolute statement such as "Ada forbids subscripting of functional values" may safely be taken that the author of the comment felt this construct should not be forbidden. An indication of "(?)" after an abstract indicates that the abstractor feels that he may not have fully understood the intent of the comment.

### Internal Format

The file is organized in such a way as to make automatic processing relatively easy. When formatted versions of the issues file are produced, a copy is put in <TNE-Archive> under the name Issues.Formatted. In order to allow for the variety of output devices, the "Formatted" file is not paginated. since the LIR log is annotated with references to the Issues file, it is possible to see where a particular LIR has been entered and cross-referenced.

There are three kinds of entries: section names, abstracts, and references. Section name entries are of the form:

"$" <section number> <section name>.

Each section of the LRM is represented by its section name as found in the table of contents, even if no abstracts are found under it. Abstracts of comments are of the form:

```
        "%" <section number>.<comment serial letter>
            <comment abstract>  <cross-references>
```

Comment serial letters run A-Z then ZA-ZZ. Cross-references are of the form:
"XX" <section number>" ".

References to other documents are of the form "<references>", where the references are document numbers separated by commas. In certain categories of documents (notably P2Rs), section or page numbers within the document are given in parentheses after the document numbers: these page numbers are usually stripped off before further processing of references.

A representative fragment of the internal form of the Issues file follows:

```
$ 6.7 Blocks

% 6.7.A It should be possible to name all blocks, perhaps uniformly
with loops. XX5.6!
! LIR.066, LIR.222

        <end of abstract>
```

The formatted Issues file contains exactly the same information, but is formatted for human readability. The processed LIR log found in another section of this report contains back-references from LIR's to issue entries.

## 1. INTRODUCTION

### 1.1 Design Goals

1.1.A The language addresses too many conceptual levels: pragmas and separate compilation, for example, are support system functions. XX2.7 XX10.0
LIR.584

### 1.2 Language Summary

### 1.3 Sources

### 1.4 Syntax Notation

1.4.A Clarify the meaning of brackets.
LIR.489

1.4.B The syntax rules should be numbered for easy, unambiguous reference
LIR.605

1.4.C Nonterminals should be capitalized. This helps distinguish syntact
Name from the concept of 'name'.
LIR.637

1.4.D Some better metasyntax for one or more repetitions (with separators
should be used, eg "Name list ," or "Name , ..." for present "Name {, Name}
LIR.637

1.4.E Observing the grammatical distinction between "which" (descriptive)
and "that" (restrictive) would clarify manual explanations.
LIR.638

## 1.5 Documentation

## 2. LEXICAL ELEMENTS

2.0.A There should be a complete lexical grammar, separate and distinct f
the phrase structure grammar, in the LRM.
LIR.639

### 2.1 Character Set

2.1.A Commas are preferred to vertical bars in the syntax. XX3.6.2 XX3.7
XX5.5 XX11.2
LIR.308

2.1.C The Ada character set uses characters reserved for national use
according to ISO 646. "!" in particular should be removed.
LIR.394

### 2.2 Lexical Units and Spacing Conventions

2.2.A The symbol "=>" should be replaced by ":=" in aggregates and "then"
in case-like statements. XX3.6.2 XX3.7.2 XX5.5 XX9.7 XX11.2
LIR.205 LIR.313

### 2.3 Identifiers

2.3.A Underscores should be allowed but not be significant.
LIR.346

2.3.B For compatibility with other naming conventions (GCOS, CP-6
Multics), "$" should be a letter and terminal "_" should be allowed.
LIR.482

## 2.4 Numbers

2.4.A Real number literals should not require a decimal point or trailinc or leading zero.
LIR.040 LIR.425

2.4.B A number's form should not affect its type: "23" should be a legal floating number, and "1.2E6" a legal integer.
LIR.148

2.4.C There is no way to write a Boolean-array constant (bitstring) as a numeric literal.
LIR.245 LIR.245

2.4.D Based integers need not he built in. Numrd("16#2A") suffices.
LIR.294

2.4.E "_" for spacing should be permitted anywhere within a number.
LIR.320

## 2.5 Character Strings

2.5.A The interchangeability of " and % as string delimiters causes unnecessary confusion.
LIR.084 LIR.104 LIR.217 LIR.493

2.5.B There should be a distinct convention for character literals, eg 'a', $a, #a, rather than allowing the length-one string to stand for them.
XX4.4
LIR.297

2.5.C The character """ does not distinguish opening and closing and is not Steelman approved. <<...string...>> is suggested. XX2.1
LIR.314

2.5.D What is the syntax of character_literal? XX3.5.1
LIR.499

## 2.6 Comments

2.6.A Embedded comments desired.
LIR.345 LIR.402 LIR.560

2.6.B Comments should appear at the beginning of lines, terminated by "--
LIR.193

## 2.7 Pragmas

2.7.A Pragmas that alter the semantics of programs should be deleted or incorporated as language features, eg Environment, Include. XX8.6 XXB
EVR.001(p15) EVR.002(#215) P2R.022(#07) LIR.069 LIR.160
LIR.532

2.7.B Redundant pragmas should be deleted. Pragmas which never influence compilation should not be addressed in the LRM.
EVR.002(#302) P2R.022(#05)

2.7.C There should be some indication of the compulsory strength of a pragma. The programmer should be notified whenever a pragma is not acted upon by the compiler.
EVR.001(p15) EVR.005(#3.0)

2.7.D There should ba a conditional compilation pragma.
LIR.036

2.7.E There should be a pragma requiring compile-time initialization.
LIR.249

2.7.F Pragmas should have well-defined scopes and syntactic positions.
LIR.292

2.7.G There should be a sliding scale of space vs. time optimization.
LIR.300

2.7.H Pragmas should be part of the support system, not the language.
LIR.584

2.7.I Pragmas should be allowed static expressions or at least names as parameters.
LIR.626

2.7.J The Environment and Suppress pragmas have "name" parameters, despit the syntax definition. XX8.6 XX11.6
LIR.217

## 2.8 Reserved Words

2.8.A The word 'delta' should not be reserved: it is too common a variable name. XX3.5.5
LIR.401

# 3. DECLARATIONS AND TYPES

3.0.A Explicit type parameters should be permitted for any user-defined t arrays should not be a special case. Type parameters should be bound for individual variables of the type at the point of their allocation (e.g. at point of declaration for non-access types).
EVR.002(#103) P2R.013(#02) P2R.018(#01) P2R.027(#05) P2R.027(#06)
P2R.038(#06) P2R.039(#05) P2R.046(#03) LIR.142

3.0.B Name equivalence should apply to types.
P2R.012(#03)

3.Ø.C PL/I-like based variables are desired.
LIR.129

3.Ø.D The concept of "elaboration" is not fully and clearly defined.
XX7.Ø XX9.Ø
LIR.143 LIR.325

3.Ø.E Types should have attributes such as 'Is.Scalar for use in restrictive assertions in generics. XX12.Ø XXA
LIR.258

3.Ø.F Full functional values are desired: variables should be allowed to have functions as values; functional arguments and values should be allowec Some errors will be undetectable by the compiler, but integration into the language is safer than machine insertions.
LIR.335 LIR.369 LIR.596

3.Ø.G Parameters of types should be explicit; there should be a defaultir mechanism for them.
LIR.142

## 3.1 Declarations

3.1.A Ada, like Pascal, requires declaration before use. This is a semantically empty restriction on program structure.
P2R.Ø13(#Ø5)

3.1.B Declarations should start with a keyword. This makes parsing and reading easier.
LIR.63Ø

## 3.2 Object Declarations

3.2.A There should some way to force static allocation of local variables. XX13.Ø
LIR.326

3.2.B If No_Value_Error is to be removed, all objects should be required to be initialized explicitly or implicitly, eg, integers to Maxint, access objects to Null.
LIR.426

3.2.C The semantics of constants should more completely specified (cf. private types, constant record components). XX3.7.1
LIR.485

3.2.D The right hand side of object_initialization should allow an expression_list.
LIR.6Ø5

3.2.E Constants set at load time are needed.
LIR.137

## 3.3 Type and Subtype Declarations

3.3.A See 3.6.A.
P2R.039(#23)

3.3.B The Ada set type is adequate for bit string operations, but is not an acceptable substitute for sets. True Pascal-like sets would be a valuat aid to readability and conceptual clarity in complex flow-of-control proble
EVR.005(#13.0) P2R.002(#01) LIR.058

3.3.C For implementation of library packages, a mechanism to defeat stror typing should be provided. This could be provided by the "any" type.
P2R.036(#13)

3.3.D It should be possible to define initialization and finalization routines for types.
EVR.003(#3.3) P2R.046(#02)

3.3.E Constraints do not appear to be the distinguishing feature of subtypes. There is some confusion in the definition. Constraints should t reformulated so that types can genuinely be composed. Without this, the important notions of modularity à la Parnas are difficult to express.
EVR.003(#3.1) P2R.013(#01) P2R.039(#06)

3.3.F Incomplete type declarations are unnecessary since the identifiers thus declared must needs be types in the contexts where they are used.
LIR.054

3.3.G Type declarations should be able to provide default initial values for all types.
LIR.164 LIR.355 LIR.497

3.3.H It seems that "type t is range 0..10" defines t as a subtype of an anonymous base type. What is that type? What arithmetic is used for t anc intermediate expressions of type t expressions? XX3.5.4
LIR.266

3.3.I Subtypes should be eliminated as defeating strong typing, in favor of derived types alone.
LIR.312

3.3.J Convenient and intuitive syntax for sets (arrays of booleans) would be very helpful. Pascal sets liked.
LIR.400

3.3.K Are incomplete type declarations restricted to mutually dependent access types? What can you do with them?
LIR.495

3.3.L It should be made perfectly clear that a subtype is compatible witt its parent.
LIR.496

3.3.M Subtypes are types with the set of values restricted. It should also be possible to restrict the attributes. XX7.4
LIR.561

3.3.N The difference between "type T1 is new Integer" and "subtype T2 is Integer" appears to be only that in certain positions T1 objects must be explicitly converted. Does this slight difference justify having both concepts?
LIR.6Ø5

3.3.O Subtypes should never be implictly introduced via type derivation e
XX3.4
LIR.615

3.3.P Name equivalence of array types forces a proliferation of type name
Array types should be subject to structural equivalence. Type specificatic should be allowed as well as type names for formal parameters. XX3.6 XX4.6
LIR.221

## 3.4 Derived Type Definitions

3.4.A The facility for implicit definition by inheritance of operations f underlying types using the "new" type declaration should be flexible enough allow (encourage) alternative definitions of individual operations when the default is inappropriate. XX7.4
EVR.ØØ2(#2Ø6) P2R.Ø27(#Ø7)

3.4.B After the declaration "Feet is new Integer", the language automatically derives an unwanted operation that multiplies two values of t Feet, returning a value of type Feet.
P2R.Ø13(#Ø3)

3.4.C Derived types should inherit constants from their ancestral type.
LIR.Ø29

3.4.D Deriving from a private type should presumably be forbidden. XX7.4
LIR.486

3.4.E Subprograms declared after derivation are presumably not inherited.
LIR.498

3.4.F Does a type derived from an access type share the parent's collection? Does it inherit the length specification? What attributes doe it inherit? XX3.8 XX13.2
LIR.562

3.4.G Inheritance of operations by derived types leads to much confusion. Automatic inheritance by conversion is superior. Inter alia, it allows for mixing of types and derived types as appropriate.
LIR.2Ø4

## 3.5 Scalar Types

3.5.A See 3.5.5.G.
LIR.113

3.5.B 'Ord and 'Val are subject to pathologies and are not fully defined.
LIR.116 LIR.063

3.5.C There should be a 'Range attribute: A'Range == A'First..A'Last, or perhaps a type name should be able in general to stand for T'Range. XX3.6 XX3.3
LIR.027 LIR.150 LIR.223 LIR.235 LIR.636

3.5.D Pred and Succ should be overloaded functions rather than functional attributes of types.
LIR.155

3.5.E 'Pred, 'Succ, 'Ord, and 'Rep (and, eventuellement, 'Range) should b allowed for objects as well as types. This would make anonymous types more useful. XXA
LIR.223 LIR.428

3.5.F What are 'First and 'Last of empty ranges? and 'Ord of 'First?
LIR.220

3.5.G There is no way to write an empty range of a type with just one val
LIR.220

## 3.5.1 Enumeration Types

3.5.1.A The extent to which overloaded literals' meaning is determined by contextual information is left unclear.
LIR.074

3.5.1.B Are a and "a" equal enumerals? What is the I/O form? XX14.3.7
LIR.362

3.5.1.C Unordered enumerated types are desired. Why should, eg, colors be ordered? Of course, this would require the facility of using type names to represent the whole collection of objects of the type. XX3.6
LIR.600

## 3.5.2 Character Types

## 3.5.3 Boolean Type

## 3.5.4 Integer Types

3.5.4.A The type "integer" introduces unfortunate machine dependency.
LIR.091 LIR.154

3.5.4.B Integers should be pure ranges (not derived from Integer).
LIR.383

3.5.4.C Integer types are derived from one of Short_Integer, etc.: can a Short_Integer value be added by standard "+" to an Integer value? If yes say so; if no, portability suffers. XX4.5 XX6.6
LIR.500

3.5.4.D Can Short_Integers be assigned (converted?) to Integers?
LIR.501

3.5.4.E Unsigned integers desired. XX13.0
LIR.613

3.5.5 Real Types

3.5.5.A The implemented fixed point delta should be an integral divisor of the specified delta. The fixed point range specification should constrain rather than determine the implemented representation.
EVR.002(#202) P2R.025(#03) P2R.028(#03) P2R.039(#01) P2R.044(#01) P2R.044(#02) LIR.585

3.5.5.B Fixed point literals and values should not be rounded implicitly.
EVR.002(#202)

3.5.5.C It should be possible to specify the range of exponents.
EVR.005(#2.2) P2R.039(#02)

3.5.5.D A semantic model of Ada numerics is needed.
LIR.020

3.5.5.E The delta-type accuracy constraint syntax incorrectly specifies range constraint as optional.
LIR.105 LIR.270 LIR.403 LIR.605

3.5.5.F Fixed-point arithmetic should support general scaling.
LIR.232

3.5.5.G Range constraints are simultaneously too vague in specifying endpoints (open vs. closed intervals) and too restrictive in requiring exact endpoints (hampering development of efficient machine independent coc
LIR.113

3.5.5.H Ranges should be closed, not open.
LIR.316

3.5.5.I Floating precision should be specified not by digits, but by relat: delta, which is more accurate and more useful.
LIR.330

3.5.5.J Define the terms "floating point type" and "fixed point type".
LIR.502

3.5.5.K Are T'Small etc. defined by the range and accuracy constraints (one or both?) alone or also by the implementation?
LIR.503

3.5.5.L What are T'Small and T'Large for fixed types?
LIR.504

3.5.5.M Do not complicate ranges with open vs. closed etc.
LIR.208

## 3.6 Array Types

3.6.A The distinction between type mark and discrete range specification of array bounds makes for unnecessarily complex rules: array(T) should be equivalent to array(T Range T'First..T'Last). This would also be a more convenient notation in many cases. XX3.5
P2R.039(#23) LIR.152 LIR.476 LIR.593 LIR.612

3.6.B Ragged arrays are desired.
LIR.336

3.6.C Arrays should be stored by rows.
LIR.370

3.6.D The syntax and semantics of multiple-index arrays should be clarifi is array(a,b) entirely equivalent to array(a) of array(b)? In particular, is the type of subarrays? Can the notation A(x,y) be used for arrays of arrays? Can catenation be applied to multidimensional arrays interpreted a (one-dimensional) arrays of arrays? Is 'Length(2) meaningful for arrays of arrays? Why must all or no index positions be specified by discrete ranges XX4.1 XX4.5.3
LIR.487 LIR.506 LIR.513 LIR.567 LIR.615

3.6.E The integer i in 'First(i) should be required to be static (if it is not, what exception does a bad value raise?): the rare dynamic case can be handled with a Case statement.
LIR.505

3.6.F If T1 is an array of T3's, how do we declare a subtype of T1 with index constraints on T3 (another array type)? Extensive discussion. Discussion of the interaction of arrays of arrays and private types. Components of a structured type must be subtypes; a clear set of rules for coercion from a type to a subtype must be given. Forbid subtypes of subtyp Let the nonterminal type_mark denote a subtype: define coercion rules for i Disallow index ranges in an array_type_definition. Give the unconstrained integers and reals type names.
LIR.615

### 3.6.1 Index Range of Arrays

3.6.1.A Arrays should be one-origin by default.
LIR.043

3.6.1.B The rule on index ranges of arrays in records seems to exclude constant-length arrays within records with index range determined by external nonconstants, eg, Record S: Array(1..x) End, where x is a variable (not a record field). Bounds determinable at type declaration elaboration should be allowed.
LIR.508

## 3.6.2 Aggregates

3.6.2.A Having to specify values for all components of an aggregate is both awkward and inefficient.
LIR.008(s3.3) P2R.046(#01) LIR.163 LIR.361

3.6.2.B Mixed array aggregates with array bounds which are not static result in unnecessary run time inefficiencies.
OPA.013

3.6.2.D Initializing multidimensional non-constant aggregates is painful in the current syntax.
LIR.134

3.6.2.E Component association syntax should use ":=" rather than "=>" for consistency. XX2.2
LIR.135 LIR.313

3.6.2.F The use of simple parentheses to denote aggregates is hard on the parser and strains the type disambiguation mechanism. XX4.6
LIR.999

3.6.2.G Is 5 | Others a legal component_association?
LIR.509

3.6.2.H The syntax of Choice should indicate that the expressions on the right hand side must be static (italicized prefix Static).
LIR.605

3.6.2.I The use of "|" is confusing. A preferred syntax for aggregates would be, eg, (1,3,1) for positional, and ((1,3)=>1,(2)=>3) for named component selection.
LIR.205

3.6.2.J Null aggregates require a superfluous value: (1..0=>dummy).
LIR.220

## 3.6.3 Strings

3.6.3.A Maximum string length should be an independent system attribute not Integer'Last.
LIR.117

3.6.3.B Strings should be of fixed size but variable length or heap-allocated.
LIR.126 LIR.085

3.6.3.C Ada 'Strings' are not the same beast as strings in other languages. Better strings (variable length) are needed: perhaps access type with special lexical/syntactic form.
LIR.177 LIR.265 LIR.365 LIR.404 LIR.456

3.6.3.D Better strings are wanted: in particular, strings of different (physical) length should be type-compatible.
LIR.386

3.6.3.E Null strings should be permitted. XXC
LIR.456

## 3.7 Record Types

3.7.A The rules for allowable (dynamic) dependencies among record components are too restrictive.
LIR.008(s2.1)

3.7.B Distinction between discriminants constrained statically (at declaration) vs. dynamically (on initialization or assignment) causes confusion.
LIR.008(s2.2)

3.7.C Current semantics of record discriminants interfere with efficient implementation of parameter passing.
LIR.008(s3.3)

3.7.D It should not be possible to assign the discriminant of a variant record without assigning the entire record.
EVR.002(#201) P2R.013(#04) P2R.015(#02) P2R.046(#01)

3.7.E Union types can appear only as variant record fields. The general union type approach is preferred over variant records. XX3.3
EVR.003(#3.6) P2R.013(#02) P2R.015(#02) P2R.026(#03) P2R.046(#01) P2R.046(#07) LIR.634

3.7.F The same field name should be able to appear in different variants of a record. Representation specifications would need revision. XX13.4
LIR.018 LIR.165 LIR.213

3.7.G Null records should be forbidden.
LIR.034

3.7.H There should be a dummy field name for constant record components which are never referred to.
LIR.457

3.7.I Only one dynamic array should be allowed per record, and it should be the last component, as for variants. Requiring explicit access implementations for the general case makes costs more apparent.
LIR.510

### 3.7.1 Constant Record Components and Discriminants

3.7.1.A Eliminate (non-deferred) constant record components.
OPA.017

3.7.1.B Constants as well as deferred constants should be allowed as discriminants of records.
LIR.149

3.7.1.C Define "complete record assignment" explicitly.
LIR.511

3.7.2.D Dynamic arrays should cause immediate storage overflow if their maximum size is too great (eg Integer'Last).
LIR.512

## 3.7.2 Variant Parts

3.7.2.A There should be a way to set a record discriminant, presumably in the Unsafe_Programming package. XX13.10
LIR.385

3.7.2.B Must the discriminant variable be declared in the record?
LIR.458

## 3.7.3 Record Aggregates and Discriminant Constraints

3.7.3.A Discriminant constraints and record aggregates are semantically distinct and should therefore be syntactically distinct as well.
LIR.162

3.7.3.B All deferred constant components should be specifiable through discriminant constraint specification. XX3.7.1
LIR.588

## 3.8 Access Types

3.8.A Initialization of elements of access types should not be required
the point of allocation.
EVR.002(#203) P2R.019(#02) OPA.005 LIR.477

3.8.B There should be a free operation on access objects.
EVR.003(#2.3) EVR.005(#4.0) LIR.037 LIR.127 LIR.212
LIR.250 LIR.408 LIR.566

3.8.C It should be possible for one access type to refer directly to another access type.
P2R.015(#01)

3.8.D The built-in storage allocation mechanisms are much too restrictiv
and do not allow user-defined mechanisms. Extensive proposals.
LIR.123 LIR.055

3.8.E The rules for access constants (and therefore also access in parameters) severely constrain use of access types; nonetheless, constants of access types are not truly read-only. XX5.2.3 XX6.3
LIR.101 LIR.132 LIR.200 LIR.216 LIR.538

3.8.F Discriminants in access variables should be changeable.
LIR.055

3.8.G The access section is vague.
LIR.102

3.8.H It should be clear whether there is a garbage collector.
LIR.233

3.8.I PL/I-like separation of declaration and 'allocation' of storage areas is preferred.
LIR.246

3.8.J It should be possible to point to static data.
LIR.337 LIR.390 LIR.414

3.8.K Conversion to ancestral type of an object of derived access type can violate strong typing and create dangling references. XX3.4
LIR.348

3.8.L There should be provision for allocating access-type objects at compile time when possible.
LIR.416

3.8.M It would be nice if access types could be efficient for tightly packed data, using pointers into fields of a word and minimal-length pointers.
LIR.417

3.8.N Any variable or field of access type should be initialized to Null if it is not explicitly initialized at declaration. XX3.2
EVR.002(#203) P2R.019(#02) OPA.005 LIR.478

3.8.O A reference count scheme should be used for deallocation. (?)
LIR.479

3.8.P Access type model preferred to traditional pointers.
LIR.161

3.8.Q Anonymous access types are apparently useless. Shouldn't they therefore be illegal (either in the syntax or the semantics)? XX3.3
LIR.201

## 4. NAMES, VARIABLES, AND EXPRESSIONS

4.0.A Functions' values cannot be subscripted, sliced, or selected.
LIR.097 LIR.156

## 4.1 Names

4.1.A In the case of generic parameters, generic associations, and renam declarations, the syntax is presently incomplete. The syntax formula "[name.]designator" does not cover the case of a functional attribute such T'SUCC or T'ORD. XXZ
OPA.018

4.1.B The syntax of "name" excludes designators. XX8.5 XX12.1 XX12.2
LIR.136 LIR.225

4.1.C Subprogram calls (returning access types) should be names. Consider P(a).all := ....
LIR.271

4.1.D It should be made clear that a name cannot be used for more than one purpose in a scope: variable, type, function, etc.
LIR.483

4.1.E There are examples of 'simple names', but what is the definition?
LIR.514

4.1.F Due to limitations concerning use of "designator", it would not be possible to use stubs within the subprogram body when overloading an operat since the designator cannot subsequently appear in the visibility list of t sub-unit body. XX8.0
LIR.203

### 4.1.1 Indexed Components

### 4.1.2 Selected Components

4.1.2.A Dot selector notation can productively be considered a variant syntax of function calling.
LIR.133

4.1.2.B Implicit dereferencing is disliked.
LIR.250

4.1.2.C The concept of user-defined type attribute is unnecessary. XXZ
LIR.273 LIR.619

4.1.2.D Component selection syntax should be uniform with that of function calling and array indexing (ie parentheses).
LIR.334

4.1.2.E Must not the parenthesized index expression of an array level immediately follow the array identifier and precede the identifier of the next level? Say so.
LIR.490

4.1.2.E The syntax of selected components provides no way to distinguish among the overloadings of a name by signature when type attributes would be ambiguous. XX3.3 XX6.6
LIR.469 LIR.515

## 4.1.3 Predefined attributes

4.1.3.A The notation for user-defined and predefined attributes should be the same; dot notation is preferred.
LIR.034

4.1.3.B Editorial: identifiers are not subprograms, but their names.
LIR.491

## 4.2 Literals

4.2.A Enumeration literals should be quoted in order to distinguish them from variables.
LIR.059 LIR.153

4.2.B There should be a Null value for all types, which would cause an exception to be raised if calculated with. XXll.1
LIR.343

4.2.C User-defined literals are needed. Currently, too many explicit conversions are needed (consider private types). XX4.6 XX7.4
LIR.186

## 4.3 Variables

4.3.A Suggests that 'name' include '<name> . all', and 'name' be substitu for 'variable' in the definition of 'primary', thus eliminating the syntact term 'variable'.
LIR.272

4.3.B Slicing is clumsy: start and length ranges and default endpoint ranges are desired, ie arr(first_loc SIZE len) and arr(..cutoff).
LIR.338

4.3.C .value or .val preferred over .all.
LIR.356

4.3.D Clarify the syntax and description of Slice_variable, Name, and Variable.
LIR.455

4.3.E An access variable should denote the object, not the access. A special syntax should be used for access assignment.
LIR.480

4.3.F Replace array.all with array(all). (?)
LIR.481

## 4.4 Expressions

4.4.A Regarding an expression as possibly a one-component aggregate of i: type leads to ambiguities, difficulty of implementation, and opaque code. XX2.5
LIR.067 LIR.085 LIR.494

4.4.B Some easy way to perform such operations as incrementation is desired. The suggestion is a primary 'self' or '_' as a shorthand on the right hand of an assignment for the left hand side, thus var:=self+1. XX5.:
LIR.261 LIR.378

## 4.5 Operators and Expression Evaluation

4.5.A The precedence rules for user defined operators are the same as th for the built-in operators. The lack of implicit semantics for overloaded operators can lead to programming errors.
P2R.010(#01)

4.5.B The primitive floating point operations of floor, fraction, and modulus are missing and cannot correctly be implemented within the languag
XXC
LIR.104

4.5.C Expression evaluation order should be left to the compiler.
LIR.035 LIR.090

4.5.D The relational operators should be represented by alphabetical keywords rather than graphics and graphic digraphs. Suggests EQ, NE, LT etc. Also suggests making all operators the same length. XX2.2
LIR.307

4.5.E The basic bitstring operations And, Or, Shift, and Rotate are lacking. XX3.3
LIR.342

4.5.F Unary operators should have the highest precedence.
LIR.357

4.5.G Expressions should have their mathematical meaning, with order of evaluation left unspecified, except that parentheses should restrict that order, and a pragma should be provided to cause code to choose the most accurate evaluation order at runtime.
LIR.438

4.5.H The types of the two operands of logical, adding, and multiplying operators should presumably be the same (but cf. fixed-point multiply).
LIR.516

4.5.I Undefined sequences of operator characters should be operator lexemes definable by the user (having some fixed precedence). Consider, e +:=. XX2.2 XX5.1
LIR.594 LIR.627

4.5.J Why are In and Not In omitted from the operator (precedence) table? Is this to imply that they are not overloadable? XX4.5.2
LIR.217 LIR.605

4.5.K Operators with partial evaluation (cf. And then) are desired: a Imp == Not a OrElse b; a Default b == if not null(a) then a else b. XX5.4.1
LIR.192

4.5.L The non-terminals Exponentiating_operator and Logical_operator are never used. XX4.4
LIR.217

## 4.5.1 Logical Operators

4.5.1.A Precedence rules for "And" and "Or" should be defined.
EVR.004(#4) P2R.044(#03) LIR.230 LIR.437 LIR.448

4.5.1.B Can logical operators have boolean arrays of differing bounds as operands?: what are the result's bounds? (?)
LIR.517

## 4.5.2 Relational and Membership Operators

4.5.2.A The definition of any one of the four ordering operations should automatically define the other three so that A>B iff B<A, A>=B iff not A<B, A<=B iff not B<A.
EVR.002(#204) P2R.038(#07)

4.5.2.B The implicitly defined aggregate equality should be defined in term of the equality of its component types.
EVR.002(#205) EVR.007(s2.7) LIR.006(p02) OPA.003

4.5.2.C If a component of a composite type is of a restricted type assignment is not defined for the composite type. If a component of a composite type is of a restricted type, comparison for equality or inequali is not defined for the composite type (unless equality is defined explicitl in the package defining the type).
OPA.004

4.5.2.D Presumably, a "corresponding range..." means one of the same type as the first argument to In. (4-7 line 13)
LIR.565

## 4.5.3 Adding Operators

4.5.3.A Catenation should apply to bitstrings.
LIR.245

4.5.3.B What is meant by "the accuracy of the result is the accuracy of the operand type"?: the type's constraint, or the mathematically determinec accuracy? What is the accuracy of an operation between two values of the same type but different accuracy constraint?
LIR.518 LIR.605

## 4.5.4 Unary Operators

## 4.5.5 Multiplying Operators

4.5.5.A The definitions of mod and integer division violate the mathematical property a mod b = (a-b) mod b. The current operation is in fact the "remainder" operation: both are needed.

```
EVR.007(s2.5)  P2R.025(#02)  P2R.038(#01)  P2R.046(#19)  LIR.010
LIR.079        LIR.104       LIR.042       LIR.079       LIR.176
LIR.317        LIR.358
```

4.5.5.B Mod and Rem should be functions, not infix operators.
LIR.317

4.5.5.C Mod should be everywhere well-defined.
LIR.439

4.5.5.D Presumably fixed-point values of different type can be multiplied. XX4.5
LIR.516

4.5.5.E To multiply values of distinct fixed-point types, you apparently have to convert them, which loses accuracy: qualification of the result shc be sufficient. XX3.5.5
LIR.195

## 4.5.6 Exponentiating Operator

4.5.6.A In Integer**x, must x be positive (per 4.5.6) or non-negative (per C-1)? XXC
LIR.519

## 4.6 Qualified Expressions

4.6.A The notation "type_name (...)" is used both for resolving ambiguit and for explicit conversion, which can confuse the meaning of widely different semantics. Bad interactions with parameter semantics.
LIR.011 LIR.111

4.6.B The syntax can lead to ambiguous expressions.
LIR.162

4.6.C It should be possible to overload type names as conversion functions. XX6.6
LIR.418

4.6.D Parenthesis notation is confusing. cf. use of ".". (??)
LIR.483

4.6.E There should be some way to convert to the underlying type without knowing its name. This is particularly useful for private types in their own modules to reduce the effects of a representation change.
LIR.599

4.6.F Why are derived type conversions not allowed on the left-hand side
assignments?

$ 4.6.1 Explicit Type or Subtype Specification

$ 4.6.1.A It is hard to see when qualification would indeed be needed in th
Instr_Code(Fix) case--presumably I has some type, which would disambiguate
Fix, unless perhaps I is an overloaded function of no arguments, certainly
rather obscure case for an example! Consider using the example of the ranc
part of an array declaration.
LIR.520

## 4.6.2 Type Conversions

4.6.2.A The semantics of real-integer conversion are left vague.
LIR.105 LIR.521

## 4.7 Allocators

4.7.A Does New supply additional storage or provide a pointer into a
predefined area set up by the compiler?
P2R.004(#03)

4.7.B In present Ada, an allocator must provide initialization of
dynamically allocated objects. Consider the possibility of providing a pa
aggregate limited to discriminants (as for constraints).
OPA.006 LIR.163 LIR.589

4.7.C The user should be able to define his own allocator, and redefine
the system allocator for his own types.
LIR.055 LIR.025

4.7.D The Keyword "new" is overused: for allocation, generic
instantiation, and type derivation. XX3.3 XX12.2
LIR.025 LIR.598

4.7.E Storage areas as well as individual objects should be explicitly
allocated at runtime independent of declarations.
LIR.246

4.7.F It should be possible to allocate without initializing.
LIR.477

## 4.8 Static Expressions

4.8.A The language definition should make it clear that static expressior
may be used everywhere literals may. Static expressions should be just thc
expressions evaluable at compile or load time. The value of constants canr
always be determined before the corresponding scope entry. Similarly,
predefined operators, functions, and attributes are not always compile time
evaluable. Static expressions should not be restricted to predefined
operations, functions, and types. The definition of types is always known
during compilation. User defined functions are compile time evaluable unde

the same circumstances as predefined ones.
EVR.002(#107) COM.003

4.8.B Case (f) should presumably be restricted to constants initialized by static expressions and static indices in indexed components.
LIR.522

4.8.C Despite (d), not all predefined attributes are static.
LIR.217

## 5. STATEMENTS

5.0.B Present Ada forbids go to out of a block but permits exit and retu statements. Implementation problems exist when there are tasks local to t block.
OPA.001

5.0.C Sequences of statements should be allowed to have a value, the value of the last expression/statement. XX5.4
LIR.341

### 5.1 Assignment Statements

5.1.A The prohibition against altering discriminants of access variables is a confusing irregularity.
LIR.008(s2.2)

5.1.C The symbol "=" should be used for assignment.
LIR.315

5.1.D There should be an 'exchange' operator, ':=:'.
LIR.377

5.1.E There should be multiple assignment, 'a,b:=3': compute all destinations before any assignments.
LIR.141 LIR.432

5.1.F It should be possible to combine a binary operator with assignment la Algol-68, C), thus x:*=2 doubles x. This is particularly useful with l left-hand sides. XX4.4
LIR.468 LIR.614

#### 5.1.1 Array and Slice Assignments

5.1.1.A Overlapping slice assignment should be permitted, with copy semantics.
LIR.092 LIR.257

5.1.1.B Is assignment between variables of the same multidimensional array type with indices specified by type marks and with the same number of components, always allowed even if the arrays are of different shape? If multidimensional arrays are considered strictly equivalent to an array of subarrays, the problem does not arise. XX3.6
LIR.523

## 5.1.2 Record Assignments

5.1.2.A The current rule allows the discriminant of a record within a recor denoted by an access variable to be altered. Is this a loophole?
LIR.210

## 5.2 Subprogram Calls

5.2.A There are too many ways to make a procedure call and define aggreg values.
P2R.015(#04)

5.2.B A subprogram_call_statement should be explicitly forbidden or explicitly permitted to call a function or a value returning procedure.
LIR.021(p02)

## 5.2.1 Actual Parameter Associations

5.2.1.A The indication of parameter mode on call should be required even without keyword association.
LIR.262

5.2.1.B Keyword parameter association is liked.
LIR.267

5.2.1.C Parameter mode in calls and specifications should have similar syntax. In, etc. preferred for both.
LIR.329

5.2.1.D Mode should not be distinguished in actual parameter syntax.
LIR.347

5.2.1.E What is the definition of a "qualified variable"? XX4.6
LIR.524

5.2.1.F The order of evaluation of subprogram parameters should be specifie as undefined to allow optimization and reduce the complications implied by variety of calling syntaces.
LIR.214

## 5.2.2 Omission of Actual Parameters

5.2.2.A There should be some placeholder argument specifying the default value and not requiring naming the remaining positional parameters.
LIR.568

## 5.2.3 Restrictions on Subprogram Calls

5.2.3.A The aliasing restriction should be statically defined.
LIR.082 LIR.158

5.2.3.B All aliasing should be prohibited.
LIR.158

5.2.3.C Aliasing via parameter passing should be allowed.
LIR.368

5.2.3.D Aliasing by way of access objects is inevitable and undetectable. It should not be prohibited.
LIR.538

5.2.3.E How strict is aliasing detection?
LIR.581

## 5.3 Return Statements

## 5.4 If Statements

5.4.A When can a type derived from Boolean not function as a condition ir If statements? XX3.5.3
LIR.030

5.4.B Conditionals should be allowed as expressions. XX4.4 XX5.5
LIR.340 LIR.595 LIR.635

5.4.C There should be some way besides Goto to have common actions in branches of an If: else Else construct suggested.
LIR.434

5.4.D There should be a simple syntax for multiple End If's: End If * 3?
LIR.434

### 5.4.1 Short Circuit Conditions

5.4.1.A "And then" and "or else" should be allowed in any boolean expression: current syntax within if statements does not even allow groupir with parentheses. Their precedence should be specified. Note also that current syntax is not LALR(1) unless And then is made a special case in th lexical analysis.
P2R.039(#04) P2R.043(#11) P2R.046(#18) LIR.121 LIR.192
LIR.199 LIR.443 LIR.608

5.4.1.B Since the compiler should feel free to reorder evaluation, "and then" and "or else" are superfluous: they should be the normal interpretation of "and" and "or".
LIR.035 LIR.050 LIR.073 LIR.243 LIR.230
LIR.274

5.4.1.C Although partial evaluation of boolean expressions should be the rule in conditionals, full evaluation should be the rule in expressions.
LIR.243

5.4.1.D Short-circuit conditions should be named "and" and "or"; boolean operations should be called "&" and "|". XX4.5.1
LIR.205

## 5.5 Case Statements

5.5.A Ada requires non-manifest expressions as selectors. This restricts the order of testing which degrades optimization.
P2R.039(#11) P2R.046(#17)

5.5.B Does the keyword "of" add anything useful to the form of this statement?
P2R.019(#07) P2R.039(#22)

5.5.C Change the syntax. Suggests Pick...When => ....
LIR.389

## 5.6 Loop Statements

5.6.A An Until condition Loop statement should be added to the language.
P2R.019(#06) LIR.065

5.6.B A variable increment should be specifiable on a Loop.
P2R.019(#09) LIR.012 LIR.044

5.6.C While is unnecessary: Exit suffices.
P2R.033(#01) LIR.251 LIR.275

5.6.D It should be possible to define and use loop indices outside the loop: currently, their scope is unclear and seemingly not very useful.
P2R.035(#05) LIR.044

5.6.E Loop labels should not look like Goto labels; nor should their scope extend outside the loop body. What is the identifier in "end loop [identifier]"?
LIR.151 LIR.222 LIR.539 LIR.602 LIR.606
LIR.616 LIR.632

5.6.F For loop over sets desired. XX3.3
LIR.400

5.6.G Loop parameters should be accessible to (outside??) exception handlers.
LIR.433

5.6.H Loop indices should require explicit declaration as such.
LIR.467

5.6.I Loop indices should be of type Integer if not otherwise known from context, on analogy with array bounds, as should other ranges. XX3.5 XX3.
LIR.507

5.6.J User-defined iterators are needed for abstraction: this may imply need for functional arguments (one LIR says yes, the other no). XX3.0
LIR.596 LIR.634

5.6.K More loop types are wanted: Until (While Not) and loop test at the bottom.
LIR.185

5.6.L Loops should be generalized to allow actions ("adjustments" or "epilogues") after the exit. Loop labels would no longer be necessary. Perhaps there should also be even more complex loop constructs. Details.
LIR.194

5.6.M If the loop index is not used, it should not have to be written.
LIR.224

5.7 Exit Statements

5.7.A Add Exit Unless to Exit When.
P2R.046(#15)

5.7.B There should be a multiple-level exit with an argument of the numbe of levels.
LIR.145

5.7.C The Exit statement is unnecessary; Goto suffices.
LIR.242

5.7.D Either remove When or generalize it to Raise, Return, and Goto.
LIR.382

5.7.E Allow When after several other types of statements. (Retracted)
LIR.024

5.7.F Keep Exit, but remove When.
LIR.440

5.7.G The loop label should be required in an Exit; thus, if the loop is anonymous, it is patent that premature exit cannot occur.
LIR.602 LIR.632

5.8 Goto Statements

5.8.A Ada allows transfer of control between THEN and ELSE clauses in an statement and alternative sequence of the case statement.
P2R.035(#02) P2R.037(#02)

5.8.B The scope of a label is too small, asymmetric, and irregular.
LIR.112 LIR.122 LIR.548 LIR.569 LIR.617

5.8.C Label and Goto syntax and semantics are unclear.
LIR.072

5.8.D Conventional label syntax ("Label:") is preferred.
LIR.250

5.8.E Replace Goto with a block Exit statement.
LIR.324

5.8.F Labels are presumably in a name space entirely distinct from declared identifiers.
LIR.549

5.8.G Labels should be declared as in Pascal, thus clarifying their scop
LIR.617

## 5.9 Assert Statement

5.9.A Assertions cannot be stated to hold over regions of programs; nor c they be quantified; nor can they refer to the history of variables.
LIR.033

5.9.B The action to be taken when assertions are not satisfied should be controllable.
LIR.033

5.9.C The assert_error exception is unnecessary and dangerous, since it allows violations of assertions to influence program execution.
LIR.071

5.9.D The assert statement is unnecessary and inefficient.
LIR.244

5.9.E The current simple assertion facility suffices.
LIR.209

# 6. DECLARATIVE PARTS, SUBPROGRAMS, AND BLOCKS

## 6. Declarative Parts

6.1.A Top-down organization of declarations is precluded by the linear elaboration of constituents of a declarative part. XX8.4
LIR.447

6.1.B Enforced divorce of declarations and representations is unnatural and error-prone; if it is to remain, representations should follow the bod not precede it. Bodies might also be allowed to be intermixed with declarations. XX6.1
LIR.525 LIR.631

6.1.C There is a syntactic ambiguity whereby module_declaration and module_specification can be confuted.
LIR.540 LIR.624

6.1.D The grammar should express the (context-free) restrictions on declarative parts by introducing variants. XX6.4 XX7.1 XX7.3 XX7.4 XX9.
LIR.624

6.1.E Declarations and bodies should be everywhere interspersible.
LIR.624

6.1.F Different kinds of declarations are too non-uniform in syntax: in s the name precedes, in others it follows, a terminal. LIR prefers consister use of name:declaration.
LIR.625

## 6.2 Subprogram Declarations

6.2.A There should be some way to tell the compiler that a subprogram nee not be compiled to be reentrant or recursively callable.
LIR.032 LIR.039 EVR.005(#11.0)

6.2.B LRM does not specify case of character-string designators, e.g. 'mc
LIR.034

6.2.C The supposed inefficiency of allowing all procedures to be recursi or reentrantly callable is a myth. The present design should be retained.
LIR.166

6.2.D Comma should be allowed to separate parameters in procedure declarations as in calls.
LIR.322

6.2.E The syntax of parameter_declaration should enforce the prohibition on defaults for Out and In Out parameters.
LIR.541

6.2.F There should be functional arguments. XX3.0
LIR.178 LIR.623

6.2.G Operators should have a nonterminal in order to tighten up the definition of designator. The quotes around them appear unnecessary.
LIR.624 LIR.627

## 6.3 Formal Parameters

6.3.A The semantics of parameter passing should be better defined. Both reference and copy semantics are desired.
EVR.001(p07) EVR.002(#102) EVR.003(#1.1) EVR.004(#2) EVR.005(#8.0)
EVR.006(s4.a) EVR.007(s2.3) P2R.014(#02) P2R.015(#05) P2R.022(#03)
P2R.028(#04) P2R.028(#01) P2R.036(#11) P2R.038(#05) P2R.043(#01)
DCR.001 LIR.039

6.3.B Named parameters complicate the language and contribute little. Defaulted parameters appear dangerous: accidental omission of one or more parameters is a source of hard to find errors.
EVR.003(#3.5) EVR.004(#6) EVR.005(#6.0) P2R.043(#09) DCR.002

6.3.C Are the rules for type checking of actual against formal parameters well defined? Is "treated just as in assignment" sufficient?
EVR.004(#1)

6.3.D The semantics of parameter binding should be defined. The definiti should be by copy only.
LIR.017(p01) DCR.001

6.3.E Allow certain formal and actual names to be marked as volatile depending on their behavior. Allow the translator to bind all non-volatile objects by reference if it can thereby gain efficiency.
LIR.017(p03)

6.3.F Parameter passing semantics should be more precisely defined in terms of copying; reference passing would be considered an implementatic that compilers may use when it does not affect the meaning of the program. If this optimization is to be of reasonable applicability, it may be necessary to mark variables shared by several tasks.
OPA.011 DCR.001

6.3.G We do not need both keyword and positional parameters.
P2R.012(#05) DCR.002 LIR.087

6.3.H There is some redundancy is giving three parameter binding classes: IN, OUT, IN-OUT.
P2R.033(#02)

6.3.I Programs should be able to parse their own parameter lists.
LIR.130

6.3.J LIR considers the semantics of In parameters vague.
LIR.077

6.3.K All Out parameters should be strictly undefined after unhandled exceptions.
LIR.086

6.3.L Default parameters have poorly-defined evaluation time: the default value should be calculated at the point of call.
LIR.143

6.3.M Default values for in out and out parameters should be allowed: for in out, the default would be used for in and ignored on out; for out it would be the out value if no other value were given.
LIR.164

6.3.O The current copy semantics are good. The LRM should specify the conditions under which reference implementation will be 'safe'.
LIR.256

6.3.P There should be ways to force reference and copy binding.
LIR.450

6.3.Q Can a formal Out parameter be read after being assigned to?
LIR.542

6.3.R Constraints on actuals should not constrain formals: they should be checked on return. XX3.3
LIR.543

6.3.S Reference binding is not compatible with portability across architectures.
LIR.161

6.3.T The subprogram specification should be able to enforce keyword or positional form of call for uniformity's sake.
LIR.190

6.4 Subprogram Bodies

6.4.A What are "identical subprogram specifications" in this context?
LIR.034

6.4.B Semantics of "Inline" are vague and inefficient, and hard to implem for recursive or separate subprograms; a macro preprocessor is preferred.
LIR.045

6.4.C Can recursive programs be Inline?
LIR.544

6.4.D What is the rule of equivalence between subprogram bodies and declarations? Presumably, it does not distinguish X:T and X:in T, but does distinguish X,Y: T := expr and X: T := expr; Y: T := expr (consider side effects). Presumably types are differentiated by meaning, not by name. (Signature issue)
LIR.217 LIR.545

6.4.D The note about Inline appears to preclude inline expansion when it not requested: compilers might well want to expand, eg, subprograms called once.
LIR.605

6.4.E Is there any difference between the elaboration and the execution o a program?
LIR.605

6.4.F The conformity among unit bodies could be emphasized through a comm syntactic category. XX6.7 XX7.1
LIR.624

## 6.5 Function Subprograms

6.5.A The user should be able to designate the difference between those side effects (i.e., references and assignments to non-local variables) of functions for which the implementation preserve the order and number of occurrences, and those for which the implementation need not.
EVR.001(p08) EVR.002(#105) EVR.006(s4.d) POS.003(p01) P2R.037(#04)
P2R.039(#10) LIR.005(p01)

6.5.B It is not necessary to restrict calls to value returning procedures to assignment statements, initializations, and procedure calls.
EVR.003(#1.5) EVR.004(#7) EVR.006(s4.c) EVR.007(s2.8) P2R.026(#08)
LIR.005(p06) LIR.141

6.5.C Functions should be allowed to perform storage management.
EVR.002(#105) LIR.005(p04) LIR.006(p04)

6.5.D The present definition of functions and value returning procedures does not appear simple to explain or to use.
OPA.008

6.5.E Functions and VRP's should not be distinguished.
LIR.035 LIR.075

6.5.F No_value_error should be raised in the caller's environment.
LIR.088

6.5.G The distinction between VRP's and functions is good.
LIR.253

6.5.H VRP's should be allowed Out parameters.
LIR.344

6.5.I Functions with side effects are useful. Perhaps best eliminate VRP's and add a side-effects pragma for functions.
LIR.431

6.5.J No_value_error for function values should be checked statically anc thus not be an exception.
LIR.546

6.5.K VRP's should be allowed anywhere functions are allowed; they shoulc also be allowed Out parameters (consider file var'ables).
LIR.141

## 6.6 Overloading of Subprograms

6.6.A Overload resolution should be simplified: parameter names should nc be used in overload resolution; type and order of unnamed actual parameters should be used. The meaning of "ambiguous" calls on overload definitions should be clearly defined by the language, not implementations.
EVR.001(p08) EVR.002(#108) EVR.003(#1.6) EVR.007(s2.1) P2R.037(#05)
P2R.043(#07) DCR.002 LIR.131 LIR.076 LIR.087

6.6.B When potentially conflicting declarations appear in the same local scope they should be illegal at the point of declaration.
EVR.002(#100) DCR.002

6.6.D It is unreasonable when outputting a single character to require Put(String ("A")). The TEXT_IO overloaded procedure Put at present forces this.
LIR.021(p03)

6.6.E There should be no overloading on result type.
LIR.277

6.6.F Are defaults part of the subprogram signature?
LIR.547

6.6.G The overloading resolution rules should be clarified.
LIR.582

6.6.H Accidental overloading seems likely (especially with use of libraries); this will weaken type safety. Is prevention to be left to utilities?
LIR.131 LIR.562

6.6.I Entries should be overloadable. XX9.5
LIR.587

6.6.J There should be overloading resolution changes so that there is alv
a simple and unambiguous way of calling a given (especially local) procedu
LIR.076

6.6.K New overloadings can change the meaning of programs. Overloaded
function calls are hard to read. Accidental redeclaration or overloading i
too easy. Therefore, overloading should be resolved by Type and Order onl
non-default parameters; only defaultable parameters should be passable by r
redeclarations must be restricted; literals and parameterless functions mus
almost always be qualified; and other functions may not be overloaded on re
type. XX6.3
LIR.132

6.6.1 Overloading of Operators

6.6.1.A When one overloads =, the operator /= is automatically overloaded. Does any similar relation exist between < and >= or > and <=?
P2R.046(#22) LIR.269

6.6.1.B The properties expected of functions overloading built-in operators
should be defined by the language (eg, < returns boolean; + is commutative)
LIR.114 LIR.269

6.6.1.C Assignment should be overloadable. Consider "receive" as a parameter mode. XX5.1
EVR.003(#3.2) LIR.006(p02) LIR.034 LIR.586

6.6.1.D It may be desirably to provide not-predefined overloadable built-in operators, using symbols such as ++, &&, //.
LIR.269

6.6.1.E What exactly are the overloadable operators? (In?) XX4.5.2
LIR.217

## 6.7 Blocks

6.7.A It should be possible to name all blocks, perhaps uniformly with loops. XX5.6
LIR.066 LIR.222

6.7.B Declare...begin...end is too verbose: a conciser form is preferred.
LIR.339

6.7.C Blocks should be allowed visibility clauses. XX8.3
LIR.484

## 7. MODULES

7.0.A When are package bodies elaborated?
LIR.095

7.0.B Packages with mutually dependent initializations have poorly define semantics.
LIR.096

7.0.C Packages, subprograms, and tasks should be made more similar: Initiate should have the syntax of subprogram call; the visible part of a task should allow variable and module declarations; it should be possible t Initiate packages; subprogram and module should have the same syntax (the formal_part should occur at the end of the visible part); visibility shoulc be specifiable for subprograms. XX6.0 XX9.0
LIR.279

7.0.C The military standard "module" differs from the Ada module: military standard modules are compilation units.
LIR.323

7.0.D A package specification should be able to be associated with more than one body, with a choice at link time. XX10.0
LIR.411

## 7.1 Module Structure

7.1.A Data blocking in meaningful groups and specification of data blocks on import and export lists should be allowed.
P2R.035(#06)

7.1.B The semicolons in the syntax of module_decl and module_spec are inconsistent with their subprogram analogues.
LIR.051

7.1.C There should always be a module body, even if only "null": this simplifies linker and library management.
LIR.310

7.1.D What are the semantics of packaged data in the presence of reentrancy?
LIR.460

7.1.E Module specifications should not be differentiated as Package and Task--this distinction should be made only in the body. Procedures and ent should not be differentiated in the specification part: the linker can take care of any separate compilation problems.
LIR.187 LIR.188

## 7.2 Module Specifications

7.2.A Specifications and program should not be separable: a textual inser mechanism should be used for common declarations.
LIR.247

## 7.3 Module Bodies

7.3.A Direct nesting of modules confuses visibility badly with no increa in functionality.
LIR.068 LIR.198

## 7.4 Private Type Declarations

7.4.A The inability to parameterize private types for defining constraint causes problems with type composition. XX3.0
LIR.008(s3.0) LIR.142

7.4.B Generics are not an adequate way of defining constrainable private types, because each instantiation gives a new type.
LIR.008(s3.2)

7.4.C Restrictions on operations available should apply to private type expressions within the visible part in which the type is defined.
LIR.034

7.4.D Package specification do not need explicit "private parts": the declaration "type x is private" suffices; all else should be in the body.
LIR.236 LIR.583

7.4.E Can literals of restricted type be written using the qualified expression notation outside their modules? Presumably not.
LIR.237

7.4.F Some easier way of inheriting operations for restricted types is desired, eg, 'type t is... inheriting ("<","=",">")'.
LIR.268

7.4.G Objects of restricted private type should be required to be initialized inside their type definitions.
LIR.384

## 7.5 An Illustrative Table Management Package

# 8. VISIBILITY RULES

## 8.1 Scope of Declarations

8.1.A The visibility rules should be simplified. There should be unifor visibility rules regardless of whether a definition is built-in, predefine or user-defined. Use and Restricted should not treat built-in and user-defined definitions differently.
EVR.002(#214) EVR.003(#3.7) EVR.005(#12.0) P2R.014(#03) P2R.026(#13)
P2R.037(#01) P2R.038(#08) P2R.043(#06)

8.1.B There should be partial import for management control, perspicacit and improved optimization.
EVR.003(#2.4) EVR.004(#3) P2R.025(#05) P2R.027(#01) P2R.032(#01)
P2R.036(#04) LIR.049 LIR.138 LIR.259 LIR.570

8.1.C There should be a partial export of record field names; this would allow information hiding in the Parnas sense. Currently, such hiding is nearly impossible.
EVR.003(#2.5) P2R.036(#04)

8.1.D There should be import and export of variables as read-only.
EVR.003(#2.6) P2R.032(#02) P2R.036(#05) P2R.039(#12) LIR.038
LIR.234

8.1.E The scope rules of the language should be modified to closed scope instead of open scope. This would support maintainability.
P2R.036(#02) LIR.056

8.1.F There should be partial export in general.
LIR.038 LIR.138

8.1.G The scope of Accept formal parameters is omitted: it presumably extends from the declaration to the end of the Accept.
LIR.280

8.1.H The visibility mechanism as a whole contributes more to writabilit than readability, contradicting the design goal stated in 1.i.
LIR.570

8.1.I The terminology used in describing scope is confusing. In particular "definitions" should not have "scopes", "declarations" should. Visibility ru should be based on simple principles, listed in the LIR.
LIR.198

## 8.2 Visibility of Identifiers

8.2.A XX4.1 XX5.6 XX5.8
LIR.000

8.2.B The note on redeclaration is apparently extraneous: an inner declaration of an object hides an outer declaration of a homonymous function regardless of types. The restriction on enumeration variables is also questionable.
LIR.550

8.2.C It is not clear what is visible where. What is the relation between scope and visibility? Is an enumeral of anonymous type defined in a record declared in a block visible in the body of the block?
LIR.551

8.2.D The restrictions on redeclaration may be good style, but should not be part of the language. These restrictions will also slow the compiler.
LIR.640

8.2.E Does the restriction on redeclaration apply to the visible part of a module specification, the private part, and its body's outermost declarative part considered as one declaration list?
LIR.217

## 8.3 Restricted Program Units

8.3.A Make Restricted mandatory before a compilation unit.
LIR.023

8.3.B The keyword "restricted" is used, counterintuitively, to specify what is visible, not what is restricted.
LIR.041 LIR.484

8.3.C Clarify what unit names may appear in a visibility list.
LIR.281 LIR.552

8.3.D Use often forces inclusion in the Restricted list. The functions of Restricted and Use should be reorganized to recognize that most items in Use clauses have to be imported.
LIR.303 LIR.446

8.3.E Non-enclosing sub-programs (eg library units) should be allowed in visibility lists. XX10
LIR.435

8.3.F Input_Output seems to appear in a visibility list where it is not visible. Is this because it is a 'library'? XX10.1
LIR.556

8.3.G Importations can be hidden deep within code. There should be some control over this.
LIR.570

8.3.H The importation and visibility restriction functions of the restricted list should be separated. The first name restricts scope; all the others enlarge it. XX10.2
LIR.128 LIR.203 LIR.604 LIR.611 LIR.633

## 8.4 Use Clauses

8.4.A It should be possible for a use clause to refer to a module declared in the same declaration part. (Currently the use clause must come first and there are no forward references.)
P2R.039(#24) LIR.219 LIR.252

8.4.B Any unambiguous reference to identifiers should be permitted, as in PL/I; the Use clause would then be unnecessary.
LIR.229

8.4.C It should be possible to mix Use clauses with declarations freely.
LIR.219 LIR.252

8.4.D The Use clause should be deleted as detrimental to readability: an improved Rename would be a partial replacement. XX8.5
LIR.305

8.4.E Identifiers rendered ambiguous because of the Use clause should be invisible in the scope of the invisibility.
LIR.553

## 8.5 Renaming

8.5.A "Rename" complicates verification and aliasing analysis.
LIR.159

8.5.B Rename should be a statement, not part of a declarative part. XX5.0
LIR.302

## 8.6 Predefined Environment

8.6.A The Environment pragma greatly complicates visibility. Remove it or at least clarify its effect.
LIR.454

# 9. TASKS

9.0.A Tasks intended as parallel threads of control ("processes") and tasks serving to synchronize access to shared data objects ("monitors") are logically distinct (with different implementation strategies as well), so the determination cannot be left to the translator. There are also difficulties with termination, optimization, and recognition with the interface task approach.
LIR.009 LIR.061

9.0.B A task defining a class of sharable objects should be considered as a data type, so as to permit named instances of such objects to be declare to be included as components of other data objects and to be passed as parameters; neither task families nor generics are adequate for this purpose.
LIR.009(s3.1)

9.0.C Capabilities for specifying the low-level implementation of synchronization disciplines should be provided without forcing the user to abandon the basic tasking framework.
LIR.009(s3.2)

9.0.D Allowing unrestricted access to (shared) global variables is not only unreliable and/or inefficient, but also leaves the semantics of basic operations (e.g. assignment) undefined in the presence of concurrent executio
LIR.009(s3.4)

9.0.E The absence of anonymous tasks, tasks as generic parameters and operations applicable to all tasks (e.g. suspend, reschedule, etc) seems to limit capabilities.
LIR.009(s4.10)

9.0.F There should be a way to name task invocations and to control them.
EVR.001(p09) EVR.001(p11) EVR.002(#101) EVR.003(#1.2) EVR.005(#1.4)
EVR.007(s2.6) P2R.013(#06) P2R.014(#09) P2R.018(#03) P2R.018(#08)
P2R.018(#09) P2R.027(#02) P2R.030(#02) P2R.030(#04) P2R.038(#03)
P2R.046(#09) P2R.046(#10) LIR.124

9.0.G It should be possible to achieve efficient and safe sharing of variables. Current mechanisms are either inefficient or unsafe. Perhaps the should be syntactic brackets of critical regions.
EVR.001(p09) EVR.001(p12) EVR.002(#106) EVR.003(#1.4) EVR.005(#1.2)
EVR.006(s4.e) P2R.006(#03) P2R.012(#02) P2R.014(#09) P2R.019(#01)
P2R.019(#13) P2R.022(#04) P2R.028(#05) P2R.033(#03) P2R.036(#01)
P2R.039(#14) P2R.043(#02) P2R.043(#03) P2R.046(#14) LIR.147
LIR.453

9.0.H Test and set and spin-lock, or equivalent functions, are desired.
EVR.002(#106) EVR.006(s4.e) P2R.004(#01) P2R.018(#03) P2R.018(#05)
P2R.022(#02) P2R.025(#04) P2R.027(#04) P2R.036(#03) P2R.046(#12)
P2R.046(#13) OPA.007 LIR.147

9.0.I It should be possible for the user to write and use his own scheduler
EVR.007(s1.2) P2R.003(#01) P2R.018(#02) P2R.018(#06) P2R.025(#04)
P2R.027(#04) P2R.030(#03) P2R.035(#01) LIR.157

9.0.J It is not possible to define a full pledged event abstraction that can guard a select.
P2R.018(#04)

9.0.K It should be possible to handle interrupts efficiently. The interrup information channel is now connected to a task entry. What is required is execution of interrupt handling not under scheduler control.
EVR.001(p12) EVR.003(#2.2) EVR.006(s4.b) EVR.007(s1.2) P2R.001(#01)
P2R.003(#01) P2R.015(#09) P2R.017(#02) P2R.017(#03) P2R.017(#04)
P2R.036(#09) P2R.039(#18) POS.001

9.0.L The interrupt interface is an information channel; what is needed is access to the fact of the interrupt as a control event.
EVR.001(p12) LIR.021(p06) POS.001 LIR.060

9.0.M There is difficulty in linking a family of task activations with a set of interrupts. It should be possible to attach the entry point of a family of tasks onto an arbitrary set of interrupt addresses.
P2R.039(#18)

9.0.N The capability of passing parameters to tasks at activation time should be provided; passing them via entry/accept is subject to waiting.
EVR.005(#1.8)

9.0.O There are significant problems with dynamic tasking on distributed system architectures.
POS.002

9.0.P There is no way of guaranteeing indivisible operations.
LIR.060

9.0.Q All intertask variable access should be forbidden; communication should be accomplished with entry and function calls. This simplifies semantics and extends to distributed architectures.
LIR.254

9.0.R Tasking should be more controllable: specification of preemptivity and resumptivity.
LIR.248

9.0.S Task variables are needed to avoid a problem with the visibility of the index type in task families.
LIR.282

9.0.T On distributed architectures, it should be possible to specify the subsystem on which to run a particular task (as part of Initiate?).
LIR.283

9.0.U Suspend and resume are desired.
LIR.354

9.0.V To avoid buffer tasks, there should be a predefined parameterized type Queue. XX9.12
LIR.373

9.0.W Too many buffer tasks are seen as required. The proposed solution is a mechanism to delegate the completion of an ongoing rendezvous from one task to another, allowing it to be completed in the second task and freeing the first task. Discussion.
LIR.406

9.0.X If task families are to substitute for task variables, there should be some way of finding how many members of the family are active, and some way to get the index of an inactive one.
LIR.407

9.0.Y Some concept of channels is necessary to allow configuration of communication lines among tasks defined in a library at system generation. Otherwise, either the software must be rewritten for each configuration, or installation-dependent communications tasks must be defined.
LIR.590

9.0.Z Too many buffer tasks are required. There should be a variety of entry (with In parameters only) which does not wait for completion of the rendezvous, and queues entry calls.
LIR.591 LIR.610

9.0.ZA Task variables are needed so that a server task can reply to user tasks which are not members of the same task family.
LIR.592

9.0.ZB Although the Rationale emphasizes the distinction, the LRM confutes tasks and threads of control. There should not be such ambiguities. XX11.5
LIR.620

9.0.ZC There should be a unique runtime key for task activations, since ther is no way to guarantee such with current language facilities.
LIR.124

9.0.ZD Low-level synchronization mechanisms should be provided. Channels should be primary, not rendezvous.
LIR.197

## 9.1 Task Declarations and Task Bodies

9.1.A Several problems are raised by procedures in the visible part of a task:
1) When initiating a task a procedure call can only be achieved once the declarations of the corresponding task are elaborated.
2) When a task terminates (normally or abnormally) there may still be ongoing procedure cal...
3) The interaction of procedures and accept statements is complex.
4) Without some precautions procedures permit access to locals of a task and raise issues similar to those of shared variables. XX9.4
OPA.020

## 9.2 Task Hierarchy

9.2.A Tasks should not be nested within procedures or functions; tasks should only be nested within other tasks.
EVR.005(#1.5)

## 9.3 Task Initiation

9.3.A If a procedure in the visible part of a task is called, it may be able to access variables whose declarations have not yet been elaborated. The semantics of the initiate statement are unclear: what assumptions can be made about the state of an initiated task?
LIR.031 OPA.019

9.3.B Initiate should be allowed parameters.
LIR.278 LIR.374

9.3.C It would often be useful to have tasks initialized at elaboration (eg semaphores).
LIR. 279

9.3.D More precise definition of task initiation is needed: two tasks cannot be made active simultaneously, eg, in the presence of interdependent declaration elaborations.
LIR.572

## 9.4 Normal Termination of Tasks

9.4.A There is no facility for synchronous termination of embedded tasks (particularly when such tasks are encapsulated).
LIR.009(s3.6) LIR.284

9.4.B There are problems of logic and implementat.on connected with the exit of scopes containing active tasks.
LIR.022 LIR.311

9.4.C Task termination is ambiguous and in fact may never occur.
EVR.005(#1.6)

9.4.D Suggests that synchronous termination be accomplished by predefining a condition indicating that the task's containing unit wishes to terminate.
LIR.284

## 9.5 Entry Declarations and Accept Statements

9.5.A Separating entry bodies (like procedure bodies) would make tasks easier to read and understand.
LIR.009(s3.8)

9.5.B There is no syntactic difference between an entry and a procedure invocation.
P2R.035(#03) P2R.039(#17)

9.5.C "Then" or "Begin" is preferred to "Do" in the accept statement: Do is an unnecessary extra keyword with incorrect implications.
LIR.332

9.5.D Entry declarations should be restricted to task specifications. An interrupt representation specification may appear in the task body. XX13.5.1
LIR.441

9.5.E Even null bodies of Accepts should have a syntactic terminator.
LIR.442

9.5.F Entries should either be quite distinct from procedures, or unified somehow. It is currently not clear whether many rules apply to entries: overloading, renaming, address specification, placement of declaration and body, generic subprograms, Inline. Do the rules apply differently when an entry is renamed as a subprogram? XX6.6 XX8.5 XX13.5 XX6.1 XX6.2 XX6.4
LIR.444

9.5.G The 'identifier' in entry_declaration is presumably the entry name: what part of it should be used as the identifier?
LIR.554 LIR.571

9.5.H Why is initiation of a task prohibited in an accept body? XX9.3
LIR.572

9.6 Delay Statements

9.6.A The definition should indicate that a delayed task will be queued for scheduling once the designated delay interval has passed.
EVR.002(#207)

9.6.B In addition to delaying for a specific real time interval, there should be a provision for a delay with respect to another task's execution time.
P2R.032(#03)

9.6.C The semantics of the delay statement are context dependent. (See Select statement)
P2R.036(#08) LIR.053

9.6.D There should be some way to wait for a condition (eg resource available) as well as waiting a particular length of time.
LIR.375

9.7 Select Statement

9.7.A There is no provision for selectively waiting for the acceptance of an entry call (eg timed-out calls).
LIR.009(s3.10) LIR.359 LIR.360 LIR.452

9.7.B The language should guarantee fairness in the select statement: it should not be possible for a queued entry call to be permanently blocked by subsequent entry calls sharing the same select statement.
EVR.002(#104) P2R.015(#07) LIR.189

9.7.C The language should either (a) restrict the variables that can appear in the guards of a select statement to those that cannot change while awaiting the entry call, or (b) guarantee reevaluation before selection of any alternative with a guard that may have changed.
EVR.002(#209)

9.7.D There should be conditional entries as well as conditional accepts.
EVR.002(#216) P2R.039(#15) LIR.002

9.7.E The language should guarantee that a waiting entry call will always be selected in preference to a delay.
EVR.002(#104)

9.7.F There is a need for a select guard that is true only if all others are false. When Others should be added to the select statement.
LIR.026

9.7.G Select should clearly be specified to act non-deterministically, as any programs depending on fairness will likely be implementation-dependent.
LIR.089

9.7.H The select statement is too complicated; a lower-level mechanism is preferred.
[illegible].231

9.7.[illegible] The present rendezvous concept is good: timed-out entries and suspend/resume would hurt effectiveness and uniformity.
LIR.255

9.7.J It should be possible to have exception handlers with scope co-extensive with one select alternative to catch propagated exceptions.
LIR.285

9.7.K There should be entry call timeouts. The details of a correct implementation are discussed.
LIR.319

9.7.L Entry calls should be allowed in Select as are Accept's in order to express a nondeterministic choice between consumption and production. XX9.0
LIR.397

## 9.8 Task Priorities

9.8.A The language should guarantee that priorities will be rigidly enforced during scheduling.
EVR.002(#210)

9.8.B Task priority should be assignable at initiation time; queue reordering should also be possible.
EVR.001(p11) EVR.005(#1.3) P2R.015(#10) P2R.018(#07) P2R.035(#01)

9.8.C Scheduling is vague and too restrictive. Implementation dependency i encouraged by not specifying that scheduling is non-deterministic.
LIR.060 LIR.080

9.8.D The semantics of priorities are unclear, especially in the presence o monitor-type tasks.
LIR.081 LIR.083

9.8.E Interrupt handlers should have priorities but should not be subject t scheduling.
LIR.146

9.8.F There should be some mechanism for specifying the mapping between Ada's priority and tasking constructs and the machine's.
LIR.298

9.8.G Preemptive scheduling should be possible in any implementation.
LIR.352

9.8.H Tasks should be able to set their children's priorities, but why should they be able to set their own?
LIR.605

## 9.9 Task and Entry Attributes

## 9.10 Abort Statements

9.10.A Both the ABORT statement and raising FAILURE are extremely dangerous. In particular asynchronous termination of a rendezvous causes severe problems in maintaining the consistency of internal data.
LIR.009(s3.7)

9.10.B The Abort statement is unnecessary.
LIR.242

9.10.C Abort should not take a name, but a variable as an argument; it should be possible to abort oneself and one's parent without knowing their names.
LIR.363

9.10.D Tasking exceptions should be described in the tasking section, not the exception section, as other exceptions are described with their constructs, or at least cross-referenced XX11.4
LIR.555 LIR.558

9.10.E The semantics of Abort should be simply those of raising Failure but ignoring exception handlers. XX11.5
LIR.621

## 9.11 Signals and Semaphores

9.11.A The built-in (generic) tasks SIGNAL and SEMAPHORE are non-traditional, difficult to use and unnecessary.
LIR.009(s3.9)

9.11.B Making semaphores into tasks precludes their incorporation into data objects.
LIR.060

9.11.C What is the meaning of the priority of a semaphore?
LIR.083

9.11.D What is the meaning of priority to interrupts? What happens to priority when a high-priority task needs the services of a low-priority task? Discussion.
LIR.427

## 9.12 Example of Tasking

## 10. PROGRAM STRUCTURE AND COMPILATION ISSUES

10.0.A "Independent" compilation for units communicating only through parameter lists and not global environment should be defined for external units, such as dynamically loadable units and foreign language units.
LIR.130

10.0.B The separate compilation feature is unnecessary. Source inclusion or support utilities should deal with separate compilation.
LIR.144 LIR.584

10.0.C What units are actually loaded? What is the minimum one can expect of the library and loader in terms of not loading unused units? What is the unit of loading? Subprograms, modules, compilation units?
LIR.321

## 10.1 Compilation Units

10.1.A The present system has both too many surprising consequences, and precludes too many useful optimizations. A simpler system would be quite adequate.
EVR.003(#1.3) P2R.005(#05) P2R.005(#06) P2R.006(#05) P2R.030(#01)
P2R.037(#07) DCR.003 LIR.128

10.1.B The language allows separate compilation of nested entities (modules, procedures, tasks); for the programmer, it will be very difficult to know the environment of such a separately compiled entity.
P2R.014(#07)

10.1.C The physical interface contains too much information. In particular the private part should specify the representation of any visible private types.
DCR.003

10.1.D Visibility restrictions are overly restricted in separate compilations.
LIR.139

10.1.E Stubs for subunits can sometimes be ambiguous. XX10.2
LIR.118

10.1.F The system of separate compilation incorporates too much information about what units will be compiled together into the text of the program.
LIR.120

10.1.G It can be impossible to distinguish identically named subunits without blocking their vision of a common enclosing unit.
LIR.140

10.1.H What exactly IS a program library?
LIR.556

10.1.I Syntax of compilation_unit should presumably be [visibility_restriction [Separate]] unit_body (cf. 10-5 line 1).
LIR.573

10.1.J There are cases where separate compilation seems unnecessarily illegal. (Example)
LIR.622

## 10.2 Subunits of Compilation Units

10.2.A Selected components of compilation units should be specifiable in Restricted statements, eg if Main has subunit A, permit Restricted(Main.A).
OPA.016

10.2.B The enclosing unit of a subunit should be explicitly specified in the compilation unit header--it is otherwise ambiguous for reader and compiler.
XX8.3
LIR.128 LIR.241 LIR.436 LIR.445 LIR.574
LIR.597 LIR.604 LIR.609 LIR.611

10.2.C Separately compiled overloaded subprograms within the same enclosing unit should not be allowed. XX10.1
LIR.304

10.2.D Separation of bodies from specification should not be restricted to the outermost scope. XX7.0
LIR.449

10.2.E What IS a subunit?
LIR.574

## 10.3 Order of Compilation

10.3.A The strategy for ordering separate compilations does not work in the presence of separate generic units, inline subprograms, representation specifications, and certain requirements concerning calls to procedures with side effects.
P2R.005(#01) LIR.004 COM.002 DCR.003

## 10.4 Program Library

10.4.A The program library file should not be updated by all compilations as this may compromise its integrity.
LIR.120

## 10.5 Elaboration of Compilation Units

## 10.6 Program Optimization

10.6.A It is unclear that some optimizations concerning functions and variables with "abnormal" behavior can be performed by the translator.
POS.003(p01) DCR.003

10.6.B There should be explicit conditional compilation, using pragmas.
LIR.036

10.6.C The programmer should be able to ask for many implementation choices and optimizations explicitly: omission of GC; static allocation; use of global flow analysis; suppression of runtime checks. XX2.7
LIR.410

10.6.D Discussion of optimization should be left to the Rationale.
LIR.575

# 11. EXCEPTIONS

11.0.A Manual does not make clear what exception gets raised for some cases of constraint violation.
LIR.008(s3.3)

11.0.B Underflow should not be an exception.
EVR.005(#2.1) LIR.366

11.0.C There is no way to handle user exceptions propagated beyond the scope of their definition ("Unhandled" exception). Should Others handle them?
LIR.048 LIR.559

11.0.D There should be no exception facility as it introduces too great an overhead.
LIR.244

11.Ø.E What happens when an exception propagates beyond its scope? Making exception definitions global is suggested.
LIR.24Ø

11.Ø.F Explicitly raised exceptions should leave variables' values well defined.
LIR.367

## 11.1 Exception Declarations

11.1.A No_value_error is ill founded.
OPA.Ø15

11.1.B No_value_error is too expensive to implement.
LIR.Ø34

11.1.C No_value_error from a function should be raised in the caller's environment.
LIR.Ø88

11.1.D It is not clear when and where which predefined exceptions are raised.
LIR.557

## 11.2 Exception Handlers

11.2.A The language should guarantee that actual Out parameters will not be assigned if the routine is exited abnormally (i.e., by exception). XX6.3
EVR.ØØ2(#1Ø2) P2R.Ø43(#13)

11.2.B There is a need for exceptions that will not be handled by When Others.
LIR.Ø16

11.2.C Exception handlers should have access to the environment at the point of an exception for testing and debugging.
LIR.Ø57

11.2.D It should not be possible to access/reference unelaborated or incompletely elaborated declarations from within an exception handler.
EVR.ØØ2(#213) DCR.ØØ5 OPA.Ø12

11.2.E It should be possible to specify explicitly in the exception handler whether terminative or resumptive semantics apply to the particular handler.
EVR.ØØ5(#15.Ø)

11.2.F There should be some way to identify an exception caught by "Others" for debugging and error messages.
LIR.184 LIR.399

11.2.G It should be possible to return and continue after an exception.
LIR.465

11.2.H Exceptions should pass parameters, eg Assert(35) x>0;. XX5.9
LIR.466

11.2.I If an exception propagates out of a scope and back in, is it handled by the named handler or Others? Presumably the named handler. Example given.
LIR.526

## 11.3 Raise Statements

## 11.4 Exceptions Raised During Tasking

11.4.A Propagation of Tasking_error compounds the problems of asynchronous termination, especially with regard to procedures in the visible part of task:
LIR.009(s3.7)

## 11.5 Raising an Exception in Another Task

11.5.A The asynchronous exception Tasking_error may be raised on the accepting task during a rendezvous. This disruption causes problems for tasks that require indivisible updates of their data structures in order to maintain consistency.
LIR.003

11.5.B The semantics of raising the failure exception in another task are unclear and sometimes counter-intuitive.
LIR.119

11.5.C The semantics of the Failure exception are complicated in the presenc of multiple threads of control corresponding to a task: exception propagation is dangerous. The example of Rationale 12.4.1 is flawed and demonstrates the dangers. The Failure exception should only take effect when the thread of control executes inside the task or when it returns to it. The question remains as to whether a thread of control waiting in an entry is executing inside or outside the task. XX9.0
LIR.620

## 11.6 Suppressing Exceptions

11.6.A The language should restrict the consequences when a suppressed exception occurs.
EVR.002(#212)

11.6.B The semantics of suppressing the ASSERT_ERROR exception should be specified.
LIR.019

# 12. GENERIC PROGRAM UNITS

12.0.A  There should be task generic parameters. For example, task entries should be allowed as generic parameters.
P2R.039(#08)  LIR.015

12.0.B  There is a need for a specification and assertion language for generics. It is not clear at this time what the problems will be. There are strong reservations about a language that allows thing to look the same but have different meanings.
P2R.043(#05)

12.0.C  The generic facility does not provide true parameterized types nor can it express type interrelations and properties (eg T is a discrete type). How can a record field be guaranteed to exist?: the type cannot be restricted nor can the field name be a generic parameter. Consider also the interaction with separate compilation.
LIR.070  LIR.196  LIR.388

12.0.D  Overloaded generic subprograms cannot be disambiguated: prohibit them. XX4.1.2 XX6.6
LIR.527

12.0.E  A generic subprogram can have the same signature as a non-generic subprogram but be distinguishable. Can one overload the other? XX6.6
LIR.528

12.0.F  Overloading of generic subprograms by generic clause is not allowed. But it could be. XX6.6
LIR.529

12.0.G  Generic functional arguments may require implementation techniques identical to those required for functional arguments. XX6.0
LIR.623

12.0.H  Generic parameters should be allowed to be generic subprograms.
LIR.623

12.0.I  All compilation and error-checking of generic subprograms should occur at instantiation time.
LIR.207

12.0.J  A general macro facility is desired.
LIR.211

## 12.1 Generic Clauses

12.1.A  The concept of a "designator" as an "attribute of a type" is vague and confusing.
LIR.136

12.1.B  The syntax of Subprogram_specification as a Generic_Parameter allows a Generic_clause. Forbid this either in the syntax or the semantics.
LIR.215  LIR.287  LIR.473

12.1.C What are the exact semantics of generic Out and In out parameters?: suggests forbidding them.
LIR.288

12.1.D The attribute 'Size of a generic type parameter should be available in the generic body.
LIR.290

12.1.E Allow entries as generic parameters.
LIR.291

12.1.F Generic units and entries should be allowed as generic parameters. Visibility for the generic body should be defined by the point of instantiation. Extensive discussion of interdependent generic tasks.
LIR.398

12.1.G Record component names should be allowed as generic parameters.
LIR.419 LIR.474

12.1.H There should be a way to indicate that the parameter declarations among the generic parameters are commutative. (??)
LIR.459

12.1.I Exceptions and packages should be allowed as generic parameters.
LIR.474

## 12.2 Generic Instantiation

12.2.A Implicit instantiation of generic subprograms is needed. Implicit instantiation of other generic definitions is not needed.
EVR.002(#301) EVR.003(#3.4)

12.2.B Ada relies heavily on generics. In particular, they are the means for realizing parameterized types. Procedures and functions that take parameterized types must also be generic. Thus the compiler must be able to recognize when generic procedure instantiations may share code. Can it?
EVR.003(P02)

12.2.D There is a problem with instantiating a generic with a type that is an unconstrained array type.
LIR.028

12.2.E "New name" should presumably read "new designator".
LIR.034

12.2.F The syntax of generic_association should allow "designator Is", but formal_parameter restricts it to identifiers.
LIR.576

12.2.G Generic parameters used in static-evaluation contexts in the body should not be required to be static.
LIR.289

12.2.H The syntax of generic definition and instantiation violates the principle that specifications should parallel uses. Syntax suggested.
LIR.191

## 12.3 Example of a Generic Package

## 13. REPRESENTATION SPECIFICATIONS AND IMPLEMENTATION DEPENDENT FEATURES

13.Ø.A There should be an escape mechanism that will permit the user to specify the storage management algorithm for pointer/heap storage. XX3.8
EVR.ØØ2(#3Ø4)

13.Ø.B The programmer might be restrained if acceptable space and access efficiency were needed, for example by prohibiting arrays with dynamic bounds or minimizing shared variables. XX3.6
EVR.ØØ1(p13)

13.Ø.C Programs using pointers cannot be guaranteed to be free of garbage collector overhead. XX3.8
P2R.Ø22(#Ø1) LIR.243 LIR.25Ø

13.Ø.D Non-stack storage allocation is needed to implement parallelism and dynamic storage: where, then, is local storage for a task allocated? In a single address space model, the new process must be allocated storage of some fixed size at initiation. Fixup action must be taken on overflow, or a probe is needed before growing the stack. Both are too inefficient for embedded computer applications. XX9.Ø
P2R.Ø27(#Ø3)

13.Ø.E There are no facilities for program overlays.
LIR.ØØ1

13.Ø.F Make it clear that a length specification for a collection inhibits garbage collection (and hence permits user definition of Allocate and Free as shown in Washington April meeting). XX3.8
OPA.ØØ2

13.Ø.G Representation change is prohibited for derived record and enumeration types with user attributes but not for similar array types.
LIR.1ØØ

13.Ø.H Representations should be a part of type declarations (not separate) and have a more compact form. XX6.1
LIR.157 LIR.249 LIR.276

13.Ø.I Is bit Ø the low-order or the high-order bit? XXA
LIR.157 LIR.351

13.Ø.J The For/Use construct is overloaded.
LIR.247 LIR.249

13.Ø.K More of the attributes of a type should be incorporated into declarations rather than representations. XX3.Ø
LIR.249

13.Ø.L There should be a representation specification for fixed-point numbers defining the value of the most significant digit and precise layout. (Some suggest making this part of the type definition itself.) XX3.5.5
LIR.3Ø6 LIR.35Ø LIR.391 LIR.412 LIR.413
LIR.423

13.Ø.M Lack of inheritance of representations by derived types seen as possibly burdensome.
LIR.462

13.Ø.N Some sort of representation specification is desired for arrays.
LIR.463

## 13.1 Packing Specifications

## 13.2 Length Specifications

13.2.A The length specification for an access type should not be required to be static.
LIR.429

13.2.B What is the type of the static expression?
LIR.577

## 13.3 Enumeration Type Representations

13.3.A It should be possible to specify contiguous representations of runs of enumerals without writing them all ~ut: suggests that unmentioned enumerals received the representation of the preceding enumeral plus one.
LIR.286

13.3.B The current syntax for enumeral representation is not transparent in meaning. The requirement that a representation aggregate be named when there is but a single enumeral is a disturbing irregularity in the syntax. Perhaps the syntax of aggregates is to blame. XX3.6.2 XX4.6
LIR.531

## 13.4 Record Type Representations

13.4.A The syntax is considered clumsy and redundant.
LIR.Ø64

13.4.B Alignment clause cannot specify, e.g., 1 mod 8, but only Ø mod 8.
LIR.Ø93

13.4.C "At" is a poor keyword here.
LIR.093

## 13.5 Address Specifications

13.5.A Can two variables be given the same address? Clarify manual.
LIR.352

13.5.B For...Use at is too static and one-memory oriented. XXA
LIR.396

### 13.5.1 Interrupts

13.5.1.A Interrupts should not be queued.
LIR.146

13.5.1.B Interrupts should be "masked out" inside their own handlers.
LIR.146

13.5.1.C What happens when two entries are attached to the same interrupt?
LIR.239

13.5.1.D How does one guarantee immediate servicing of interrupts?
LIR.239

13.5.1.E There is apparently a problem in implementing Ada interrupts on the UYK-20.
LIR.180

## 13.6 Change of Representations

## 13.7 Configuration and Machine Dependent Constants

13.7.A A floating-point real-time clock is impractical; moreover, the clock should measure time of day rather than time since initiation. There is an ISO standard on date and time which should be consulted.
EVR.002(#208) P2R.002(#03) P2R.025(#06) DCR.005 LIR.301
LIR.393 LIR.422

13.7.B The notion of task cumulative processing time ('Clock) forces inefficiencies; System'Clock, however, is useful.
OPA.009 DCR.006

13.7.C There should be an implementation-independent fixed-point clock.
LIR.415

13.7.D What manner of beast are System and Option? They are predefined names, but not reserved words or package names. Are they object names? LIR suggests they be predefined internal packages; their attributes would thus be selected by dot notation. XXA XXC
LIR.471 LIR.492

## 13.8 Machine Code Insertions

13.8.A There should not be any special mechanism unique to machine and/or assembly language.
EVR.002(#211) P2R.014(#06) P2R.022(#06) P2R.028(#02) P2R.042(#01)

13.8.B This section is too vague.
LIR.034

13.8.C Assembler insertions should have conventional syntax.
LIR.424

## 13.9 Interface to Other Languages

13.9.A The same mechanism should be used for assembly language and machine code interfaces as is used for interfacing other programming languages.
EVR.002(#211) P2R.008(#01) P2R.044(#07)

13.9.B It is not specified how to invoke a procedure from another language.
P2R.037(#06)

13.9.C There are some problems with the foreign code interface for Fortran, e.g. matrix representation, slice parameters, functions as parameters, and variable length parameter lists.
LIR.014

13.9.D How are Ada programs called by programs in other languages?
LIR.157

13.9.E It is suggested that the Ada interface conventions for a given machine become the standard conventions for the other languages on that machine. All machine and OS formats should be defined as Ada data structures. Ada should be the standard intermediate language.
LIR.296

13.9.F There should be a standard (unsafe) way of building an Ada array from a block of storage passed into an Ada routine from a non-Ada routine.
LIR.387

13.9.G More support is needed for interface to other languages. Perhaps machine-dependent code should be isolated in a special module as proposed in Euclid.
LIR.578

## 13.10 Unsafe Type Conversion

13.10.A Reinterpreting the type of an object without real conversion is desired.
LIR.062

13.10.B The name Unsafe_programming is too strong.
LIR.309 LIR.451

13.1Ø.C The applicability of Unsafe_conversion to I/O is not made clear.
LIR.349

13.1Ø.D What exactly does Unsafe_Conversion do? When it is imposing a type on previously untyped data, it should check for Range_Error.
LIR.451

## 14. INPUT-OUTPUT

14.Ø.A For an I/O handler, multiple instantiations and numerous names are required to use files of all types. This is extremely cumbersome if many types are present.
P2R.Ø39(#Ø7)

14.Ø.B The untyped binary I/O on which typed binary I/O is built should be visible to users and standard across implementations, since it must of necessity exist under Input_Output.
LIR.1Ø7

14.Ø.C The I/O model is at too high a level, too sequentially oriented and overly attached to the idea of one data type per file.
LIR.Ø46

14.Ø.D There should be a high-level model of real-time data stream I/O.
LIR.327

14.Ø.E The I/O package should not be addressed in the LRM.
LIR.371

14.Ø.F There should be some standard way to time-out from I/O.
LIR.376

### 14.1 General User Level Input-Output

14.1.A The departure from conventional I/O techniques may lead to nonstandard I/O techniques between similar systems. In particular "conventional" read, write, and format statements are missing.
EVR.ØØ5(#5.Ø) LIR.238

14.1.B Objects of mixed types should be allowed to coexist on files.
LIR.1Ø7 LIR.327

14.1.C A standard package implementing Fortran-like formats should be defined.
LIR.299

#### 14.1.1 Files

14.1.1.A There should be a function for determining whether a filename corresponds to an existing and accessible file.
LIR.421

14.1.1.B Renaming of files is missing. There is no way provided to write and end of file mark or change the valid length of a file. It should not be necessary to open a file to delete it.
LIR.206

### 14.1.2 File Processing

14.1.2.A The names Read and Write should be exchanged with Get and Put for naturalness and Pascal compatibility. XX14.3
LIR.372 LIR.395

14.1.2.B End of file should be a predicate, not an exception. XXC
LIR.430

14.1.2.C Read without advancing the file pointer is missing.
LIR.206

### 14.2 Specification of the Package INPUT OUTPUT

14.2.A It is worthwhile to treat I/O devices as uniformly as possible. This raises many subtleties of treating I/O devices as files.
LIR.007 LIR.013

14.2.B Array I/O should be defined.
LIR.106 LIR.327

14.2.C Some method of forcing buffers out (i.e. draining, flushing) should be defined.
LIR.109

14.2.D The exceptions in different instantiations of the generic package Input_Output cannot be distinguished. The package should have functions which return information as to what caused the exception. XX11.1 XX12.2
LIR.475

14.2.E Delete is missing (cf. 14.1.1).
LIR.217

### 14.3 Text Input-Output

14.3.A Imbedded carriage control characters would be nonstandard across systems, and cause confusion in Ada I/O; thus, a machine independent mechanism should exist for carriage control.
LIR.108

14.3.B Can Ada I/O support a text editor efficiently? More support is needed for terminal text I/O.
LIR.579

14.3.C There should be some standard text I/O for structured types (records).
LIR.579

14.3.D Simulated I/O (Fortran Encode/Decode) into strings is desired: Get and Put should be overloaded on the File parameter.
LIR.182

14.3.E Input and output of text lines are desired.
LIR.183

<u>14.3.1 Standard Input and Output Files</u>

<u>14.3.2 Layout</u>

14.3.2.A "Tab" is used for "HT" despite Appendix C. XXC
LIR.105

14.3.2.B The effect of Tab is nonstandard (should be next multiple of eight plus one).
LIR.165 LIR.580

14.3.2.C Do control characters actually appear in files, or do they just indicate effects? In particular, the distinction between Newline and CR & LF is unclear. If the characters appear in files, how does the file system work on record-oriented systems?
LIR.098

14.3.2.D Tab stops should be user-specifiable. Outputting a Tab should insert the appropriate number of spaces.
LIR.181

<u>14.3.3 Input-Output of Characters and Strings</u>

14.3.3.A LRM confuses issue of quotes within strings.
LIR.103

<u>14.3.4 Input-Output for Other Types</u>

<u>14.3.5 Input-Output for Numeric Types</u>

14.3.5.A Real number input syntax should be more liberal.
LIR.110

14.3.5.A The Get function rounds inputs to Float'Digits rather than to the full precision of the object gotten.
LIR.105

14.3.5.B Do positive numbers print with initial "+", blank, or digit?
LIR.099

<u>14.3.6 Input-Output for Boolean</u>

## 14.3.7 Input-Output for Enumeration Types

14.3.7.A The case of enumeration types' output should be specifiable.
LIR.099

## 14.4 Specification of the Package TEXT IO

14.4.A An expression on 14-11 is missing needed qualifications.
LIR.105

14.4.B Tab stops are poorly defined and probably non-standard.
LIR.125

## 14.5 Examp!- of Text Input-Output

## 14.6 Low Level Input-Output

## A Predefined Language Attributes

A.A The word "machine" is used where "implementation" is meant.
LIR.105

A.B The definition of ASCII as an enumeration type is circular.
LIR.063

A.C T'Rep is incompatible with Put in that it does not take width and fra
arguments.
LIR.099

A.D There should be a useful 'Address for data not word-aligned.
LIR.260 LIR.381

A.E 'Size should apply to program units. This is useful for memory allocation, swap control, and overlays.
LIR.263

A.F The 'Address for non-contiguous packages (eg pure and impure parts) is not well defined and not entirely useful.
LIR.264

A.G What are the types of 'Delta, 'Small, System'Min_Int, and System'Max_Int? XX3.5.5 XX13.7
LIR.330 LIR.470 LIR.492

A.H What is the meaning of 'Address in segmented and multiprocessor architectures?
LIR.331 LIR.396

A.I The attributes 'Size, etc. should be defined in terms of digits, not bits, and the base of the machine should be a system attribute. 'Small and 'Large should also be defined in a radix-independent way. XX13.2
LIR.331

A.J Bit positions should be 1-origin, not 0-origin. XX13.4
LIR.331

A.K Page size should be a system attribute.
LIR.370

A.L 'Bits and 'Radix should only be defined for floating point types. 'Bits should be renamed 'Mantissa; abolish 'La and 'Small. XX3.5.5
LIR.379

A.M Abolish 'Access_Size--the meaning of 'Size should be uniform. Introduce, eg, 'Denoted_Size if desired.
LIR.380

A.N 'Size should be clearly defined to be the maximum size of an object of the type. (Consider records with variants, etc.)
LIR.381

A.O There should be a predefined attribute of any type converting to a fixed length array of character.
LIR.420

A.P The identifier Priority is both an attribute name and a type. This is legal but confusing. Change the type to Task_Priority.
LIR.472

A.Q Why are there no System'Min_Float and 'Max_Float? XX3.5.5
LIR.605

A.R 'Rep should allow more than three digits of exponent when necessary.
LIR.179

B Predefined Language Pragmas

B.A When Pragma Optimize is not used, are no optimizations performed? How does one optimize only part of a module? What is the default state of Optimize?
LIR.533

B.B Define the effect of the pragmas Page, List, and Include on listings more precisely.
LIR.534 LIR.535 LIR.536

C Predefined Language Environment

C.A Attributes of record components should not be applicable to discriminants, since they need not be present.
LIR.008(s4.0)

C.B The characters %, ", |, and # lack enumeration literals.
LIR.103

C.C There should be a predefined type Time_interval distinct from Time with only appropriate operations on each (eg no Time+Time).
LIR.301

C.D Exponentiation and Mod should be defined for more combinations of types to encourage uniformity. XX3.5
LIR.333

C.E Common mathematical functions (square root, sine, etc.) should be predefined.
LIR.392

D Glossary

D.A Suggests some additions in the area of access variables. XX3.8 XX4.7
LIR.477

E Syntax Summary

E.A There are too many 'indeterminisms' in the current syntax.
LIR.293

E.B Syntax summary is incomplete.
LIR.295

Index

Index.A The index is inadequate.
LIR.115

Index.B The index omits Program Units (1.7) and Declare (6.7) and indexes Initial value incorrectly.
LIR.464

Index.C Some grammar non-terminals are not in the index. The standard identifiers (First, Environment, Integer) should certainly be included also.
LIR.537

Z General questions of syntactic style

Z.A The syntax has too many noise words and too much redundancy in general. On the other hand, some keywords are overloaded with quite distinct meanings in different contexts, e.g. else, exception, for, new others, restricted, range, is....
LIR.041 LIR.597 LIR.601 LIR.603 LIR.607
LIR.629

Z.B Syntax is too verbose and keywords are too long.
LIR.047 LIR.078

Z.C The permissible nesting of subprograms, generics, and modules is vagu
LIR.052

Z.D Every type of "end" should be qualified by the block name or type. A least, the meaning and optionality of identifiers after End's should be uniform. Currently they are not: End name for task bodies, but End Case for cases, and just End for Begins. XX5.6 XX6.4 XX7.1
LIR.094 LIR.217 LIR.583

Z.E The syntax is far too permissive: semantic distinctions are blurred a ambiguities often engendered. XX7.1 XX5.5 XX2.6.2 XX3.5.5 XX3.6 XX10.2 XX5.2 XX6.2
LIR.162 LIR.628

Z.F Semicolons should be used for statement termination only; commas should be used to separate items in a series (eg parameters).
LIR.205 LIR.250

Z.G Semicolon should be a statement separator, not a terminator: consider especially, the semicolons after end's of different kinds.
LIR.443

Z.H In several places, the syntax [name.]designator is used where it seem new nonterminal, subprogram_designation, defined as name | [name.]character_string, would be more appropriate. cf. renaming_declaration generic_parameter, and generic_association. XX4.1 XX8.5 XX12.1 XX12.2
LIR.618

Z.I The "upper-level" syntax (unit headers) is ad hoc and poorly structur For instance, generic names appear at the wrong place. The whole upper level should be redesigned starting from the abstract syntax. XX12.1 XX8.3
LIR.633

Z.J Empty fields (not "null;") are allowed in some surprising places. Usually, the metasyntax {...} is used where "one or more" would seem more appropriate. XX3.7 XX5.0 XX5.5 XX6.7 XX9.7
LIR.202 LIR.218

## APPENDIX C: Documents

A significant number of "questions of interpretation" about Ada arose (primarily from implementors). These were questions about unclear points in Preliminary Ada, and were not intended to bring up questions of design. The objective was to answer questions of immediate importance to implementors and users in general.

These questions were submitted to the language design team and were answered in November 1979. It was planned that the asking and answering of questions would be an ongoing process, but few questions came up later.

It should be recalled that these questions and answers refer to Preliminary Ada only and may be irrelevant or incorrect with respect to Final Ada.

The questions and their respective answers are found in the file Questions.Answered. Both question and answer are preceded by a section number.

## APPENDIX D: Documents Maintained By Intermetrics For Ada Test and Evaluation

Nine types of documents have been archived during the Test and Evaluation process. Each has its own log and set of files containing the text of those individual documents which have been received in machine form. Logging conventions and file naming conventions are consistent over types. Each log file is named XXX.LOG where "XXX" is a 3 letter code for the type. The text of documents is stored in files named XXX.001, XXX.002, XXX.003, etc. Again, "XXX" is the 3 letter code corresponding to a particular type; the sequence number provides a uniform and unique reference to such documents.

Log files contain one line of summary information about each document of the type they log: a sequence number, length, source, and subject. All log files contain at least a sequence number and source. Most log files also record the length of the documents. An entry of 0 in this field indicates that the document is not available on-line.

The source is the person(s) or organization submitting the document. If an institutional affiliation is known it is put in parentheses after the author's name. Some documents have been submitted without an indication of source; others have only an institution's name. Note that should a document have as its source the name of an institution, its contents does not necessarily reflect the official position of that organization.

The subject field is an attempt to encapsulate in a very small space the most informative title or summary for a given comment. If a subject reaches a conclusion, that conclusion is briefly indicated; if a comment indicates a problem in a certain area, that problem is made as explicit as possible in the small field available.

An entry, then, looks like this:

| Doc # | Length | Source | Subject |
|---|---|---|---|
| 001 | 32 pgs. | J. Jones (X Corp.) | Parameterized Types Needed |

Text files are simply on-line copies of the original documents. If the document was mailed to us via the Arpanet the block header is retained for reference.

Document Types Logged

1. Phase Two Reviews (P2R)

These documents are not actually available on line, but are nonetheless logged in order to establish sequence numbers by which they may be uniquely identified.

```
        Log file : P2R.LOG
        Text files: P2R.001, P2R.002, P2R.003, ...
```

2. Evaluation Reports (EVR)

Evaluation reports include extracts of selected Phase Two Reviews as well as portions of documents previously designated as "All Others." Most documents of this type are available as on-line text. The set of EVR's was essentially closed rather early. Newer documents are generally archived as one of the types described below.

```
        Log file : EVR.LOG
        Text files: EVR.001, EVR.002, EVR.003, ...
```

3. Language Issue Reports (LIR)

Language Issue Reports are received by Intermetrics from the community at large.They must generally be in the format specified by HOLWG in order to be classified as LIR's. About 80% of these were submitted in machine form (either over the Arpanet or by transportable media) and are therefore on-line.

```
        Log file : LIR.LOG
        Text files: LIR.001, LIR.002, LIR.003, ...
```

4. Comments (COM)

There are Comments of many types: comments on LIR's, short points, questions, dialogs on certain issues etc. In general, whereas LIRs are intended to address one topic each, comments may address a range of topics within one document. This often leads to more general comments, or to comments related to an overall analysis of suitability of Ada to particular areas. The titles of comments are therefore less specific than those of LIR's. Comments are heterogeneoty in form, content, and topicality. MSG headers are retained for each comment in order to preserve their history.

```
        Log file : COM.LOG
        Text files: COM.001, COM.002, COM.003, ...
```

5. Position Papers (POS)

Various individuals or groups were expected to submit position papers. These were to be in-depth treatments of specific problems or related issues. This has not proven to be a popular type of submission.

```
        Log file : POS.LOG
        Text files: POS.001, POS.002, POS.003, ...
```

6. Draft Change Requests (DCR)

As described in section 2, a formal procedure was established whereby drafts of proposals for language changes written by Intermetrics and discussed by the Distinguished Reviewers would be submitted for review and action by HOLWG.

Those documents, called Draft Change Requests, were generated until the procedure was discontinued. They were and still are considered drafts; their presence in the log does not indicate their evolutionary disposition.

During the review process, certain Draft Change Requests (DCR's) underwent revisions, reflecting the comments and opinions expressed during Reviewers' Meetings. A version number is thus appended to the name of all such files. The log indicates the most current version.

Log file : DCR.LOG
Text files: DCR.001, DCR.002, DCR.003, ...

7. Language Design Notes (LDN)

These are proposals from CII-HB for changes to the language. They should not be construed as a commitment by the language design team to implement the changes.

Log file : LDN.LOG
Text files: LDN.001, LDN.002, LDN.003, ...

8. Official Problem Acknowledgments (OPA)

These are statements about language problems officially recognized by the CII-HB Design Team.

Log File : OPA.LOG
Text Files: OPA.001, OPA.002, OPA.003, ...

The full logs appear in Appendix F.

## APPENDIX D: Accessing The Archive

The files described have been made available for public inspection and use over the Arpanet at the University for Southern California's Information Sciences Institute machine "E", Arpanet address USC-ISIE.

The ISIE machine is a Tops-20 system. It accepts File Transfer Protocol (FTP) and Remote Login (Telnet) connections across the Arpanet. Files relating to Ada test and evaluation are found on the disk directory <TNE-Archive>.All comments on the Ada language which were submitted via Arpanet mail are stored here, or archived on the ISI magnetic tape backup.

During the Test and Evaluation period, all the files described have been made available on-line for public access by the community. The files will continue to be accessible on-line indefinitely, although requests may have to be entered to the ISI system for retrieval of files from tertiary (magnetic tape) storage.

An anonymous account is available for File Transfer Protocol connections.

The following dialogue is an example of a typical FTP User program:

```
 Ftp                               --invoke Ftp
 ...machine response
Conn ISIE                          --connect to ISIE
 ...machine response
 Login Anonymous your_name         --login to ISIE
 ...machine response
Get <TNE-ARCHIVE>file_name         --do file transfer
 ...
 Disc                              --disconnect from ISIE
 Quit                              --return from FTP
```

## APPENDIX E: TER Code Breakdown

Many participants in the T&E analysis submitted algorithms written in a language normally used by the participant, and often included a version of that algorithm written in Ada; this offered not only an excellent means of comparison between the two languages, but also helped illustrate Ada in an applications context.

The list below indicates which of the submitted contained code samplings.

| TER # | Original Code | Ada Code |
|---|---|---|
| 1 | - | A |
| 2 | - | A |
| 3 | - | - |
| 4 | - | A |
| 5 | - | A |
| 6 | - | A |
| 7 | - | - |
| 8 | - | A |
| 9 | - | A |
| 10 | - | A |
| 11 | - | A |
| 12 | - | A |
| 13 | E, D | C, B, A |
| 14 | - | B, A |
| 15 | - | A |
| 16 | A | - |

| | | |
|---|---|---|
| 17 | A | B |
| 18 | C, B | (A) |
| 19 | - | A |
| 20 | - | A |
| 21 | - | A |
| 22 | - | - |
| 23 | - | A |
| 24 | - | A |
| 25 | - | A |
| 26 | D, A | E, F, C, B |
| 27 | - | A |
| 28 | - | A |
| 29 | A | B |
| 30 | - | - |
| 31 | - | A |
| 32 | - | A |
| 33 | B | - |
| 34 | - | - |
| 35 | - | A |
| 36 | A | - |
| 37 | A | B |
| 38 | - | - |
| 39 | A | B |
| 40 | - | A |
| 41 | A | B |
| 42 | - | - |
| 43 | - | - |
| 44 | - | - |
| 45 | - | - |
| 46 | - | - |
| 47 | - | - |
| 48 | - | - |
| 49 | - | - |
| 50 | - | A |
| 51 | - | - |
| 52 | D, A | C |
| 53 | - | A |
| 54 | C, A | B |
| 55 | A | - |
| 56 | A | B |
| 57 | - | A |
| 58 | A | - |
| 59 | - | - |
| 60 | - | - |
| 61 | A | B |
| 62 | A, H, D, E | F, B |
| 63 | - | - |
| 64 | A, D, C | B |
| 65 | B | A |
| 66 | C, A | B |
| 67 | - | - |
| 68 | - | - |
| 69 | - | A |
| 70 | - | A |

| | | |
|---|---|---|
| 71 | - | - |
| 72 | A | - |
| 73 | - | A |
| 74 | - | A |
| 75 | - | A |
| 76 | - | - |
| 77 | - | A |
| 78 | - | A |
| 79 | - | - |
| 8Ø | A | - |
| 81 | - | - |
| 82 | - | - |
| 83 | - | - |
| 84 | A | - |
| 85 | - | A |
| 86 | - | A |

## APPENDIX F: Document Logs

| P2R # | SIZE | SOURCE |
|---|---|---|
| ØØ1 | ØØ pgs. | Levin/Jones/Bladen(USAF) |
| ØØ2 | ØØ pgs. | (Boeing) |
| ØØ3 | ØØ pgs. | (Lear Siegler) |
| ØØ4 | ØØ pgs. | (Grumman - USAF) |
| ØØ5 | ØØ pgs. | (Boeing - USAF) |
| ØØ6 | ØØ pgs. | (TRW - USAF) |
| ØØ7 | ØØ pgs. | (ESD/TOIT - USAF) |
| ØØ8 | ØØ pgs. | (AFCCPC - USAF) |
| ØØ9 | ØØ pgs. | (Hq.SAC - USAF) |
| Ø1Ø | ØØ pgs. | (Aerospace Corp. - USAF) |
| Ø11 | ØØ pgs. | (RADC - USAF) |
| Ø12 | ØØ pgs. | (TI - USAF) |
| Ø13 | ØØ pgs. | (Sperry Univac) |
| Ø14 | ØØ pgs. | (French MOD) |
| Ø15 | ØØ pgs. | (Stanford AI lab) |

| | | |
|---|---|---|
| 016 | 00 pgs. | (U. of Grenoble) |
| 017 | 00 pgs. | Windaur/Goede(MB&P Ger) |
| 018 | 00 pgs. | Hilfinger/Newcommer(CMU) |
| 019 | 00 pgs. | (TRW - USAF) |
| 020 | 00 pgs. | (BGS) |
| 021 | 00 pgs. | Lebling(PDL) |
| 022 | 00 pgs. | Evans/Morgan/Forsdick(BBN) |
| 023 | 00 pgs. | Feirtag/Melliar-Smith(SRI) |
| 024 | 00 pgs. | Wulf(CMU) |
| 025 | 00 pgs. | (DCA) |
| 026 | 00 pgs. | Wirth(ETH Zurich) |
| 027 | 00 pgs. | (CORADCOM - ARMY) |
| 028 | 00 pgs. | (Brown U.) |
| 029 | 00 pgs. | (Swedish DRI) |
| 030 | 00 pgs. | Elzer(Donier System Gmbh) |
| 031 | 00 pgs. | Habermann (CMU) |
| 032 | 00 pgs. | (Mitre Corp.) |
| 033 | 00 pgs. | (NASA) |
| 034 | 00 pgs. | (General Electric) |
| 035 | 00 pgs. | (System Consultants, Inc.) |
| 036 | 00 pgs. | (IABG - Ger MOD) |
| 037 | 00 pgs. | (Universitat Karlsruhe) |
| 038 | 00 pgs. | (UK Dept. of Industry) |
| 039 | 00 pgs. | (LPT-E) |
| 040 | 00 pgs. | (UK MOD) |
| 041 | 00 pgs. | Schuman/Abrial(French Navy) |
| 042 | 00 pgs. | (Computer Sciences Corp. - USAF) |

| | | |
|---|---|---|
| 043 | 00 pgs. | (U. Texas) |
| 044 | 00 pgs. | (General Reasearch Corp.) |
| 045 | 00 pgs. | (HqMC, Code CCA-50) |
| 046 | 00 pgs. | Computer Science Dept.(CMU) |

| EVR # | SIZE | SOURCE | SUBJECT |
|---|---|---|---|
| 001 | 17 pgs. | (HOLWG) | Washington Review Summary |
| 002 | 09 pgs. | Fisher/Wetherall | Phase 2 Change Requests |
| 003 | 02 pgs. | Wulf | Change Requests |
| 004 | 02 pgs. | Good/London | Change Requests |
| 005 | 04 pgs. | (Navy) | Language Issue |
| 006 | 01 pgs. | (USAF) | Language Issues |
| 007 | 02 pgs. | (MOD) | Language Issues |
| 008 | 19 pgs. | Intermetrics | Language Issues |

| LIR # | SIZE | SOURCE | SUBJECT |
|---|---|---|---|
| 001 | 01 pgs. | I.C. Pyle (York) | Program Overlays |
| 002 | 03 pgs. | Andy Hisgen (CMU) | Timed Out Entry Calls |
| 003 | 03 pgs. | Andy Hisgen (CMU) | TASKING ERROR Exceptions |
| 004 | 09 pgs. | Tichy/Hubbard (CMU) | Separate Compilation |
| 005 | 09 pgs. | Saxe (CMU) | Functions and VRP's |
| 006 | 07 pgs. | Saxe/Smith (CMU) | User-Defined Types |
| 007 | 04 pgs. | Nassi (DEC) | TEXT_IO Proposals |
| 008 | 19 pgs. | Hilfinger (CMU) | Discriminant Constraints |
| 009 | 41 pgs. | Hilfinger (CMU) | Tasking Facilities |
| 010 | 02 pgs. | Firth (RMCS) | MOD Operator |

| | | | |
|---|---|---|---|
| 011 | 03 pgs. | Firth (RMCS) | Explicit Conversions |
| 012 | 02 pgs. | T. Sepan (Hughes) | Iteration Variable |
| 013 | 03 pgs. | Springer (IBM) | I-O Package |
| 014 | 02 pgs. | MacLaren (Boeing) | Fortran Interface |
| 015 | 03 pgs. | MacLaren (Boeing) | Entry Generic Parameters |
| 016 | 02 pgs. | Firth (RMCS) | Exception Handling |
| 017 | 05 pgs. | Firth (RMCS) | Parameter Binding |
| 018 | 02 pgs. | Firth (RMCS) | Variant Records |
| 019 | 02 pgs. | Firth (RMCS) | Suppressing ASSERT_ERROR |
| 020 | 11 pgs. | Firth (RMCS) | Semantics of Numerics |
| 021 | 09 pgs. | Woodger (UK) | LRM Clarifications |
| 022 | 03 pgs. | Firth (RMCS) | Task Termination |
| 023 | 02 pgs. | Firth (RMCS) | Compilation Units |
| 024 | 02 pgs. | Firth (RMCS) | EXIT WHEN Extensions |
| 025 | 02 pgs. | Firth (RMCS) | Allocator Function |
| 026 | 03 pgs. | Firth (RMCS) | SELECT Guards |
| 027 | 02 pgs. | Firth (RMCS) | RANGE Attribute |
| 028 | 02 pgs. | Firth (RMCS) | Array Generic Parameters |
| 029 | 02 pgs. | Firth (RMCS) | Derived Types |
| 030 | 02 pgs. | Firth (RMCS) | BOOLEAN Type |
| 031 | 03 pgs. | Firth (RMCS) | Procedures in Tasks |
| 032- | 01 pgs. | T. Sepan (Hughes) | Recursive/Reentrant |
| 033- | 01 pgs. | Taylor (Boeing) | Assertions |
| 034- | 03 pgs. | Goos (Karlsruhe) | Diverse Points |
| 035- | 03 pgs. | Goos (Karlsruhe) | Functions and Order |
| 036- | 02 pgs. | Goos (Karlsruhe) | Conditional Compilation |
| 037- | 01 pgs. | (Ger.MOD/IABG) | Absence of FREE |

| | | | |
|---|---|---|---|
| 038- | 02 pgs. | (Ger.MOD/IABG) | Visibility Restrictions |
| 039- | 01 pgs. | (Ger.MOD/IABG) | Recursive/Re-entrant |
| 040- | 01 pgs. | (Ger.MOD/IABG) | Numeric Literals |
| 041- | 01 pgs. | (Ger.MOD/IABG) | Keyword Overloading |
| 042- | 01 pgs. | (Ger.MOD/IABG) | MOD Operation |
| 043- | 01 pgs. | (Ger.MOD/IABG) | Array Bounds |
| 044- | 01 pgs. | (Ger.MOD/IABG) | Loop Syntax |
| 045- | 01 pgs. | (Ger.MOD/IABG) | INLINE Pragma |
| 046- | 02 pgs. | (Ger.MOD/IABG) | Input/Output |
| 047- | 02 pgs. | (Ger.MOD/IABG) | Keywords |
| 048- | 01 pgs. | (Ger.MOD/IABG) | Unhandled Exceptions |
| 049- | 01 pgs. | (Ger.MOD/IABG) | Visibility-Rules |
| 050- | 01 pgs. | (Ger.MOD/IABG) | Conditional Evaluation |
| 051- | 01 pgs. | (Ger.MOD/IABG) | Declaration Syntax |
| 052- | 01 pgs. | (Ger.MOD/IABG) | Syntax Description |
| 053- | 01 pgs. | (Ger.MOD/IABG) | Usage of DELAY |
| 054- | 01 pgs. | (Ger.MOD/IABG) | Incomplete Type Declaration |
| 055- | 01 pgs. | (Ger.MOD/IABG) | Dynamic Allocation |
| 056- | 01 pgs. | (Ger.MOD/IABG) | Scope Rules |
| 057- | 01 pgs. | (Ger.MOD/IABG) | Exceptions |
| 058- | 01 pgs. | (Ger.MOD/IABG) | Absence of SET Type |
| 059- | 01 pgs. | (Ger.MOD/IABG) | Pollution of Name-space |
| 060- | 01 pgs. | (Ger.MOD/IABG) | Low-Level Tasking |
| 061- | 01 pgs. | (Ger.MOD/IABG) | Asynchronous Communication |
| 062- | 01 pgs. | (Ger.MOD/IABG) | Storage Management |
| 063- | 02 pgs. | (Ger.MOD/IABG) | Type CHARACTER |
| 064- | 01 pgs. | (Ger.MOD/IABG) | Record Representations |
| 091- | 03 pgs. | Nagle(Ford Aerospace) | Integers |

| | | | |
|---|---|---|---|
| 065- | 01 pgs. | (Ger.MOD/IABG) | Loop Control |
| 066- | 02 pgs. | Firth (RMCS) | Named Block |
| 067- | 01 pgs. | Knut Ripkin | One-Component Aggregates |
| 068- | 02 pgs. | Knut Ripkin | Module Visibility |
| 069- | 01 pgs. | Fisher/Dewar (NYU) | Suppress Pragmat |
| 070- | 03 pgs. | Fisher/Dewar (NYU) | Generic Facility |
| 071- | 02 pgs. | Fisher/Dewar (NYU) | Assert_Error |
| 072- | 02 pgs. | Fisher/Dewar (NYU) | Labels and Goto's |
| 073- | 01 pgs. | Fisher/Dewar (NYU) | Boolean Operators |
| 074- | 02 pgs. | Fisher/Dewar (NYU) | Overloaded Literals |
| 075- | 01 pgs. | Fisher/Dewar (NYU) | Functions and VRP's |
| 076- | 01 pgs. | Fisher/Dewar (NYU) | Overloading |
| 077- | 01 pgs. | Fisher/Dewar (NYU) | IN parameters |
| 078- | 01 pgs. | Fisher/Dewar (NYU) | Keywords |
| 079- | 01 pgs. | Fisher/Dewar (NYU) | MOD Operator |
| 080- | 02 pgs. | Fisher/Dewar (NYU) | Scheduling Semantics |
| 081- | 02 pgs. | Fisher/Dewar (NYU) | Priorities |
| 082- | 01 pgs. | Fisher/Dewar (NYU) | Aliasing |
| 083- | 01 pgs. | Fisher/Dewar (NYU) | Definition of Semaphore |
| 084- | 01 pgs. | Fisher/Dewar (NYU) | Character Strings |
| 085- | 01 pgs. | Fisher/Dewar (NYU) | Character Strings |
| 086- | 01 pgs. | Fisher/Dewar (NYU) | OUT parameters |
| 087- | 01 pgs. | Fisher/Dewar (NYU) | Named Parameters |
| 088- | 01 pgs. | Fisher/Dewar (NYU) | No_Value_Error |
| 089- | 02 pgs. | Fisher/Dewar (NYU) | Select Statement |
| 090- | 02 pgs. | Fisher/Dewar (NYU) | Evaluation Order |
| 091- | 03 pgs. | Nagle(Ford Aerospace) | Integers |

| | | | |
|---|---|---|---|
| 092- | 02 pgs. | Firth (RMCS) | Overlapping Slice Assignment |
| 093- | 01 pgs. | (Ger.MOD/IABG) | Alignment Clause |
| 094- | 01 pgs. | (Ger.MOD/IABG) | END Statements |
| 095- | 02 pgs. | Andy Hisgen (CMU) | Package Elaboration |
| 096- | 03 pgs. | Hisgen/Tichy (CMU) | Package Initialization |
| 097- | 02 pgs. | Andy Hisgen (CMU) | Subprogram Result Values |
| 098- | 04 pgs. | Bruce Leverett (CMU) | Control Characters in I/O |
| 099- | 02 pgs. | Bruce Leverett (CMU) | Formatting: Put and Rep |
| 100- | 02 pgs. | Mary Shaw (CMU) | Derived Types |
| 101- | 04 pgs. | Wm. A. Wulf (CMU) | Access Type Constants |
| 102- | 02 pgs. | David R. Smith (CMU) | Access Types |
| 103- | 02 pgs. | David R. Smith (CMU) | The Characters ", $, # and ¦ |
| 104- | 02 pgs. | David R. Smith (CMU) | Floating Point Values |
| 105- | 02 pgs. | David R. Smith (CMU) | Reference Manual Problems |
| 106- | 01 pgs. | David R. Smith (CMU) | Array I/O |
| 107- | 02 pgs. | David R. Smith (CMU) | Binary Files/Mixed Types |
| 108- | 03 pgs. | David R. Smith (CMU) | Control Characters |
| 109- | 02 pgs. | David R. Smith (CMU) | I/O Buffering |
| 110- | 01 pgs. | David R. Smith (CMU) | Real Numbers |
| 111- | 05 pgs. | Ronald Brender (CMU) | Qualified Expressions |
| 112- | 02 pgs. | Ronald Brender (CMU) | Scope of Labels |
| 113- | 04 pgs. | David R. Smith (CMU) | Range Constraints |
| 114- | 02 pgs. | Wm. A. Wulf (CMU) | Properties of Operators |
| 115- | 01 pgs. | Joseph Newcomer (CMU) | Inadequacy of Index |
| 116- | 02 pgs. | Joseph Newcomer (CMU) | Problems with ORD |
| 117- | 01 pgs. | Nassi (DEC) | String Length |
| 118- | 01 pgs. | I.C. Pyle (York) | Ambiguous Stubs |

| | | | |
|---|---|---|---|
| 119- | 02 pgs. | Goodenough (SofTech) | FAILURE Exception ambiguity |
| 120- | 02 pgs. | (Ger.MOD/IABG) | Separate Compilation |
| 121- | 03 pgs. | Bruce Leverett (CMU) | Short Circuit Conditions |
| 122- | 03 pgs. | (Ger.MOD/IABG) | Scope of labels |
| 123- | 13 pgs. | E. Van Horn | Dynamic Storage Allocation |
| 124- | 03 pgs. | Firth (RMCS) | Task Runtime Identity |
| 125- | 01 pgs. | Finseth (MIT) | Tab Stop Columns |
| 126- | 01 pgs. | Finseth (MIT) | String Length |
| 127- | 01 pgs. | Finseth (MIT) | FREE Statement needed |
| 128- | 04 pgs. | Firth (RMCS) | Identification of stubs |
| 129- | 01 pgs. | Finseth (MIT) | Based Variables |
| 130- | 01 pgs. | Finseth (MIT) | Independent Compilation |
| 131- | 10 pgs. | Firth (RMCS) | Overloading |
| 132- | 03 pgs. | Firth (RMCS) | Access Constant |
| 133- | 05 pgs. | Firth (RMCS) | Virtual Record Components |
| 134- | 01 pgs. | Thompson (TRW/DSSG) | Array Aggregates |
| 135- | 01 pgs. | I.C. Pyle (York) | Aggregate Notation |
| 136- | 01 pgs. | I.C. Pyle (York) | Syntax of Names |
| 137- | 00 pgs. | T. Sepan (Hughes) | Need BLOCK DATA |
| 138- | 00 pgs. | F. Cox (Georgia Tech) | Visibility Control |
| 139- | 00 pgs. | F. Cox (Georgia Tech) | Visibility Restrictions |
| 140- | 00 pgs. | F. Cox (Georgia Tech) | Separate Subunits |
| 141- | 02 pgs. | Habermann (CMU) | Value Returning Procedures |
| 142- | 03 pgs. | Habermann (CMU) | Parameterized Types |
| 143- | 02 pgs. | Burkinshaw (IABG) | Default Values for Parameters |
| 144- | 01 pgs. | Michael Compton | Order of Compilation |
| 145- | 01 pgs. | Michael Compton | EXIT and other Loop Constructs |

| | | | |
|---|---|---|---|
| 146- | 07 pgs. | MacLaren (Boeing) | Interrupt Handlers |
| 147- | 04 pgs. | MacLaren (Boeing) | Shared Data |
| 148- | 02 pgs. | Levine (Intermetrics) | Numeric Literals |
| 149- | 01 pgs. | Levine (Intermetrics) | Variant Records |
| 150- | 02 pgs. | Levine (Intermetrics) | RANGE notation |
| 151- | 01 pgs. | Levine (Intermetrics) | Loop and Block Label Scope |
| 152- | 01 pgs. | Levine (Intermetrics) | Array Bounds Specification |
| 153- | 01 pgs. | Levine (Intermetrics) | Enumeration Literals |
| 154- | 02 pgs. | Levine (Intermetrics) | Integer Type Definition Form |
| 155- | 01 pgs. | Levine (Intermetrics) | T'PRED and T'SUCC |
| 156- | 01 pgs. | Levine (Intermetrics) | Component Selection |
| 157- | 04 pgs. | T. W. Jones | Diverse Points |
| 158- | 01 pgs. | R. Schwartz | Aliasing Restrictions |
| 159- | 02 pgs. | R. Schwartz | RENAMES Statement |
| 160- | 01 pgs. | J. Keeton (MITRE) | Pragma Semantics |
| 161- | 04 pgs. | Firth (RMCS) | Models of Access Types |
| 162- | 02 pgs. | D. Perry (CMU) | Permissive Syntax |
| 163- | 02 pgs. | D. Perry (CMU) | Partial Aggregate Initialization |
| 164- | 02 pgs. | D. Perry (CMU) | Consistent Initialization |
| 165- | 00 pgs. | J. T. Galkowski (IBM) | Field Names in Variants |
| 166- | 00 pgs. | J. T. Galkowski (IBM) | Recursion is Efficient |
| 167- | 00 pgs. | J. A. Edwards | Obsolete |
| 168- | 00 pgs. | J. A. Edwards | Obsolete |
| 169- | 00 pgs. | J. A. Edwards | Obsolete |
| 170- | 00 pgs. | Joe Farley | Obsolete |
| 171- | 00 pgs. | J. A. Edwards | Obsolete |
| 172- | 00 pgs. | J. A. Edwards | Obsolete |

| | | | |
|---|---|---|---|
| 173- | 00 pgs. | J. A. Edwards | Obsolete |
| 174- | 00 pgs. | J. A. Edwards | Obsolete |
| 175- | 00 pgs. | J. A. Edwards | Obsolete |
| 176- | 00 pgs. | D. Jones | MOD Function |
| 177- | 01 pgs. | T. Hastings (DEC) | Varying Strings |
| 178- | 01 pgs. | T. Hastings (DEC) | Functional Arguments |
| 179- | 01 pgs. | John Sauter (DEC) | Floating Exponents |
| 180- | 01 pgs. | Dennis Noble | Interrupt Handling |
| 181- | 01 pgs. | Michael King (NWC) | TAB Character |
| 182- | 01 pgs. | Michael King (NWC) | Need for Simulated IO |
| 183- | 01 pgs. | Michael King (NWC) | GET_LINE and PUT_LINE Needed |
| 184- | 02 pgs. | Michael King (NWC) | Exception Handler |
| 185- | 02 pgs. | Michael King (NWC) | Need More Loop Constructs |
| 186- | 01 pgs. | Michael King (NWC) | Lack of User Defined Literals |
| 187- | 01 pgs. | R. Krutar (NRL) | PACKAGE and TASK |
| 188- | 01 pgs. | R. Krutar (NRL) | PROCEDURE and ENTRY |
| 189- | 01 pgs. | R. Krutar (NRL) | Entry Calls |
| 190- | 01 pgs. | R. Krutar (NRL) | Subprogram Calling |
| 191- | 01 pgs. | R. Krutar (NRL) | Subprogram Calling |
| 192- | 01 pgs. | R. Krutar (NRL) | AND THEN and OR ELSE |
| 193- | 01 pgs. | R. Krutar (NRL) | Comment Convention |
| 194- | 02 pgs. | R. Krutar (NRL) | EXIT Statement |
| 195- | 03 pgs. | Marc Hubbard (NWC) | Fixed Point Rounding |
| 196- | 03 pgs. | H. Huber (NWC) | Generic Facility Inadequate |
| 197- | 02 pgs. | H. Wettstein (IABG) | Tasking Facilities |
| 198- | 02 pgs. | H. Huber (NWC) | Scope Rules Ambiguous |
| 199- | 02 pgs. | Levine (Intermetrics) | Short Circuit Conditions |

| | | | |
|---|---|---|---|
| 200- | 01 pgs. | Levine (Intermetrics) | Access Constants too Restricted |
| 201- | 01 pgs. | Levine (Intermetrics) | Anonymous Access Types |
| 202- | 03 pgs. | Levine (Intermetrics) | Ada Grammar Allows Empty Fields |
| 203- | 02 pgs. | Ellis Thomas (SDL) | Identification of Stubs |
| 204- | 02 pgs. | D. Notkin (CMU) | Derived Types in Ada |
| 205- | 02 pgs. | D. Notkin (CMU) | Ada Syntax |
| 206- | 02 pgs. | D. Notkin (CMU) | Ada I/O |
| 207- | 00 pgs. | Michael Compton | Relax Generic Parm Constraints |
| 208- | 00 pgs. | Michael Compton | Range Checking |
| 209- | 00 pgs. | Michael Compton | Assertion Facilities |
| 210- | 00 pgs. | L. J. Gallaher | Variant Records of Type Access |
| 211- | 00 pgs. | R. Johnson (Boeing) | Macro Facility |
| 212- | 00 pgs. | R. Johnson (Boeing) | FREE Statement is Needed |
| 213- | 00 pgs. | J. T. Galkowski (IBM) | Field Names in Variants |
| 214- | 00 pgs. | Goos (Karlsruhe) | Subprogram Calls |
| 215- | 00 pgs. | Goos (Karlsruhe) | Generic Clauses |
| 216- | 00 pgs. | Goos (Karlsruhe) | Access Type Objects |
| 217- | 03 pgs. | Ellis Thomas (SDL) | Inconsistencies in the LRM |
| 218- | 01 pgs. | UK DOI RTL/2 Team | Null Statement |
| 219- | 02 pgs. | UK DOI RTL/2 Team | Use Clause |
| 220- | 02 pgs. | UK DOI RTL/2 Team | Empty Subranges |
| 221- | 02 pgs. | UK DOI RTL/2 Team | Types of Array Bounds |
| 222- | 02 pgs. | UK DOI RTL/2 Team | Named Blocks |
| 223- | 02 pgs. | UK DOI RTL/2 Team | Range Notation |
| 224- | 01 pgs. | UK DOI RTL/2 Team | Loop Indices |
| 225- | 03 pgs. | UK DOI RTL/2 Team | Syntax of Names |
| 226- | 02 pgs. | UK DOI RTL/2 Team | Visibility Rules |

| | | | |
|---|---|---|---|
| 227- | 03 pgs. | UK DOI RTL/2 Team | Arrays and Strings |
| 228- | 02 pgs. | UK DOI RTL/2 Team | Syntax Rules for Types |
| 229- | 00 pgs. | Michael Compton | USE Clause Redundant |
| 230- | 00 pgs. | Michael Compton | Short Circuit; Boolean Syntax |
| 231- | 00 pgs. | Michael Compton | SELECT Statement |
| 232- | 00 pgs. | Michael Compton | Fixed Point Variables |
| 233- | 00 pgs. | Michael Compton | Garbage Collection |
| 234- | 00 pgs. | Michael Compton | READONLY EXPORT Desired |
| 235- | 00 pgs. | Michael Compton | RANGE Attribute |
| 236- | 00 pgs. | Michael Compton | Private Parts of Specifications |
| 237- | 00 pgs. | Michael Compton | Restricted Types |
| 238- | 00 pgs. | Michael Compton | Formatted I/O |
| 239- | 00 pgs. | Michael Compton | Interrupts; Context Switching |
| 240- | 00 pgs. | Michael Compton | Exception Declaration |
| 241- | 00 pgs. | Michael Compton | Compilation Unit Genealogy |
| 242- | 00 pgs. | J. A. Edwards | EXIT and ABORT |
| 243- | 00 pgs. | J. A. Edwards | Short Circuit Conditions |
| 244- | 00 pgs. | J. A. Edwards | Exception Overhead; Assert |
| 245- | 00 pgs. | Joe Farley | Bitstring Literals |
| 246- | 00 pgs. | J. A. Edwards | Storage Management |
| 247- | 00 pgs. | J. A. Edwards | Package Declarations |
| 248- | 00 pgs. | J. A. Edwards | Task Scheduling; Select |
| 249- | 00 pgs. | J. A. Edwards | FOR/USE Overloading |
| 250- | 00 pgs. | J. A. Edwards | Selector etc. Syntax |
| 251- | 00 pgs. | J. T. Galkowski (IBM) | WHILE...LOOP Superfluous |
| 252- | 00 pgs. | J. T. Galkowski (IBM) | Position of USE |
| 253- | 00 pgs. | J. T. Galkowski (IBM) | Functions and VRP's |

| | | | |
|---|---|---|---|
| 254- | 00 pgs. | J. T. Galkowski (IBM) | Forbid Intertask Data Sharing |
| 255- | 00 pgs | J. T. Galkowski (IBM) | Current Tasking Best |
| 256- | 00 pgs. | J. T. Galkowski (IBM) | Current Binding Semantics Best |
| 257- | 00 pgs. | J. T. Galkowski (IBM) | Overlapping Slice Assignment |
| 258- | 00 pgs. | A. G. B. Cooper | Generic Type Parameters |
| 259- | 00 pgs. | A. G. B. Cooper | Selective Importing |
| 260- | 00 pgs. | A. G. B. Cooper | Bit Position Attributes |
| 261- | 00 pgs. | A. G. B. Cooper | Incrementing etc. (self) |
| 262- | 00 pgs. | A. G. B. Cooper | OUT Parameters in Calls |
| 263- | 00 pgs. | A. G. B. Cooper | 'Size of Program Units |
| 264- | 00 pgs. | A. G. B. Cooper | Noncontiguity and Addresses |
| 265- | 00 pgs. | A. G. B. Cooper | Better Strings Desired |
| 266- | 00 pgs. | A. G. B. Cooper | Subtypes of Anonymous Types |
| 267- | 00 pgs. | A. G. B. Cooper | Keyword Parameters Liked |
| 268- | 00 pgs. | A. G. B. Cooper | Operation Inheritance |
| 269- | 00 pgs. | A. G. B. Cooper | Overloaded Relational Operators |
| 270- | 00 pgs. | A. J. M. Van Gils | Syntax of Accuracy-Constraint |
| 271- | 00 pgs. | A. J. M. Van Gils | Procedure Call on LHS |
| 272- | 00 pgs. | A. J. M. Van Gils | Syntax of Primary |
| 273- | 00 pgs. | A. J. M. Van Gils | Subprogram Attribute |
| 274- | 00 pgs. | A. J. M. Van Gils | Short Circuit Conditions |
| 275- | 00 pgs. | A. J. M. Van Gils | WHILE Redundant |
| 276- | 00 pgs. | A. J. M. Van Gils | Private Representation Secificat: |
| 277- | 00 pgs. | A. J. M. Van Gils | Overloading Disambiguation |
| 278- | 00 pgs. | A. J. M. Van Gils | Give Initiate Parameters |
| 279- | 00 pgs. | A. J. M. Van Gils | Consistency for Tasks, Modules |
| 280- | 00 pgs. | A. J. M. Van Gils | ACCEPT Parameters Scope |

| | | | |
|---|---|---|---|
| 281- | 00 pgs. | A. J. M. Van Gils | Visibility Lists |
| 282- | 00 pgs. | A. J. M. Van Gils | Task Variables |
| 283- | 00 pgs. | A. J. M. Van Gils | Distributed Systems |
| 284- | 00 pgs. | A. J. M. Van Gils | Task Termination |
| 285- | 00 pgs. | A. J. M. Van Gils | Except Propagation from Rendezuou |
| 286- | 00 pgs. | A. J. M. Van Gils | Representation of Enumerals |
| 287- | 00 pgs. | A. J. M. Van Gils | Syntax of Generic Parameter |
| 288- | 00 pgs. | A. J. M. Van Gils | Generic Instantiation Semantics |
| 289- | 00 pgs. | A. J. M. Van Gils | Generic Parameters |
| 290- | 00 pgs. | A. J. M. Van Gils | 'Size of Generic Type Parameters |
| 291- | 00 pgs. | A. J. M. Van Gils | Allow Entries as Generic Paramete |
| 292- | 00 pgs. | A. J. M. Van Gils | Syntactic Position of Pragmas |
| 293- | 00 pgs. | A. J. M. Van Gils | Syntax Ambiguous |
| 294- | 02 pgs. | William A. Whitaker | Radix Syntax |
| 295- | 01 pgs. | William A. Whitaker | Completeness of Formal Syntax |
| 296- | 00 pgs. | J. T. Galkowski (IBM) | Interface Convention |
| 297- | 02 pgs. | Firth (RMCS) | Character Literals |
| 298- | 01 pgs. | William A. Whitaker | Priorities and Hardware |
| 299- | 01 pgs. | William A. Whitaker | I/O Formatting |
| 300- | 02 pgs. | William A. Whitaker | OPTIMIZE SPACE or TIME |
| 301- | 03 pgs. | William A. Whitaker | Better TIME and INTERVAL |
| 302- | 01 pgs. | UK DOI Coral66 Team | RENAMES Allowed Only as Declarati |
| 303- | 01 pgs. | UK DOI Coral66 Team | Use and Restricted |
| 304- | 01 pgs. | Ellis Thomas (SDL) | Separate Compilation and Overload |
| 305- | 01 pgs. | UK DOI Coral66 Team | Use and Name Space |
| 306- | 02 pgs. | Goodenough (SofTech) | Fixed Point Representations |
| 307- | 02 pgs. | William A. Whitaker | Use Fortran-like Relational Opera |

| | | | |
|---|---|---|---|
| 308- | 03 pgs. | William A. Whitaker | Banish Vertical Bar (\|) |
| 309- | 01 pgs. | William A. Whitaker | Rename "Unsafe_Programming" |
| 310- | 01 pgs. | Ellis Thomas (SDL) | Modules Without Bodies |
| 311- | 02 pgs. | Thomas & Gilbert (SDL) | Local Tasks |
| 312- | 06 pgs. | M. Devlin (AFSCF/BJB) | Banish Subtypes |
| 313- | 02 pgs. | William A. Whitaker | Right Arrow Symbol |
| 314- | 01 pgs. | William A. Whitaker | Quote Character |
| 315- | 02 pgs. | William A. Whitaker | Use "=" for Assignment |
| 316- | 02 pgs. | William A. Whitaker | Range Endpoints |
| 317- | 02 pgs. | William A. Whitaker | MOD and REM |
| 318- | 01 pgs. | William A. Whitaker | LRM Erratum |
| 319- | 02 pgs. | MacLaren (Boeing) | Timed Out Entry Calls |
| 320- | 01 pgs. | Ellis Thomas (SDL) | Number Syntax |
| 321- | 01 pgs. | Thomas & Gilbert (SDL) | Separate Compilation & Linking |
| 322- | 01 pgs. | Lieberman & Kiernan | Parameter Syntax |
| 323- | 01 pgs. | Lieberman & Kiernan | Meaning of "Module" |
| 324- | 01 pgs. | Lieberman & Kiernan | Eliminate Goto |
| 325- | 01 pgs. | Lieberman & Kiernan | Package Elaboration/Initializatic |
| 326- | 01 pgs. | Lieberman & Kiernan | Local Static Variables |
| 327- | 02 pgs. | T. Conrad (NUSC) | Device I/O |
| 328- | 01 pgs. | T. Conrad (NUSC) | Educational Materials Needed |
| 329- | 01 pgs. | William A. Whitaker | Parameter Syntax |
| 330- | 03 pgs. | William A. Whitaker | Floating Precision not Digits |
| 331- | 02 pgs. | William A. Whitaker | Non-Binary Machines |
| 332- | 01 pgs. | William A. Whitaker | "Do" in Accept |
| 333- | 02 pgs. | William A. Whitaker | Predefine Operators for All Type |
| 334- | 02 pgs. | William A. Whitaker | Selection by Parentheses |

| | | | |
|---|---|---|---|
| 335- | 00 pgs. | T. A. Montgomery | Procedural Variables and Argument |
| 336- | 00 pgs. | T. A. Montgomery | Arrays of Arrays |
| 337- | 00 pgs. | T. A. Montgomery | Locatives |
| 338- | 00 pgs. | T. A. Montgomery | Slicing Clumsy |
| 339- | 00 pgs. | T. A. Montgomery | Local Declarations Verbose |
| 340- | 00 pgs. | T. A. Montgomery | Conditional Expressions Desired |
| 341- | 00 pgs. | T. A. Montgomery | Allow Statements in Conditions |
| 342- | 00 pgs. | T. A. Montgomery | Bit Vector Operators |
| 343- | 00 pgs. | T. A. Montgomery | Null Valve |
| 344- | 00 pgs. | T. A. Montgomery | VRP's Overly Constrained |
| 345- | 00 pgs. | T. A. Montgomery | Embedded Comments Desired |
| 346- | 00 pgs. | T. A. Montgomery | Make "_" Not Significant |
| 347- | 00 pgs. | T. A. Montgomery | Parameter Association Symbols |
| 348- | 00 pgs. | Arthur Sorkin (SEL) | Dangling References |
| 349- | 00 pgs. | Janet Louie (IBM) | Word-Level Referencing |
| 350- | 00 pgs. | Janet Louie (IBM) | Fixed Point Representation |
| 351- | 00 pgs. | Janet Louie (IBM) | Where is Bit 0? |
| 352- | 00 pgs. | Janet Louia (IBM) | Address Specification Vague |
| 353- | 02 pgs. | MacLaren (Boeing) | Preemptive Priority Scheduling |
| 354- | 02 pgs. | UK DOI Coral66 Team | SUSPEND and RESUME of Tasks |
| 355- | 01 pgs. | UK DOI RTL/2 Team | Default Initial Values for All T |
| 356- | 01 pgs. | UK DOI RTL/2 Team | The notation .all is strange |
| 357- | 01 pgs. | UK DOI RTL/2 Team | Precedence of Unary Operators |
| 358- | 01 pgs. | UK DOI RTL/2 Team | MOD Operators |
| 359- | 01 pgs. | UK DOI RTL/2 Team | Entry Calls with Timeout |
| 360- | 01 pgs. | UK DOI Coral66 Team | Timeouts |
| 361- | 02 pgs. | UK DOI Coral66 Team | Incomplete Record Assignment Des |

| | | | |
|---|---|---|---|
| 362- | 01 pgs. | UK DOI Coral66 Team | Quoted Literal Ambiguity |
| 363- | 02 pgs. | UK DOI Coral66 Team | Task Variables |
| 364- | 01 pgs. | UK DOI Coral66 Team | Arrays of Arrays - Strings |
| 365- | 01 pgs. | UK DOI Fortran Team | Large Attributes |
| 366- | 01 pgs. | UK DOI Fortran Team | Underflow |
| 367- | 02 pgs. | UK DOI Fortran Team | Exceptions |
| 368- | 02 pgs. | UK DOI Fortran Team | Aliasing |
| 369- | 01 pgs. | UK DOI Fortran Team | Procedure Parameters |
| 370- | 01 pgs. | UK DOI Fortran Team | Array Layout and paging |
| 371- | 01 pgs. | UK DOI RTL/2 Team | I/O is Inadequate |
| 372- | 00 pgs. | Nils Jorgen Olsson | Text_IO Procedure Names |
| 373- | 00 pgs. | Nils Jorgen Olsson | Built-in Queues |
| 374- | 00 pgs. | Svon Tafvelin | Initiate with Arguments |
| 375- | 00 pgs. | Sven Tafvelin | Waiting for Resources |
| 376- | 00 pgs. | Sven Tafvelin | I/O Timeouts |
| 377- | 00 pgs. | L. J. Gallaher | "Exchange" Operator |
| 378- | 01 pgs. | Burkinshaw (IABG) | Long Identifiers |
| 379- | 01 pgs. | I. C. Pyle (York) | Real Type Attr: 'LARGE, 'BITS, e |
| 380- | 01 pgs. | I. C. Pyle (York) | 'SIZE of Access Types |
| 381- | 01 pgs. | I. C. Pyle (York) | 'SIZE of Records with Variants. |
| 382- | 01 pgs. | I. C. Pyle (York) | When Clause Irregular |
| 383- | 00 pgs. | J. T. Galkowski (IBM) | Integers as Pure Ranges |
| 384- | 00 pgs. | Arthur Sorkin (SEL) | Restricted Private Type Init. |
| 385- | 00 pgs. | Bureau of the Census | Unsafe Discriminant Setting |
| 386- | 00 pgs. | Bureau of the Census | Need Variable Length Strings |
| 387- | 00 pgs. | Bureau of the Census | Access to Laternal Formats |
| 388- | 00 pgs. | Bureau of the Census | Type Attributes in Generics |

| | | | |
|---|---|---|---|
| 389- | 00 pgs. | W. B. Carson | Case Syntax Awkward |
| 390- | 00 pgs. | Wayne Johnson (IBM) | Pointers to Static Objects |
| 391- | 00 pgs. | Wayne Johnson (IBM) | Binary Point Position |
| 392- | 00 pgs. | Wayne Johnson (IBM) | Predefine Common Math Functions |
| 393- | 00 pgs. | Wayne Johnson (IBM) | Make Time an Integer |
| 394- | 00 pgs. | Jonas Agerberg | Character Set |
| 395- | 00 pgs. | Jonas Agerberg | I/O Function Names |
| 396- | 00 pgs. | Jonas Agerberg | Single-Memory Orientation |
| 397- | 00 pgs. | Arthur Sorkin (SRL) | Allow Entries in Selects |
| 398- | 00 pgs. | William Eventoff | Visibility and Generics |
| 399- | 00 pgs. | William Eventoff | Identifying "other" Exceptions |
| 400- | 00 pgs. | Eventoff & Rabinowitz | Provide Sets |
| 401- | 00 pgs. | Eventoff & Rabinowitz | "Delta" Bad Keyword |
| 402- | 00 pgs. | Eventoff & Rabinowitz | Allow Embedded Comments |
| 403- | 00 pgs. | Eventoff & Rabinowitz | Accuracy-Constraint Syntax |
| 404- | 00 pgs. | Eventoff & Rabinowitz | Strings Inadequate |
| 405- | 00 pgs. | Christopher Henrich | Aggregate Notation |
| 406- | 00 pgs. | Greg Burns (ITT) | Buffer Tasks: Rendezvous Delegat: |
| 407- | 00 pgs. | Greg Burns (ITT) | Task Family Attributes |
| 408- | 00 pgs. | Harry Carl (Honeywell) | Garbage Collection |
| 409- | 00 pgs. | Harry Carl (Honeywell) | Require Specified Optimizations |
| 410- | 00 pgs. | Harry Carl (Honeywell) | Run Time Environment |
| 411- | 00 pgs. | Harry Carl (Honeywell) | Many Bodies with One Name |
| 412- | 00 pgs. | Braudaway & Louie (IBM) | Fixed-point Representation |
| 413- | 00 pgs. | Wayne Johnson (IBM) | Fixed-point Representation |
| 414- | 00 pgs. | Wayne Johnson (IBM) | Pointing to Static Objects |
| 415- | 00 pgs. | Braudaway & Johnson | Time |

| | | | |
|---|---|---|---|
| 416- | 02 pgs. | A. J. Scarpelli (AFAL) | Access Type Initialization |
| 417- | 01 pgs. | A. J. Scarpelli (AFAL) | Access Types for Small Objects |
| 418- | 02 pgs. | Richard Wolff (NWC) | Conversion of User-Defined Types |
| 419- | 01 pgs. | W. Roos & J. Cross | Component Names as Generic Parm |
| 420- | 01 pgs. | D. Holdsworth | Fixed-Length-Result 'Rep Desired |
| 421- | 01 pgs. | K. Hopper | Exist Predicate for Files Desire |
| 422- | 01 pgs. | Charles Eckert | Time should be Fixed or Integer |
| 423- | 01 pgs. | Charles Eckert | Fixed-Point Scale Factors Desire |
| 424- | 01 pgs. | Charles Eckert | Machine Code Insertions Clumsy |
| 425- | 01 pgs. | Charles Eckert | Allow .#, #., and # for Float Le |
| 426- | 01 pgs. | K. Hopper | Require Init. (given no NO_VAL_E |
| 427- | 01 pgs. | K. Hopper | Priority on Entry Queues |
| 428- | 01 pgs. | John Hutchison | Anonymous Types Lack Attributes |
| 429- | 01 pgs. | John Hutchison | Dynamic Length Spec for Access T |
| 430- | 01 pgs. | Bill Robinson | End of File as Predicate |
| 431- | 01 pgs. | Bill Robinson | VRP = Function + Pragma |
| 432- | 01 pgs. | P. Burkinshaw (IABG) | Multiple Assignments |
| 433- | 01 pgs. | P. Burkinshaw (IABG) | For Variable Scope (Exceptions) |
| 434- | 01 pgs. | P. Burkinshaw (IABG) | Collapsed Else's; Multiple End I |
| 435- | 01 pgs. | Ellis Thomas (SDL) | Subprogram Names in Visibility L |
| 436- | 01 pgs. | Bill Robinson | Multiple Subunits - Same Name |
| 437- | 01 pgs. | Bill Robinson | Logical Operators' Precedence Ru |
| 438- | 02 pgs. | William A. Whitaker | Expression Evaluation Order |
| 439- | 02 pgs. | William A. Whitaker | MOD |
| 440- | 02 pgs. | Firth (RMCS) | Remove Exit When |
| 441- | 01 pgs. | Brian Wichmann | Entries Declared on Task Bodies |
| 442- | 00 pgs. | Thomas J. Wheeler | Accept Syntax |

| | | | |
|---|---|---|---|
| 443- | 00 pgs. | David Gries | And then vs. Cand |
| 444- | 01 pgs. | I. C. Pyle | Entries as Subprograms |
| 445- | 01 pgs. | Maureen E. Gordon | Restricted Clause and Enclosure |
| 446- | 01 pgs. | Maureen E. Gordon | Use Clause Redundant |
| 447- | 01 pgs. | Maureen E. Gordon | Order of Declaration Elaboration |
| 448- | 01 pgs. | Maureen E. Gordon | Logical Operators' Precedence |
| 449- | 01 pgs. | Maureen E. Gordon | Allow Body Separation Everywhere |
| 450- | 01 pgs. | Maureen E. Gordon | Allow Reference/Copy Choice |
| 451- | 01 pgs. | Maureen E. Gordon | Clarify Unsafe_Conversion |
| 452- | 01 pgs. | Maureen E. Gordon | Entry Call Time-Out |
| 453- | 01 pgs. | Maureen E. Gordon | Data Locking |
| 454- | 03 pgs. | Ronald Brender (DEC) | Eliminate/Clarify Environment Pr |
| 455- | 00 pgs. | Dan W. Scott | Array Slicing; Array Syntax |
| 456- | 00 pgs. | Dan W. Scott | Strings |
| 457- | 00 pgs. | Dan W. Scott | Constant Record Components' Name |
| 458- | 00 pgs. | Dan W. Scott | Discriminants |
| 459- | 00 pgs. | Dan W. Scott | Generic Parameter Permutation |
| 460- | 00 pgs. | Dan W. Scott | Reentrancy; Own |
| 461- | 00 pgs. | Dan W. Scott | Humorous Comment |
| 462- | 00 pgs. | Dan W. Scott | Representation |
| 463- | 00 pgs. | Dan W. Scott | Representation of Arrays |
| 464- | 00 pgs. | Dan W. Scott | LRM Index Shortcomings |
| 465- | 00 pgs. | David T. Moore | Resumptive Exceptions |
| 466- | 00 pgs. | David T. Moore | Identifying Instances of Exceptic |
| 467- | 00 pgs. | David T. Moore | Loop Indices' Declaration |
| 468- | 00 pgs. | Ray Van Tassle | Obsolete |
| 469- | 00 pgs. | I. C. Pyle | 'Address of an Overloaded Subproc |

| | | | |
|---|---|---|---|
| 470- | 00 pgs. | I. C. Pyle | Type of 'Delta and 'Small? |
| 471- | 00 pgs. | I. C. Pyle | What are 'System' and 'Option'? |
| 472- | 00 pgs. | I. C. Pyle | Priority as Type Name |
| 473- | 00 pgs. | I. C. Pyle | Generic Subprog as Generic Parm |
| 474- | 00 pgs. | I. C. Pyle | Field Names, etc. as Generic Par |
| 475- | 00 pgs. | I. C. Pyle | Exceptions in Generic Packages |
| 476- | 00 pgs. | I. C. Pyle | Types as Array Bounds |
| 477- | 00 pgs. | Dan W. Scott | Access Types: Allocation, Init. |
| 478- | 00 pgs. | Dan W. Scott | Access Type Initialization |
| 479- | 00 pgs. | Dan W. Scott | 'Free' Operation |
| 480- | 00 pgs. | Dan W. Scott | Dereferencing Considered Clumsy |
| 481- | 00 pgs. | Dan W. Scott | .all of Arrays |
| 482- | 00 pgs. | Dan W. Scott | "$" and "_" in Identifiers |
| 483- | 00 pgs. | Dan W. Scott | Qualification Syntax Disliked |
| 484- | 00 pgs. | Dan W. Scott | 'Restricted'; Blocks' Visibility |
| 485- | 00 pgs. | Dan W. Scott | Constants |
| 486- | 00 pgs. | Dan W. Scott | Types Derived from Private Types |
| 487- | 00 pgs. | Dan W. Scott | Multidemensional Arrays |
| 488- | 00 pgs. | Dan W. Scott | Scope names |
| 489- | 00 pgs. | Dan W. Scott | [...] in Meta-Syntax |
| 490- | 00 pgs. | Dan W. Scott | Selected Component Syntax |
| 491- | 00 pgs. | Dan W. Scott | Corrigendum 4.1.3 |
| 492- | 00 pgs. | Ada Group Tokyo | What is System? |
| 493- | 00 pgs. | Ada Group Tokyo | "string $" and "string $ string" |
| 494- | 00 pgs. | Ada Group Tokyo | Strings of Length One |
| 495- | 00 pgs. | Ada Group Tokyo | Non-Access Incomplete Type Decl. |
| 496- | 00 pgs. | Ada Group Tokyo | Is a Subty Compatible w/its Base |

| | | | |
|---|---|---|---|
| 497- | 00 pgs. | Ada Group Tokyo | Allow Init. for Non-Record Type |
| 498- | 00 pgs. | Ada Group Tokyo | Inheritance of Subprg by Derive |
| 499- | 00 pgs. | Ada Group Tokyo | Syntax of Character_Literal? |
| 500- | 00 pgs. | Ada Group Tokyo | Allow Short_Integer + Integer |
| 501- | 00 pgs. | Ada Group Tokyo | Compatibility among Int, Short_ |
| 502- | 00 pgs. | Ada Group Tokyo | What are Floating & Fixed point |
| 503- | 00 pgs. | Ada Group Tokyo | Clarify 'small and 'large |
| 504- | 00 pgs. | Ada Group Tokyo | 'small and 'large of Fixed Poin |
| 505- | 00 pgs. | Ada Group Tokyo | Make I in 'Length (I) Static |
| 506- | 00 pgs. | Ada Group Tokyo | What is the Type of Subarrays? |
| 507- | 00 pgs. | Ada Group Tokyo | Type of Range Components in For |
| 508- | 00 pgs. | Ada Group Tokyo | Bounds of Dynamic Arrays |
| 509- | 00 pgs. | Ada Group Tokyo | Is "2 ! Others => 0" Legal? |
| 510- | 00 pgs. | Ada Group Tokyo | Allow Only One Dynamic Arr per |
| 511- | 00 pgs. | Ada Group Tokyo | What is complete Record Assignm |
| 512- | 00 pgs. | Ada Group Tokyo | Arrays with Index Type Integer |
| 513- | 00 pgs. | Ada Group Tokyo | Multi-dim Arr as Arrays of Arra |
| 514- | 00 pgs. | Ada Group Tokyo | What is a Simple Name? |
| 515- | 00 pgs. | Ada Group Tokyo | Ambiguity of Subprg as Type Att |
| 516- | 00 pgs. | Ada Group Tokyo | Define.Type Compatibility for O |
| 517- | 00 pgs. | Ada Group Tokyo | Logical Oper Arrays of Diff Bou |
| 518- | 00 pgs. | Ada Group Tokyo | Define Result Accuracy Precisel |
| 519- | 00 pgs. | Ada Group Tokyo | Is Integer**0 Allowed? |
| 520- | 00 pgs. | Ada Group Tokyo | Improve Type Qualification Exam |
| 521- | 00 pgs. | Ada Group Tokyo | Define Int (Real) Unambiguously |
| 522- | 00 pgs. | Ada Group Tokyo | Clarify Static Expressions |
| 523- | 00 pgs. | Ada Group Tokyo | Compatibility of Multi-Dim Arra |

| | | | |
|---|---|---|---|
| 524- | 00 pgs. | Ada Group Tokyo | What is a Qualified Variable? |
| 525- | 02 pgs. | Maureen E. Gordon | Interleave Reps. with Declaratio |
| 526- | 00 pgs. | Ada Group Tokyo | Exception Propagation |
| 527- | 00 pgs. | Ada Group Tokyo | Disallow Overloading of Generics |
| 528- | 00 pgs. | Ada Group Tokyo | Inter-Overload of Generic & Subp |
| 529- | 00 pgs. | Ada Group Tokyo | Generic Overloading |
| 530- | 00 pgs. | Ada Group Tokyo | Improve Rep/Aggregate Syntax |
| 531- | 00 pgs. | Ada Group Tokyo | Clarify "Integer" in Appendix A |
| 532- | 00 pgs. | Ada Group Tokyo | Prag Envir., Include Change Sem. |
| 533- | 00 pgs. | Ada Group Tokyo | Clarify Optimize Pragma |
| 534- | 00 pgs. | Ada Group Tokyo | Clarify Page Pragma |
| 535- | 00 pgs. | Ada Group Tokyo | Clarify List Pragma |
| 536- | 00 pgs. | Ada Group Tokyo | Clarify Include Pragma |
| 537- | 00 pgs. | Ada Group Tokyo | Improve LRM Index |
| 538- | 00 pgs. | Ada Group Tokyo | [illegible]ccessed Objects Passed "i |
| 539- | 00 pgs. | Ada Group Tokyo | Define Label after End Loop |
| 540- | 00 pgs. | Ada Group Tokyo | Syntax of Modules Ambiguous |
| 541- | 00 pgs. | Ada Group Tokyo | Are Defaults Allowed for Cut&InC |
| 542- | 00 pgs. | Ada Group Tokyo | Can an Out Formal be Read in Bod |
| 543- | 00 pgs. | Ada Group Tokyo | Do Cnstr on Actual Apply to Form |
| 544- | 00 pgs. | Ada Group Tokyo | Can Recursive Subprograms be Inl |
| 545- | 00 pgs. | Ada Group Tokyo | Define Identity of Signatures |
| 546- | 00 pgs. | Ada Group Tokyo | Check Func. No_Val_Err Staticall |
| 547- | 00 pgs. | Ada Group Tokyo | Are Defaults Part of Signatures? |
| 548- | 00 pgs. | Ada Group Tokyo | Label Scope |
| 549- | 00 pgs. | Ada Group Tokyo | Name Space of Labels |
| 550- | 00 pgs. | Ada Group Tokyo | Clarify Overloading vs. Hiding |

| | | | |
|---|---|---|---|
| 551- | 00 pgs. | Ada Group Tokyo | Clarify Scope and Visibility |
| 552- | 00 pgs. | Ada Group Tokyo | Contents of Visibility Lists |
| 553- | 00 pgs. | Ada Group Tokyo | Use" Conflict Rule |
| 554- | 00 pgs. | Ada Group Tokyo | What is the End Ident. in Accept? |
| 555- | 00 pgs. | Ada Group Tokyo | Explain Tasking Except. in Task S |
| 556- | 00 pgs. | Ada Group Tokyo | What is a Program Library? |
| 557- | 00 pgs. | Ada Group Tokyo | Define Predefined Except. Precise |
| 558- | 00 pgs. | Ada Group Tokyo | Cross-Reference Tasking_Error |
| 559- | 00 pgs. | Ada Group Tokyo | Let "Other" Includ Out of Scope E |
| 560- | 00 pgs. | Japan Joint Systems | Imbedded Comments Desired |
| 561- | 00 pgs. | Japan Joint Systems | Partial Attr Inheritance by Subty |
| 562- | 00 pgs. | Japan Joint Systems | Derived Access Types |
| 563- | 00 pgs. | Japan Joint Systems | Subtyping and Recursive Types |
| 564- | 00 pgs. | Japan Joint Systems | Expr. Evaluation Order Over-Restr |
| 565- | 00 pgs. | Japan Joint Systems | Corrigendum 4.5.2 |
| 566- | 00 pgs. | Japan Joint Systems | Free Operation Desired |
| 567- | 00 pgs. | Japan Joint System | Multidim. Arrays as Arrays of Arr |
| 568- | 00 pgs. | Japan Joint Systems | Default Positional Parms. Desirec |
| 569- | 00 pgs. | Japan Joint Systems | Label Scope |
| 570- | 00 pgs. | Japan Joint Systems | Selective Import Desired |
| 571- | 00 pgs. | Japan Joint Systems | Define Label after End |
| 572- | 00 pgs. | Japan Joint Systems | Clarify Initiation |
| 573- | 00 pgs. | Japan Joint Systems | Corrigendum 10.1 |
| 574- | 00 pgs. | Japan Joint Systems | Specifying Enclosing Unit of Subu |
| 575- | 00 pgs. | Japan Joint Systems | Optm Discussions belong in Ratior |
| 576- | 00 pgs. | Japan Joint Systems | Disignator as Generic Parameter |
| 577- | 00 pgs. | Japan Joint Systems | Type of Length Expression |

| | | | |
|---|---|---|---|
| 578- | 00 pgs. | Japan Joint Systems | Improve Other-Language Interface |
| 579- | 00 pgs. | Japan Joint Systems | Improve I/O: Terminals, Records |
| 580- | 00 pgs. | Japan Joint Systems | Corrigendum 14.3.2 |
| 581- | 00 pgs. | Japan Joint Systems | Clarify Definition of Aliasing |
| 582- | 00 pgs. | Japan Joint Systems | Overloading Disambiguation |
| 583- | 00 pgs. | Japan Joint Systems | Syntax Irregularities |
| 584- | 00 pgs. | Japan Joint Systems | Pragmas etc. Are Not Language Iss |
| 585- | 00 pgs. | Akira Nagashima (CMU) | Make Fixed-Point Delta Exact |
| 586- | 00 pgs. | Akira Nagashima (CMU) | User-Defined Assignment; VRPs on |
| 587- | 00 pgs. | Akira Nagashima (CMU) | Allow Entry Overloading |
| 588- | 00 pgs. | Akira Nagashima (CMU) | Allow Deferred Const as Discrim C |
| 589- | 00 pgs. | Akira Nagashima (CMU) | Provide Discrim Cnstr in Allocato |
| 590- | 00 pgs. | Bjarne Dacker (Sweden) | Defining Accept/Call Relations at |
| 591- | 00 pgs. | Bjarne Dacker (Sweden) | No-Wait Message Passing Desired |
| 592- | 00 pgs. | Bjarne Dacker (Sweden) | Dynamic Task Identification Desir |
| 593- | 00 pgs. | Thomas J. Pennello | Subtypes as Ranges |
| 594- | 00 pgs. | Thomas J. Pennello | Add User-Defined Operators |
| 595- | 00 pgs. | Thomas J. Pennello | Conditional Expressions Valuable |
| 596- | 00 pgs. | Thomas J. Pennello | Iterators Desired |
| 597- | 00 pgs. | Thomas J. Pennello | "Restricted" Overloaded |
| 598- | 00 pgs. | Thomas J. Pennello | "New" Overloaded |
| 599- | 00 pgs. | Thomas J. Pennello | Base-Type Function |
| 600- | 00 pgs. | Thomas J. Pennello | Unordered Enumerations Desired |
| 601- | 00 pgs. | Thomas J. Pennello | "Range" Overloaded |
| 602- | 00 pgs. | Thomas J. Pennello | Distinguish Loop from Goto Labels |
| 603- | 00 pgs. | Thomas J. Pennello | "Is" Overloaded |
| 604- | 00 pgs. | Thomas J. Pennello | Separate Visibility from Importat |

| | | | |
|---|---|---|---|
| 605- | 00 pgs. | J.F.H. Winkler | Syntax Comments |
| 606- | 00 pgs. | Richard J. Meyers | Loop & Goto Labels |
| 607- | 00 pgs. | Richard J. Meyers | Overloading "New" |
| 608- | 00 pgs. | Richard J. Meyers | AND THEN and OR ELSE Anywhere |
| 609- | 00 pgs. | Richard J. Meyers | State Subunit's Identity |
| 610- | 00 pgs. | Richard J. Meyers | Asynchronous Entries |
| 611- | 00 pgs. | Richard J. Meyers | Restricted Overloaded |
| 612- | 00 pgs. | Richard J. Meyers | Types as Discrete Ranges |
| 613- | 00 pgs. | Ray Van Tassle | Unsigned Integer Type |
| 614- | 00 pgs. | Ray Van Tassle | Incrementing etc (":+=") |
| 615- | 00 pgs. | (Univ. of Copenhagen) | Distinguish Types from Subtypes |
| 616- | 00 pgs. | (Univ. of Copenhagen) | Define Label after End Statement |
| 617- | 00 pgs. | (Univ. of Copenhagen) | Label Scope |
| 618- | 00 pgs. | (Univ. of Copenhagen) | Syntax of Subprogram Attributes |
| 619- | 00 pgs. | (Univ. of Copenhagen) | Improve Identification of Subprgs |
| 620- | 00 pgs. | (Univ. of Copenhagen) | Semantics of Task Failure |
| 621- | 00 pgs. | (Univ. of Copenhagen) | Semantics of Abort |
| 622- | 00 pgs. | (Univ. of Copenhagen) | Separate Compilation |
| 623- | 00 pgs. | (Univ. of Copenhagen) | Allow Functional Arguments |
| 624- | 00 pgs. | (Univ. of Copenhagen) | Syntax Incomplete and Ambiguous |
| 625- | 00 pgs. | (Univ. of Copenhagen) | Regularize Declaration Syntax |
| 626- | 00 pgs. | Frank De Remer | Allow Static Expressions in Pragr |
| 627- | 00 pgs. | Frank De Remer | Oper Designators; User-Coined Ope |
| 628- | 00 pgs. | Frank De Remer | Tighten Up Syntax |
| 629- | 00 pgs. | Frank De Remer | Overloading of Keywords |
| 630- | 00 pgs. | Frank De Remer | Regularize Declaration Syntax (LA |
| 631- | 00 pgs. | Frank De Remer | Freer Placement of Rep. Specs. |

| | | | |
|---|---|---|---|
| 632- | 00 pgs. | Frank De Remer | Distinguish Loop from Goto Labels |
| 633- | 00 pgs. | Frank De Remer | Restructure Unit Header Syntax |
| 634- | 00 pgs. | Frank De Remer | Union Types; Iterators |
| 635- | 00 pgs. | Frank De Remer | Conditional Expressions Valuable |
| 636- | 00 pgs. | Frank De Remer | Subtypes as Ranges |
| 637- | 00 pgs. | Frank De Remer | Cap. Non-Terminals; "..." Metasyn |
| 638- | 00 pgs. | Frank De Remer | Restrictive Clauses: "That" |
| 639- | 00 pgs. | Frank De Remer | Lexical Grammar |
| 640- | 00 pgs. | Frank De Remer | Remove Redeclaration Restrictions |
| 641- | 06 pgs. | Goodenough (SofTech) | Optimization and Exceptions |
| 642- | 04 pgs. | Firth (RMCS) | AND THEN and OR ELSE |
| 643- | 02 pgs. | Goodenough (SofTech) | Omitted Exceptions? |
| 644- | 02 pgs. | Benjamin M. Brosgol | Semantic Checking of Generic Bodi |
| 645- | 04 pgs. | Goodenough (SofTech) | References to Unelaborated Object |
| 646- | 07 pgs. | Goodenough (SofTech) | Efficient Machine Code Insertions |
| 647- | 01 pgs. | Belmont (Intermetrics) | Type in Range Constraints |
| 648- | 01 pgs. | William A. Whitaker | Order of Evaluation |
| 649- | 01 pgs. | W. Paul Sherer | Allow Overlapping Slice Assignmen |
| 650- | 04 pgs. | William A. Whitaker | Need for a FREQUENCY Pragma |
| 651 | 00 pgs. | Foldesson (SAAB-SCANIA) | Delay and Cyclic Programs |
| 652 | 00 pgs. | Douglas W. Jones | Parameters and Tasking |

| COM # | SIZE | SOURCE | SUBJECT |
|---|---|---|---|
| 001- | 01 pgs. | Nagle (Ford Aero.) | Integer Semantics |
| 002 | 01 pgs. | Goodenough | LIR.003 |
| 003 | 01 pgs. | Goodenough | EVR.002 |

| | | | |
|---|---|---|---|
| 004- | 08 pgs. | Firth | Comments on DCR 1-6 (V1) |
| 005- | 01 pgs. | Holdworth | Standard Portable Language |
| 006- | 01 pgs. | Belmont | Structure of MODULES |
| 007- | 01 pgs. | Belmont | Separate Compilation |
| 008- | 04 pgs. | Goodenough | Notes, Aug.6 Review Meeting |
| 009- | 01 pgs. | Gillmann | Ada Syntax |
| 010- | 05 pgs. | Compton | 16 Diverse Points |
| 011- | 02 pgs. | Belz | Revised DCR.003 |
| 012- | 01 pgs. | Gillmann | Constant arrays and records |
| 013- | 01 pgs. | German MOD | LCR.003 |
| 014- | 164pgs. | Habermann et al. | Diverse Points, GANDALF |
| 015- | 05 pgs. | Firth | Parameter Passing |
| 016- | 02 pgs. | M. Ben-Ari | Blocks, Short Circuits |
| 017- | 01 pgs. | S. Ljungkuist | Set Type |
| 018- | 03 pgs. | E. M. Greene | General Comments |
| 019- | 01 pgs. | D. T. Moore | 'ADDRESS Attribute/Registers |
| 020- | 16 pgs. | Robert Milne | General Comments |
| 021- | 03 pgs. | MacLaren | Physical Interfaces |
| 022- | 13 pgs. | Hilfinger | Comments on DCR.003 |
| 023- | 03 pgs. | Firth | Side-Effects |
| 024- | 11 pgs. | Firth | Minutes of Sept. Meeting |
| 025- | 01 pgs. | Firth | Draft on Side-Effects |
| 026- | 01 pgs. | Brownlie | Tasking |
| 027- | 02 pgs. | Firth | Thoughts on Ada T&E |
| 028- | 00 pgs. | C. Yandow | Task/Module Distinction |
| 029- | 00 pgs. | D. T. Moore | Indexing into Records |
| 030- | 00 pgs. | J. D. Cox | Proposed Enhancements |

| | | | |
|---|---|---|---|
| 031- | 00 pgs. | J. Byrd | Negative Numbers |
| 032- | 00 pgs. | R. D. Johnson | Macro Facility Needed |
| 033- | 00 pgs. | R. D. Johnson | FREE Needed |
| 034- | 00 pgs. | M. T. Devlin | Simplifications |
| 035- | 01 pgs. | Firth | Comments on v3 DCR's |
| 036- | 07 pgs. | Goodenough | Comments on v3 DCR's |
| 037- | 02 pgs. | MacLaren | Comments on v3 DCR's |
| 038- | 01 pgs. | Goodenough | Comments on v4 DCR's |
| 039- | 01 pgs. | Davis | Response to COM.029 |
| 040- | 08 pgs. | Evans | Ada Tasking |
| 041- | 03 pgs. | Hilfinger | Comments on COM.040 |
| 042- | 03 pgs. | MacLaren | Interface Concepts |
| 043- | 02 pgs. | Hilfinger | Comments on COM.042 |
| 044- | 07 pgs. | MacLaren | Interface Costs |
| 045- | 00 pgs. | Shulman | Comments on Reference Manual |
| 046- | 00 pgs. | Cooper | Binary File IO In Ada |
| 047- | 00 pgs. | Archer | ADA Subset Definition |
| 048- | 02 pgs. | Paul Hilfinger | Comments on Interface Costs |
| 049- | 00 pgs. | Mark Hapner | Parameter Binding Semantics |
| 050- | 00 pgs. | Mark Hapner | Large Applications |
| 051- | 00 pgs. | Mark Hapner | Comment on DCR.002v3 |
| 052- | 00 pgs. | Mark Hapner | Reentrancy |
| 053- | 00 pgs. | Mark Hapner | Compound Type Constraints |
| 054- | 00 pgs. | Mark Hapner | Tasking |
| 055- | 00 pgs. | Kenneth Dickey | Square Brackets for Subscripts |
| 056- | 00 pgs. | J. K. Reid | Intrinsic Functions for Floats |
| 057- | 00 pgs. | Rudolf Marty | Padding on String Assignment |

| | | | |
|---|---|---|---|
| 058- | 01 pgs. | Jean E. Sammet | Need for Real Time Clock |
| 059- | 09 pgs. | Richard L. Schwartz | Aliasing and Ada |
| 060- | 00 pgs. | Serafino Amoroso | BCL's review of Ada |
| 061- | 01 pgs. | Robert Firth | Comment on LIR.203 |
| 062- | 02 pgs. | NADC-ADA | Static Variables |
| 063- | 02 pgs. | Robert Firth | Ada Construction Kit |
| 064- | 01 pgs. | Lee MacLaren | Fixed Point Representation |
| 065- | 01 pgs. | Lee MacLaren | Dynamic Priorities |
| 066- | 01 pgs. | Paul Hilfinger | Foreign Procedure Parameters |
| 067- | 07 pgs. | Robert Firth | Ada 'Blackboard' Issues |
| 068- | 02 pgs. | Mark Davis | Named Parameters and Overloading |
| 069- | 01 pgs. | Mark Davis | Side-effects and Functionality |
| 070- | 00 pgs. | A. D. Hill | I/O; Exceptions; etc. |
| 071- | 00 pgs. | M. K. Shen | Suspension; Scheduling; Goto |
| 072- | 00 pgs. | Francisco Oyarzun | Priv. part; Derefer; Incomplete t |
| 073- | 00 pgs. | James A. Harle | Listings; Pragmas |
| 074- | 00 pgs. | Alexander Goodall | Select Statement |
| 075- | 00 pgs. | D. G. Elliott | Numbers; Extensibility; etc. |
| 076- | 00 pgs. | Neil Parker | Form of Manual; Syntax; etc. |
| 077- | 00 pgs. | Thomas R. Amoth | Repeat Until and While Do |
| 078- | 00 pgs. | A. Silberschatz | Accepts vs. "Parts" |
| 079- | 00 pgs. | C. H. Lindsey | Neologisms; Various Points |
| 080- | 00 pgs. | W. R. LaLonde | Strings |
| 081- | 00 pgs. | (IABG) | Systems Programming: Various Poin |
| 082- | 00 pgs. | Lawrence J. Gallaher | Scheduling; Iteration |
| 083- | 00 pgs. | Raymond T. Boute | Arbitrary Restric: Various Points |
| 084- | 10 pgs. | Lee MacLaren (Boeing) | Object-Oriented Synchronization |

| | | | |
|---|---|---|---|
| 085- | 03 pgs. | William Whitaker | Pascal Standards Meeting |
| 086- | 01 pgs. | MacLaren | Waiting at Scope Exit |
| 087- | 01 pgs. | MacLaren | No Tasking_Errors in Servers |
| 088- | 00 pgs. | (Univ. of Tokyo) | Give Examples of Pathologies |
| 089- | 04 pgs. | MacLaren | Entry Families vs. Task Objects |
| 090- | 00 pgs. | Dr. Neumann (Germany) | Extended Rendezvous |
| 091- | 00 pgs. | Dr. Neumann (Germany) | Telegram Communication |
| 092- | 00 pgs. | Teodor Rus (Rumania) | Semantic Formalization |
| 093- | 00 pgs. | (Finland) | Various Points |
| 094- | 00 pgs. | Andrew Arblaster | Human Factors Report |
| 095- | 00 pgs. | Peter Wallis | Literals of User-Defined Type |
| 096- | 00 pgs. | (Germany) | Else Syntax Error-Prone |
| 097- | 00 pgs. | W. R. LaLonde | Strings |
| 098- | 00 pgs. | Thomas A. Marciniak | Permanent Data Structures etc. |
| 099- | 00 pgs. | A. E. Chandler | Constant Graphs |
| 100- | 03 pgs. | Firth (RMCS) | Recursion is Efficient |
| 101- | 02 pgs. | Firth (RMCS) | Overloading |
| 102- | 04 pgs. | Firth (RMCS) | Low Level Tasking |
| 103 | 00 pgs. | R. Schwartz (SRI) | Artificial Intelligence Applicati |
| 104 | 00 pgs. | G. Bage (L M Ericsson) | Generalization of Tasking |
| 105 | 00 pgs. | Folkesson (SAAB-SCANIA) | Cycles: Delay and 'Clock |
| 106 | 00 pgs. | J. Welsh & A. Lister | Tasking: CCSP and DPS |
| 107 | 00 pgs. | (BBN Inc.) | Task Efficiency and Multiprocessi |
| 108 | 02 pgs. | Davis Stevenson | Fairness and the H-N Technique |
| 109 | 11 pgs. | (Intel Corp.) | Extensions |
| 110 | 00 pgs. | (Univ. of Copenhagen) | Comments on Preliminary Ada |

| POS # | SIZE | SOURCE | SUBJECT |
|---|---|---|---|
| 001 | 1 pgs. | UK MOD | Interrupt Handling |
| 002 | 3 pgs. | UK MOD | Modern Architectures |
| 003 | 01 pg. | Firth (MOD) | Side Effects and Optimization |

| DCR # | SIZE | SUBJECT |
|---|---|---|
| 001v4 | 01 pgs. | Parameter Binding Semantics |
| 002v4 | 01 pgs. | Parameter Passing Conventions |
| 003v4 | 02 pgs. | Physical Interfaces |
| 004v3 | 01 pgs. | Array Slice Assignment |
| 005v3 | 01 pgs. | Exceptions In Declarations |
| 006v3 | 01 pgs. | Real Time Clock |
| 007v3 | 02 pgs. | Side-Effects and Functionality |
| 008v4 | 02 pgs. | Type Compostion |

| OPA # | SIZE | SOURCE | SUBJECT |
|---|---|---|---|
| 001 | 1 pgs. | Design Team | Transfer Statements |
| 002 | 1 pgs. | Design Team | Garbage Collection |
| 003 | 1 pgs. | Design Team | Composite Type Equality |
| 004 | 1 pgs. | Design Team | Restricted Types |
| 005 | 1 pgs. | Design Team | Access Variable Initialization |
| 006 | 1 pgs. | Design Team | Access Types, Allocators |
| 007 | 1 pgs. | Design Team | Spin Locks, etc. |
| 008 | 1 pgs. | Design Team | Functions, VRP's |

| | | | |
|---|---|---|---|
| ØØ9 | 1 pgs. | Design Team | Cumulative Processing Time |
| Ø1Ø | 1 pgs. | Design Team | Type TIME |
| Ø11 | 1 pgs. | Design Team | Parameter Passing |
| Ø12 | 1 pgs. | Design Team | Exception Handler |
| Ø13 | 1 pgs. | Design Team | Array Aggregates |
| Ø14 | 1 pgs. | Design Team | Null String |
| Ø15 | 1 pgs. | Design Team | NO_VALUE_ERROR |
| Ø16 | 1 pgs. | Design Team | Compilation Unit Naming |
| Ø17 | 1 pgs. | Design Team | Constant Record Components |
| Ø18 | 1 pgs. | Design Team | Syntax: [name.]designator |
| Ø19 | 1 pgs. | Design Team | Task Initiation |
| Ø2Ø | 1 pgs. | Design Team | Task Procedures |

## APPENDIX G

## TER TOPICS, SUMMARIES, AND EXTRACTS

<u>Topics Mentioned in I&E Reports</u>

## Chapter 1

Add: Language subsets: 25
Change: Make declaration syntax more uniform: 30
Improve syntax: 4
Require blocks rather than sequence of statements: 30

## Chapter 2

Add: Abbreviations for keywords: 3, 30
Imbedded comments: 30, 72
Alternate character set support: 13
Bit string constants: 13, 41, 44, 51, 59
Redund: Bases other than 2, 8, 10, and 16: 16

### Identifiers

Change: Make "_" non-significant: 30, 48
Like: "_" in identifiers: 19
Long identifiers: 19, 37, 75
Redund: Significance of "_" in tokens: 7

## Chapter 3

Add: Strings: 29, 35, 38, 45, 59, 61, 63, 72
Bit handling: 26, 71, 77
Function as data: 7
Multi-level structures: 3
Reference variables: 7, 19, 30
Simula classes: 7
Static allocation of access objects: 13
Unsafe pointers: 14
Variable declarations after local program bodies: 64
Static variables: 34
Change: "=>" has two meanings: 19, 30
Ranges should not have to be contiguous: 30
Delta is poor keyword: 19
Expressions in range constraints(?): 6
Require specification of maximum size of strings: 2
Store matrices by column: 16
Types too restrictive: 15
Allow anonymous types in record fields: 26
Use structure equivalence for arrays: 30
Guaranteed one-step conversion between derived types: 30
Like: Aggregate syntax: 7
Aggregates: 29, 40
Arrays: 13
Enumeration types: 7, 34, 35, 37, 38, 58, 68, 75, 80
Derived types: 38
Machine-independent data definition: 2
Overloading: 2, 7, 35, 37, 42, 61

Record syntax: 19
Record variant semantics: 29
Initialization in declarations: 86
Strong typing: 2, 3, 10, 16, 18, 26, 29, 31, 46, 48, 50, 52, 54, 58, 61, 62, 63, 71, 72, 73, 77, 80, 86
Variant arrays in records: 86
Arrays with unspecified index range: 86
Type constraints: 1, 20, 49
User-defined types: 5, 17, 26
Scope for access types: 29

Redund: Subtypes: 67
Either subtypes or derived types: 19
Derived types: 29
Named components in aggregates: 25

**Numbers**

Add: Implicit conversion of numeric types (when no loss of precision): 30
Change: Fixed-point delta should be exact: 27, 28
Like: Precision specification: 13

**Chapter 4**

Add: Conditional expressions: 7, 28, 30
Multiple assignments: 30
Method of expressing parallelism in expression evaluation: 21
'Free' operation: 29, 69
Standard built-in math library: 19
Standard builtin array operations: 16, 19

Change: Accurate fixed point arithmetic (specification, coercion): 6, 85, 88
Define mathematical properties of user-defined operators: 1
More control over allocation: 13, 15
Qualified expression syntax: 13
Time should not be floating point: 19, 86
User type names should be overloadable as conversion functions: 83

Like: Attributes: 20, 21, 29
Expression structure: 19
Array slicing: 29, 88
No automatic type conversion: 14

Redund: Allocators for access types: 1
Array slicing: 18

**Chapter 5**

Add: Compound statements: 7
Block exits: 63
Exit from named block: 30

Change: Remove mandatory semicolon before end, elsif, etc.: 30
Allow mixing of "and then" and "or else": 26
Allow VRP's as conditions: 30
Overloading rules too complicated w.r.t. parameters: 26

| | |
|---|---|
| Redund: | Elsif: 18, 19 |
| | Function call syntax: 20, 66 |
| | Keyword parameter-association syntax (:=:, etc.): 7, 19, 27 |
| | Assert: 30, 54, 64 |
| | Labels and gotos: 1, 4, 30, 64 |
| | Short circuit conditions: 18, 54, 66 |

Loops

| | |
|---|---|
| Add: | More loop constructs: 13, 16, 27, 67 |
| Change: | Use "do" not "loop" as keyword: 19 |
| Like: | Structured programming constructs: 2, 5, 10, 13 |
| Redund: | Exit when: 6, 54, 65 |
| | Exit: 4 |

## Chapter 6

| | |
|---|---|
| Add: | Functional arguments: 6, 20, 21, 28, 40, 74, 65 |
| | Intermixed declarations: 7 |
| | Generalize initialization in type declarations: 30 |
| | Not recursive/reentrant declaration: 2 |
| | Variable number of parameters: 29 |
| | Guaranteed by-value calls: 64 |
| Change: | Define parameter passing: 18 |
| | Reference passing preferred: 14 |
| | Functionality should not be compiler-verified: 66 |
| Like: | Initialization in declarations: 29 |
| | Default parameters: 7, 29, 35 |
| | Recursion: 20, 21, 22 |
| | Functions and VRP's: 21, 22 |
| | Parameter modes: 36 |
| Redund: | Declarations in blocks: 7 |
| | Default parameters: 25 |
| | Initial values in declarations: 25 |
| | Optionality of block declarations: 1 |
| | Recursion support: 13 |
| | Tasks and Procedures should be merged: 5 |
| | VRP's: 26, 29 |

## Chapter 7

| | |
|---|---|
| Change: | Allow representations in private part: 28 |
| Like: | Information hiding/data abstraction in general: 10, 14, 20, 34 |
| | Packages: 4, 6, 10, 16, 29, 40, 46, 50, 52, 56, 56, 61, 66, 73 |
| | Private types, parts: 2, 8 |
| | Separate specifications: 1, 2 13, 19, 38, 47, 50, 56, 69 |
| Redund: | Nested packages: 15 |
| | Scoping hierarchy: 13 |
| | Separate specifications: 16 |

# Chapter 8

Change: Clarification of separate compilation and visibility: 1
Loop index should be valid beyond end of loop: 27
Restricted is poor keyword: 19
Visibility rules disliked: 13, 46, 49

Like: Logical scope rules: 16, 18
Restricted visibility: 4, 22, 23, 55, 87
Private types: 87

Redund: Use clause: 25

# Chapter 9

Add: Background tasks: 13
Initiate parameters: 11
Mutual exclusion to data access: 22, 23
Timed-out entry calls: 30, 86, 87
Suspend and resume of tasks: 82
Easier cyclic scheduling: 88

Change: Disallow data shared between tasks: 20, 21
Forbid aborting or changing priority of parent tasks: 6, 85
Interrupt semantics: 13, 26, 82
More control over scheduling: 13, 26, 82
Preemptive priorities: 27
Rendezvous too restrictive: 15
Static priority: 1

Like: Tasking: 4, 10, 20, 21, 27, 29, 33, 71, 75, 77, 83, 85, 86, 88
Task families: 86
Rendezvous arguments: 29, 38

Redund: Tasking too complex: 15
Signals and semaphores: 30

# Chapter 10

Change: Allowing deferred constants to be set in a separate compilation unit: 2
Have different visibility rules for separate compilation: 30
Separate units should have full upward visibility: 87

Like: Program structure: 16
Separate compilation: 10, 19, 26, 54, 66, 72, 73, 87

# Chapter 11

Like: Exception handling: 7, 16, 20, 29, 33, 38, 56, 86
Change: Exceptions in declarative parts should propagate up: 66

# Chapter 12

| | |
|---|---|
| Add: | Type restrictions for generic parameters: 6, 65 |
| | Component names as generic parameters: 29 |
| Change: | Generics: 20 |
| Like: | Generics: 2, 16, 36, 56, 68, 66, 87, 66 |
| Redund: | Generics: 3, 69 |

## Chapter 13

| | |
|---|---|
| Add: | Overlays: 1, 26 |
| | Representation of integers as bit fields: 16 |
| | Records with overlapping fields: 29 |
| | Representation specification of fixed point binary point: 16, 19 |
| | Better Fortran interface: 67 |
| Change: | Improve alignment specifications: 13 |
| | Machine code inserts clumsy: 15, 41, 47 |
| | Incorporate representations into type definitions: 27 |
| Like: | Record representation: 19, 38, 44 |
| | Representation specifications: 19, 27, 56, 66 |
| | Machine-code insertions: 27 |
| | Unsafe conversion: 66 |

## Chapter 14

| | |
|---|---|
| Add: | Timeout on I/O: 11 |
| | Fortran-like Formats: 0 |
| | Mixed-mode files: 62 |
| | A high-level real-time I/O mechanism: 62 |
| Change: | EOF not exception: 14, 62 |
| | I/O incomplete: 13 |
| | Operating system assumed too big: 13 |
| | Extend Text_IO: 0 |
| Like: | I/O as package: 1, 7 |
| Redund: | Send_control, Receive_control (in Low_level_IO): 1 |

## Chapter 2

| | |
|---|---|
| Change: | Keywords are overloaded: 67 |
| Like: | Matching keywords (eg if -- endif): 67 |

Test and Evaluation Report Data Summary
24 March 1980

**Format**

Number. Institution [country]: author
General description -- (R) means program was redesigned.
Original language(s).
Host computer(s) -> Target computer(s) (if given).
Number of Ada statements and identifers used (if given).

"-" indicates that the information was not given in the TER

1. University of York [England]: I. C. Pyle
   Non-text I/O of coded time signals in real time
   Modula
   DEC PDP-11 -> DEC PDP-11
   28 statements: 32 identifiers

2. Hughes Aircraft: Tony Sepan
   Real time, multiprogramming system
   Hiftran (Structured Fortran)
   DEC PDP-11/70 -> DEC PDP-11/70
   276 statements: 151 identifiers

3. - [Japan]: -
   PL/I syntax checker
   CPL-B (PL/I subset)
   - -> Fujitsu M, Hitachi M, NEC ACOS, Mitsubishi Cosmo
   478 statements: 140 identifiers

4. Aerospace Corp.: Charles A. Crummer
   -
   -
   IBM 360/75 -> IBM ASP 202
   200 statements: 70 identifiers

5. -: Lt. Robert C. Seigrist
   Student text processing exercise
   Cobol
   Burroughs 6700 -> Burroughs 6700
   37 statements: 10 identifiers

6. Institute for Defense Analyses: V. Schneider
   PAS Real-time executive
   SPL (Jovial)
   CDC 7600 -> RCA SCP-234
   104 statements

7. International Computers Ltd. [England]: T. A. Montgomery, I. Marshall
   Formatted listing of compiler output (CREF, Map, etc.)
   S3 (Algol 68)
   ICL 2900 -> ICL 2900
   611 statements: 230 identifiers

8. Naval Surface Weapons Center: Marc Hubbard
   Real time fire control
   Assembler
   IBM 370, UYK-20 -> UKK-20
   191 statements: 39 identifiers

9. Air Force Armament Division: Lt. Col. William A. Whitaker
   Inertial guidance--computational kernel (R)
   Fortran, Jovial
   0 statements: 0 identifiers

10. Computer Sciences Corporation: Dale D. Hurtig
    Real time digital autopilot
    Assembler, Fortran Subset
    HP -> Special purpose micro
    148 statements: 118 identifiers

11. Chalmers University of Technology [Sweden]: Sven Tafvelin
    Data buffering and spot processing in a radar system (R)
    Pascal

12. RADC/ISIS: Capt. Clair Rolla
    Data manipulation, word packing and unpacking
    Jovial
    Honeywell 6080 -> Honeywell 6080
    550 statements

13. General Dynamics: -
    Real-time, multiprogramming, data bases, network support
    C
    DEC PDP 11/34 -> DEC PDP 11/34

14. IBM Corporation: -
    Character handling, video display formatting, control block formatting
    Assembler
    IBM 360 -> IBM 360
    689 statements: 336 identifiers

15. IBM Corporation: -
    Telops: Satellite data capture, storage, and retrieval
    Assembler
    IBM 370 -> IBM 370

16. IBM Corporation: -
    VEPC: Signal processing simulation: bits, arrays, numbers
    Fortran
    IBM 370 -> IBM 370

17. IBM Corporation: -
    Terminal communications package: character string translation
    Fortran
    Interdata 8/32 -> Interdata 8/32
    256 statements: 35 identifiers

18. IBM Corporation: - (R)
    Fixed point, I/O, representation
    CMS-2Y
    AN/UYK-7 -> AN/UYK-7
    339 statements: 99 identifiers

19. IBM Corporation: -
    Signal processing: real-time, low-level I/O
    SPL (Assembler)
    CDC 6600 -> AN/UYS-1
    625 statements: 210 identifiers

20. IBM Corporation: -
    Bit manipulation, message translation, real time communications (R)
    Fortran, Assembler
    IBM 370 -> IBM 4P1/ML-1

21. IBM Corporation: -
    Mathematical computation, real time processing (R)
    Assembler, Fortran
    IBM 370 -> IBM 4P1/ASP

22. IBM Corporation: -
    Real-time Processing
    Assembler
    IBM 370 -> Zilog Z80

23. IBM Corporation: -
    Character Handling, String handling (R)
    Assembler
    IBM 370 -> IBM 370

24. IBM Corporation: -
    String & character handling, minor mathematical computations
    PL/1
    IBM 370 -> IBM 370
    155 statements: 31 identifiers

25. IBM Corporation: -
    Solo: Single-user operating system
    Pascal
    DEC PDP-11/45 -> DEC PDP-11/45
    1288 statements: 462 identifiers

26. Grumman Aerospace Corporation: Charles Mooney
    Real time trainer: equations of motion (R)
    Fortran
    Interdata 8/32 -> Interdata 8/32
    155 statements: 197 identifiers

27. E-Systems Inc.: T. W. Jones
    Hardware driver: I/O, bits, real-time
    Assembler
    UYK-20 -> UYK-20
    83 statements: 31 identifiers

28. System Development Corporation: Erwin Book
    Simulation of "21" table (R)
    Modula, ALGOL, Sue, Jovial
    Burroughs 7700, IBM 370, ANFSQ-32 -> Burroughs 7700, IBM 370, ANFSQ-32
    350 statements: 143 identifiers

29. Sperry Univac, Defense Systems Division: -
    Display fault table: characters, data-base, reentrancy (R)
    DSPL (Pascal)
    Univac 1100, Univac 1600, AN/UYK-20, Univac 1600, AN/UYK-20 -> N

30. SPL International [England]: Brian Dobbing
    Process control: real-time, operator I/O
    RTL/2
    DEC PDP-11/34 -> DEC PDP-11/04
    588 statements: 580 identifiers

31. Hollandse Signaal Apperaten B.V. [Netherlands]: Phillip van Liere
    Instrument servo control (R)
    RTL/2, Assembler
    Hollandse Signaal SMR-MU -> Hollandse Signaal SMR-MU

32. Raytheon Company: T. Nedzynski
    Interactive coordinate transformations: matrix operations (R)
    Fortran
    Univac 1108 -> Univac 1108
    691 statements: 96 identifiers

33. Martin Marietta Aerospace: W. B. Carson
    Event-driven automatic reconfiguration (R)
    Fortran, Assembler
    DEC PDP-11/70 -> DEC PDP-11/70

34. UK Coral 66 Team [England]: D. N. Shorter & K. Resander
    Process control: graphics (R)
    Coral 66
    DEC PDP-11/45 -> DEC PDP-11/45

35. Bureau of the Census: -
    Generalized mass storage sort: heavy I/O (R)
    Assembler

36. Lund Institute of Technology [Sweden]: -
    Process control with operator (model program)
    Pascal, Concurrent Pascal
    DEC LSI-11 -> DEC LSI-11

37. McDonnell Douglas Astronautics: -
    Real-time processing, Array processing, Fixed point arithmetic
    Assembler
    CDC Cyber -> RCA SCP 234
    200 statements: 350 identifiers

38. Air Force Communications Computer Programming Center: James E. Emmert
    Real-time communications

39. The Mitre Corporation: Maureen H. Cheheyl
    ACCAT Guard
    Gypsy
    DEC TOPS-20 -> -
    37 statements: 16 identifiers

40. DNACS, National Physical Laboratory [England]: Maurice Cox, Sven Hammarling
    Numerical software library (R)
    Algol 60, Fortran
    portable -> portable

41. TRW Corp.: H. Hart, J. Thompson
    Benchmark flight algorithms: mathematical
    Jovial J73/I3
    DEC PDP-10 -> DEC PDP-10
    1000 statements: 394 identifiers

42. General Dynamics: -
    Avionics: numbers, bits (R)
    Jovial J3b
    IBM 3033 -> M362-F2

43. General Dynamics: -
    Display generation
    Assembler
    Intel 8080 -> Intel 8080

44. General Dynamics: -
    Real time processing (R)
    PL/M
    MDS-80 -> Intel 8080. microprocessor
    19 statements: 8 identifiers

45. General Dynamics: -
    - (R)
    Jovial J3B
    IBM 370 -> Delco M362F

46. Grumman Aerospace, Software Systems Dept.: J.A. Garry
    Trajectory computation
    Fortran
    IBM 360 -> Honeywell 6060
    57 statements: 52 identifiers

47. Grumman Aerospace, Software Systems Dept.: J. Kmeicik
    Special-purpose data base manager
    Assembler
    Interdata 8/32 -> Interdata 8/32
    291 statements: 29 identifiers

48. Grumman Aerospace, Software Systems Dept.: R. Wellner
    Real-time flight control
    Assembler
    - -> Honeywell 5301
    27 statements: 39 identifiers

49. GTE Sylvania Inc.: Charlene Hayden
    Real time processing
    CMS-2
    IBM 370 -> AN/UYK-20
    62 identifiers

50. The Foxboro Co.: M. E. Gordon
    Model controller operating system (R)

51. The Foxboro Co.: N. B. Robinson
    Industrial controller (R)
    Assembler

52. Air Force AFAL/AAT: Alfred J. Scarpelli
    Avionics local executive
    Jovial J73/I
    DEC PDP-10 -> AN/AYK-15
    533 statements: 285 identifiers

53. Texas Instruments: -
    Benchmarks: GPS, image processing
    Assembler, Pascal (MicroTIP)
    - -> TI 9900
    2000 statements

54. Burroughs Corp.: Jane Powanda
    Real-time operating system (R)
    Assembler (CAL)
    Burroughs 774 -> -
    200 statements

55. Army USACEEIA: Leon E. Dixon
    Message annotator
    Assembler
    AS/5 (IBM 370) -> AS/5
    300 statements: 200 identifiers

56. AAI Corporation: R.A. Duff, R.L. McGarvey
    Disk I/O (R)
    Pascal
    Perkin Elmer 7/32 -> Perkin Elmer 7/32
    300 statements: 305 identifiers

57. Technology Service Corp.: D. Hollingworth, J. Lloyd
    Array processor interface (R)
    - -> Goodyear Staran
    166 statements

58. Rockwell International: John L. Whited
    Communications operating system
    Assembler
    Data General Eclipse -> ROLM 1602

59. Georgia Institute of Technology: Fred Cox
    Fire control (R)
    Assembler, Fortran (Flecs)
    Data General Eclipse S/130 -> ROLM 1602A

60. (Obsolete)

61. Georgia Institute of Technology: Lawrence J. Gallaher
    Tracking radar
    Fortran (Flecs)
    Data General Nova 3 -> Data General Nova 3
    2035 statements: 240 identifiers

62. Honeywell: P.D. Stachour, F.G. Christiansen
    Character processing
    95 statements

63. Honeywell: P.D. Stachour, F.G. Christiansen
    User command (R)
    PL/I
    Honeywell level 68 -> Honeywell level 68
    38 statements: 23 identifiers

64. Systems Consultants Inc.: -
    Command processor
    CMS-2Y
    AN/UYK-7 -> AN/UYK-7

65. Systems Consultants Inc.: -
    Document indexer
    Fortran
    HP-3000 -> HP-3000

66. Honeywell Avionics: J. M. Kamrad
    On-board real-time control system (R)
    Assembler
    Intel 8085 -> Intel 8085
    165 statements

67. Honeywell Avionics: C. Yandow
    Flight executive (R)
    Assembler

68. Honeywell Avionics: J. M. Holschbach
    Real-time radar detection
    Assembler
    Intel 8085 -> Intel 8085
    190 statements: 89 identifiers

69. IABG [Germany]: Peter Burkinshaw
    Graph theory: Hamiltonian path finding
    Pascal
    CDC Cyber -> CDC Cyber
    93 statements: 21 identifiers

70. HQ SAC/ADSW: Lt. Thomas J. Croak
    Mathematical calculations (R)
    Assembler
    Univac 1100/42 -> Univac 1100/42
    16 statements: 6 identifiers

71. -: -
    Conditional testing, bit manipulation (R)
    CMS-2Y
    AN/UYK-7 -> AN/UYK-7
    159 statements: 37 identifiers

72. Perkin-Elmer Data Systems Group: -
    Interactive transaction processing system (R)
    Assembler
    Perkin-Elmer 7/32 -> Perkin-Elmer 7/32

73. Hughes Aircraft Company: J. Whitaker
    Real-time fire control system
    CMS-2Y
    AN/UYK-7 -> AN/UYK-7

74. British Airways [England]: -
    Record I/O package
    Neliac, Assembler
    DEC PDP-10 -> DEC PDP-10
    29 statements: 66 identifiers

75. HQ SAC/ADOS: Lt. Steven C. Bush
    Database manager (R)
    Fortran, Assembler
    IBM 360/85 -> IBM 360/85
    40 statements: 9 identifiers

76. Air Force ASD/ADSD: Lt. Steven K. Rogers
    Real-time EMG Analyzer: Cross-assembler (R)
    Fortran, Assembler
    - -> Intel 8085

77. Pacific Missile Test Center:
    Diverse flight software
    Metaplan
    Xerox 560 -> CDC 5400B
    24 statements

78. Naval Avionics Center: -
    Navigation Computation (R)
    Assembler
    Honeywell 635 -> AYK-14

79. Naval Avionics Center: -
    Dual processor interface test
    Assembler
    Honeywell 635 -> AYK-14

80. Naval Electronic Systems Command: -
    Communications module (R)
    Assembler
    UYK-7 -> UYK-20

81. Dept. of the Navy: -
    Mathematical computation, comparison and interpolation (R)
    Fortran IV
    SEL 32/55 -> SEL 32/55
    383 statements: 149 identifiers

82. Dept. of the Navy: Robert Zile
    Real time mathematical computation (R)
    Fortran
    AN/UYK-7 -> AN/UYK-7
    925 statements: 150 identifiers

83. Dept. of the Navy: -
    Mathematical computation (R)
    Fortran, CMS-2, Assembler
    - -> IBM 4Pi
    200 statements: 90 identifiers

84. Dept. of the Navy: -
    Mathematical computations (R)
    F[illegible]n, SPL/I
    ( [illegible] 500 -> CDC 6600, ASP
    5[illegible] statements: 154 identifiers

85. Naval Surface Weapons Center: Marc Hubbard
    Real time processing, fixed point arithmetic
    Assembler
    IBM 370 -> UKK-20
    191 statements: 39 identifiers

86. -: -
    Generic menu package (R)

87. Sanders Associates: Robert E. Rice
    FFT, search, sort (R)
    Fortran, Pascal, Mortran (Fortran), Ratfor
    DEC Vax 11/780 -> -
    76 statements: 26 identifiers

88. McDonnell Douglas: J.J. Cobble
Autopilot, data base, message handler
Assembler

89. Naval Research Laboratory: M. Cronin, J. Gannon, D. Weiss
Software engineering tests (R)

TER #1

4.1 Comparing the particular program I coded, there would be no significant difference between Modula and Ada, but for further developments I think Ada would be faster because separate modules could be used without re-editing.

4.3 It is more voluminous and somewhat repetitive, but this is more often a help than a hindrance. Supplemented by a symbol table and cross reference list it would be very much more readable.

TER #2

2.6 ...Ada version is more portable.

4.1 Ada development should be fastest due to ability of compiler to flag illegal or unorthodox coding. It is assumed good debugging tools will support the Ada development system.

4.4 Very little would be gained by using Ada except for portability.

4.5 All depends on adequate:
- training
- documentation
- availability of inhouse expertise
- Ada compiler (efficient code produced, super error-detection)
- support software (library interfaces, TEXT_IO, STANDARD, etc.)
- a friendly development environment system
- development of a set of Ada programming practices

TER #3

3.1 The concept of package was easy to understand but difficult to use, Reference Manual didn't mention where to write to begin with.

4.2 In CPL-8 (same as P/I), implicit type conversion is the most error prone feature.

4.4 As for language feature, CPL-8 is powerful enough.

TER #4

3.2 Features such as access types, private types, and overloading did not seem to lend themselves to the project chosen. Apparently if the designer does not have Ada in mind, the design does not lend itself to many of the new ideas.

3.3 A conspicuously absent element is the hierarchical element.

3.4 Ada seemed to be able to accommodate any situation that arose in this application.

4.1 It would take longer to develop a debugged program in Ada than e.g. in IBM's PDMS (Program Development and Maintenance System.) The author of PDMS decided to keep the language simple and impose part of the methodology through the environment e.g. 50 lines/code unit, automatic indenting. No language can guarantee quality programming. There must be training sessions and a strict methodology agreed upon by the programing team members.

4.2 I feel that strong typing is important and even facilitates coding.

4.3 As I mentioned above, I think that Ada could be considerably more readable. The syntax is many times obscure.

4.5 The main problem that would arise is that the personnel would have to be sold on the advantages of Ada over e.g. FORTRAN.

4.9 PACKAGE, RESTRICTED, and TASK are particularly brilliant.

<u>TER #5</u>

2.7 Yes. The Ada constructs proved much more adaptable than Cobol.

3.5 Since the Ada design was so much more compact and simple than the Cobol program I decided to implement a more sequential method.

3.6 The program seems very clear and efficient.

<u>TER #6</u>

2.2 The main executable portions of the new design are much easier to read than the original because they are much shorter, with portions of the original code relegated to subroutines.

3.6 Recoding in Ada resulted in a small improvement in storage efficiency for this example.

4.3 Reorganization with conditional statements improves the readability.

4.4 Using Ada probably would have the same effect as any other modern language, like Pascal or PL/1.

<u>TER #7</u>

0 I believe that our experience is particularly important because we are comparing Ada with a powerful, modern language based on Algol-68. We love Ada types which are much superior to our own, but we find that Ada's rigid statement structure prevents us from writing natural solutions to our problems.

0 With this one exception we are on the whole pleased with Ada.

3.3.2.2 Little scope was found for derived types or subtypes...

3.3.2.3 Extensive redesign of some existing interfaces was required. to circumvent the lack of rows of procedures, which, though acceptable in the case of LMG, would have presented an unacceptable overhead in the case of a program such as the S3 compiler
We fear that UNSAFE_PROGRAMMING will be a pr [illegible]nent feature in programs which must interface with non-ADA [illegible]e, or which must be very compact or efficient.

3.3.2.4 Generic packages are a powerful facility and we used them to provide procedure parameters to a tree walking package. however, we feel that a run time parameterisation is necessary (LLR S3/1).

3.3.3.1 ADA compares well with S3 in terms of readability of source code. It scores heavily by the introduction of enumeration types, which are a major aid in self documentation. Similarly, the exception handling facilities encourage readability by separating error handling from the main path. The use of default parameters is a further asset....

Some of the syntactic features of ADA hinder readability, however. These are, notably the lack of conditional expressions and the absence of compound statements.... The verbosity of ADA further hinders readability, in particular the complexity of array slicing. The syntax of a block...discourages the declaration of data near to the point where it is used. The syntax of aggregates... is preferred to the S3 equivalent.

3.3.3.2 Ada, on the whole, is a more verbose language than S3, although in some areas it improves on it. ...some features of Ada may actively encourage programing errors and so reduce programmer productivity:

(1). The significance of break characters in identifiers;
(2). Need to introduce blocks to introduce new local data items;
(3). conditional expressions not being permitted;
(4). respecifying type declarations in order to add a representation specification.

3.3.3.3 Ada code may well prove to be more maintainable than equivalent S3 Code as a result of it being more self documenting.

3.3.3.4 Ada permits more elaborate run tie checking than does S3.... Ada training courses should emphasize correct use of types.

3.3.3.4 ...our experience with S3 is that [reference variables] are a valuable tool in the hands of the experienced programmer.

3.3.3.5 The separate compilation system is more versatile than that of S3....

3.3.3.6 Ada looks to be excellent in engendering portable programs.

3.3.3.7 The exception handling facility of Ada provides a convenient, high level way of handling errors detected within nested procedure calls.

3.3.3.10 Ada's provision of this facility [Input-Output] is a considerable advance on S3.

3.3.3.11 Both Ada and S3 are suitable languages for programming in the large, with the modular aspects of Ada being further enhanced by nesting packages and having visible and private parts. ...the proposed compilation system lends itself to large scale software construction systems, rather than one-one-off, small scale programming. Ada is not very easy, either to learn or write, particularly in that it introduces several features foreign to Algol-68-like languages.... An S3 style of programming based on nesting of constructs has evolved, and an Ada style does not grow easily out of this. The emphasis on statements rather than expressions seems retrograde, and the very strong typing will prove irksome to systems programmers. Generics in particular are difficult to grasp....

3.1 We were somewhat confused since strings are messy compared with references to arrays as used in S3.

4.1 Ada code will probably take longer to write than equivalent S3 code because of the verbosity of the language. We expect that the time taken to debug a program will be less as a result of the extensive run time checking, and because [many] potential run time bugs are found during compilation....

4.6 Ada is likely to appear in a better light when a software system is designed with the knowledge that Ada will be used as the implementation language.

4.10 Ada has derived many useful features from its PASCAL background, particularly its excellent typing. It seems a pity that many of the useful features of Algol 68, particularly its expression structure and use of reference types, have not been similarly incorporated in Ada.

<u>TER #8</u>

2.3 Use of packages helped tremendously to define the interfaces and Data Base Types. Enumerated types were also beneficial. Type definitions were mnemonic and readable.

3.1 ...I kept omitting ENDIF for IF statements of form: if condition then statement (single) probably a common mistake.

3.6 The constructs allowed for proper expression of my program. No certain constructs are disturbing to me.

4.1 Development time would usually be about the same, if not shorter, than most languages. Compiler will catch many mistakes. Learning Ada may be longer than usual.

4.3 Source code in Ada is as readable as other languages.

4.4 production would increase since most programming is done in assembly language. Program maintenance would be more easily performed and transition to another person/agency would be smoother. Because of compiler, many mistakes will be caught at compile time and not during executions.

4.9 [Tasking] is easy to understand and use. Private types and package structures are also well designed and should be unchanged.

<u>TER #9</u>

0 The working part of the program was enormously shorter, and for the first time it was readable in a form familiar to one versed in the science.

<u>TER #10</u>

3.4 Q: Were there any interactions... that caused you difficulties?

A: No

4.1 No
4.2 ...[type checking] will facilitate debugging and increase the reliability of the program.
4.3 Yes! The block and statement structure facilitate readability.
4.6 Expect to encounter the same problems one always encounters with new tools.
4.7 No redundant features which should be deleted were detected...
4.8 ...not sure that changes are required.

TER #11

3.1 I had no difficulties to understand the different features in the language.
4.1 It will take shorter time to program in Ada than in other languages.
4.3 The code written in Ada is generally more readable than programs written in most other languages.

TER #12

4.4 The current project uses a mixture of several languages. Using only Ada means that a maintenance programmer only has to learn one language.
4.6 I would strongly recommend it because of the structured programming techniques that Ada encourages.

TER #13

0 Our major conclusions are that Ada is suitable for both embedded computer software and support software. We are concerned, however, that the high complexity of the language and its restrictive type checking may result in inconsistent and inefficient use of the language and higher than anticipated life cycle costs.
2.3.1.2 For numerical processing Ada deserves praise as in its ability to define precision by specifying the number of digits, the range, or the delta. In comparison with other languages, Ada is rated satisfactory to superior...
For realtime executive support Ada is inadequate. It must be modified to recognize that interrupts must be processed preemptively.
2.3.2.1.1 ...Ada would be quite sufficient in these areas, [separate compilation etc.] and offers decided advantages over the corresponding PL/M constructs.
2.3.2.2 In many respects Ada would be a more suitable language... than PL/M, even though PL/M seems to be specially oriented to this kind of application.
2.3.2.3 ...Ada offers excellent facilities in the area of program

variable declarations.

2.3.3.3 Ada does have good data constructs with powerful IF and CASE statements.

2.3.5.3 The Ada language may be acceptable for the development of [operating system] due to the flexibility of the language.

2.3.6.3 The visibility rules only make the language more difficult. Usually, [illegible]en data only confuses maintenance programmers without stopping anyone intent on violating the system security.

2.3.7.2 ...Ada as it stands now would not be suitable... due to the way in which it services interrupts. If the requested change were made... then the determining factor for Ada's usefulness...would be Ada's efficiency....
...while the current version of Ada is not useful for this kind of application, future versions could be changed to be suitable.

2.4 ...Ada is apparently suitable for the generation of support software of software tools.

2.5.1.1 The current Ada pointers are unuseable for static data structures.

2.5.1.2 One of the features lacking in Ada is the adequate control of dynamic storage allocation.

0 In general, Ada is suitable for both embedded computer programs and support software. However, the design appears to favor the latter.

<u>TER #14</u>

2.6 ...I didn't know enough about Ada when I started. Has I know then what I know now, I would have never tried to fit into an existing system, rather, I would have redesigned the system from scratch.

2.7 The problems were all with pointers and access types.

3.2 none in particular

4.1 Comparing Ada to PL/I and to Assembler will probably produce an "about equal" comparison. I think that even if it takes longer to write an Ada program (and I am not certain that it does), the costs and times for maintenance will be lower. Certainly the readability and correctness should be better than when using current languages.

4.4.A Ada requires better planning, interface specification, and documentation.

4.4.D The use of a totally new language is an excellent vehicle for helping to remove old habits such as leaping into code with out thinking, use of non-structured programming, etc.

<u>TER #15</u>

2.2 Ada would provide better controls over the use of data and hence better unit level design in quite a few areas of the system....

4.1 Application logic should be shorter.... Longer implementation for system design and system interfaces.
4.2 Strong data typing appears to be highly overrated as a technique significantly avoiding programming errors.
4.3 Ada can be used to create highly readable code and does much to discourage poor practices.
3 The ada development is in general much enhanced by clear and precise use of technical terminology. However some unusual usages seem to have crept in.

TER #16

2.2 Though no redesign was done the code was better than before because it was easier to read.
2.5 Everything necessary for array handling is available in Ada, but some additional capabilities would be helpful.
2.6 By putting the procedures and data in packages there was a better feel for the relationship between variables in the program. Having a feature like packages encourages one to do this.
3.1.1 Problems arise trying to figure out where the variable should be declared.
3.1.4 ...in discussions with people working on other Ada programs I found access types very confusing.
3.5 More experience in the use of overloading operators is necessary before a decision as to whether or not to use it can be made.
In general the code was clean and a good compiler would probably generate efficient object code from it.
4.1 It would seem that programming/debugging in Ada would take a little less time.
4.2.2 ...it would appear that type checking is a very helpful aid in detecting errors.
4.3.1 The Ada code is more readable....
4.3.2 The syntax of comments made the code less readable.
4.4.1 Better data organization and interaction through the use of packages.
4.4.2 Better program organization through the use of structured constructs.
4.4.3 Less execution time errors because of Ada's type checking.
4.4.4 Cleaner exception handling because of Ada's exception mechanism.
4.5.4 For a project that could be written all in Ada and did not need much low level support... Ada would be a good choice.... For signal processing applications, I am not convinced that any high-level language is suitable.

TER #17

3.1 - 3.6 The only difficulty (of other than a minor nature) encountered in this Ada implementation was in determining the array [limit in records]....
Although the Ada data structure is definitely superior to the FORTRAN implementation, there is concern about... representation
4.2 [I] believe that many errors not normally detectable until execution will be caught by the language translator.
4.3 The Ada code resembles the design language so closely that well-written programs should require fewer comments than in non-structured languages.
4.6 Assuming... [a good] compiler... I would welcome the use of Ada on my next... project.
4.9 The textual structure and data typing facilities of Ada are features that should definitely remain in the language.

TER #18

2.2 The redesign is far better than the original. The program is shorter in length..., modular in structure, and easier to read. In addition, there are fewer control flags....

TER #19

2.6 The Ada code will be better structured. Related objects are grouped together in packages. Readability is enhanced.
3.6 I believe that clear expression of the program was possible. Ada does promote readable code....
4.1 I think developing a new program will take longer in Ada than in a language with less typing. The difference would be in the rigorous definition and specification of types, initializing packed objects by aggregates instead of hex, and the specification of subprogram parameter lists. However, I believe that the type checking, etc. performed by an Ada compiler will catch many of the routine errors customarily made in programming.
4.3 I believe an Ada program can be self documenting....
4.4 I do believe Ada to be a very useable language, unlike some other HOLs....

TER # 20

2.2 New design is better than the original in that it is far more robust and maintainable. The new design is not worse than the original in any significant way.
4.1 It will definitely take longer to develop a program.... However, the resulting Ada programs will be far better.... ...the extended amount of time required to program in Ada [is] a comment upon the haphazard manner in which present day HOLs are

used to develop DoD software....

4.4 An application program would stand more of a chance of being correct, maintainable, and modifiable.

4.7 The features of Ada are very interdependent and excisions cannot be made without a great deal of care. I would like to see something done about excessive language complexity... [but] it is not something decided by taking a vote.

TER #21

4.1 For this particular problem, developing a debugged program would probably be easier and faster....

4.3 Assumptions of problem solution are more explicit. However, the code is less readable [than that for our special-purpose compiler]

4.4 ...the main advantage Ada provides is a set of well-thought-out concepts like "tasks", "rendezvous", and "entries" for dealing with concurrency. These make designing and implementing a correct problem solution far easier than with ad hoc multitasking facilities.

4.7 The features of Ada are very interdependent and excisions or changes cannot be made without a great deal of care. ...these changes [detailed list precedes] might compromise language usability.

TER #22

4.1 Given the lee side of the Ada learning curve, I believe Ada programs will not be significantly more difficult to design, code or debug. Ada is a very concise language, which should allow a programmer with experience in it to implement a program in it with no more difficulty than before.

TER #23

2.2 The new design is definitely more readable and maintainable. Each data structure and operation is clear as to its purpose.

TER #24

2.6 In Ada it was easier to "think Structured" limited number of reasonable constraints - vs PL/I this assisted me in coding a structured program.

4.1 After a little familiarization and practice Ada coding did not take as long....

4.4 Ada appears to be an excellent language to teach Basic programming fundamentals in. It is readable, fairly easy to use, and extremely easy to transform algorithms from a Design Language (e.g., PDL) into.

TER #25

4.1 I believe that developing a debugged program in Ada will take longer....
   (1) Ada is larger;
   (2) Ada is not considerably more powerful;
       I would expect Ada would be in a much more favorable footing against older languages such as Fortran or PL/1.
   (3) The ability to use processes, classes and monitors as true data types in Concurrent Pascal is not quite matched in Ada.
4.3 I believe my Ada reprogramming of the SOLO operating system is less readable than the original. The key reason for this is the inability to use package names as parameters of other packages. Otherwise, it would have been roughly as readable as the original.

TER #26

4.1 Developing a debugged program may very well take longer in Ada than in Fortran. The Ada requirements for explicitly typing every variable will lead to better programs that are more reliable and probably easier to maintain over the total life cycle. However, the detailed specification of each variable type does lead to additional lines of code that have to be developed, debugged and integrated.
4.2 The type checking of variables is in my opinion a key feature of Ada....
4.3 ...Ada did not seem to offer any more readability than Fortran However, if this was coded initially in Ada..., it is conceivable that highly readable programs would result.
4.4 A universal (real time) language... has very obvious benefits and this would apply to all our projects.

TER #27

4.1 ...I feel that developing programs in Ada will take no more time than developing programs in any other high order language....
4.3 I felt that Ada is more readable than FORTRAN or assembly language, but no more readable than Pascal or ALGOL.
4.9 The definition of tasks in the language allows the user to more easily see the tasks.

TER #28

2.2 It is far more understandable than the ALGOL, SUE, or JOVIAL versions.

3.1 My application was well suited to Ada and therefore I had no difficulty in the use of Ada....

3.6 I do feel that the program is expressed clearly and yet will permit a good compiler to generate efficient code.

4.1 I believe developing a debugged program in Ada will take less time than in any other language within my experience, namely JOVIAL, ALGOL, CMS-2, Assembler, LISP, Sue, PASCAL, Modula, and some others that are unfamiliar to the world. This is for two principal reasons.
   . A more understandable program can be written
   . A successful compilation means testing is much further along.

4.3 My Ada program is more readable than the 4 [illegible] vious versions of this program. ...the Ada implementation more nearly reflects my design concept.

4.4 The more people that work on a project the more the value of Ada becomes apparent.

4.5 If Ada were exclusively used in my application area, no special problems would result.

4.6 I look forward with pleasure to the use of Ada for my next embedded computer system. There will be many fewer problems.

4.9 I like the overall character and consistency of the language and so I would not like to see it changed in any significant way.

4.10 Even before Ada compilers are available, the language can be used as a design tool. It is a far superior vehicle than the current crop of PDL's/pseudo codes.

<u>TER #29</u>

2.2 [Two] redesigns... were judged to be major improvements... for all the right reasons - greater clarity, ease of modification and testing, top-down design, etc.
The solutions to the other three areas... were less satisfying.

4.1 Learning to use Ada properly seems to take considerable time, but it is a substantial improvement over other languages.

4.2 The team believes that in general strong typing promotes good design and facilitates testing.

4.3 Team members had somewhat conflicting opinions to just how much more readable Ada code is.... All agreed that Ada was at least better, and that programs written in Ada would be easier to debug and test.

4.4 Ada.. increases the probability of writing correct programs.

4.6 Everyone but participant p6 definitely agreed that if a good Ada compiler were available, they would program their next project in Ada.

4.10 Ada seems to be generally superior, and subsequent designs should require less effort.

<u>TER #30</u>

2.6 The original RTL was very well designed and Ada merely allows this design to be mapped into slightly better code....

4.1.1 Algol 68 and S3:
Overall, I think that development would take longer in Ada due to the rigorous requirements in the design and coding. Algol 68 programs are often coded somewhat "loosely".

4.1.2 RTL/2 and Algol 60:
These languages are significantly more restricted and secure than Algol 68 and consequently, I think that Ada development time will be similar to these languages.

4.1.3 Fortran and Coral 66:
The insecurity of these languages causes excessive debugging time. Ada should be much faster.

4.1.4 Assembler (various):
Ada development time for a debugged program should be immeasurably better.

4.2 ...Ada actively encourages the production of "correct" coding.

4.3 Ada programs do not seem to be any more readable than RTL/2 and Algol 68 programs. There are several problem areas: Separate compilation; an [over-] abundance of block declarations [for] local data; general verbosity.

4.4.1 Ada would ensure no incompatibilities in interfaces....

4.4.2 All systems are notoriously non-portable, even if written in a portable language. Ada should cause a complete reversal in this trend.

4.5.2 lack of references [to] static [objects] is a fundamental, and serious, problem.

<u>TER #31</u>

2.1 The concept of modules permitted the definition of constructs to be localized. [Previously,] each exported construct was available throughout the entire program.

2.2 The generality of the module structure permits the hiding of unnecessary details from the interface specifications making the overall structure easier to understand. The distinction between the logical properties of an object and its representation were of help in this area.

2.6 ...the ability to perform information hiding... greatly facilitates the reading of the program text.

4.1 ...programs will take less time to produce in debugged form... [because of the] complete control over interfaces....

4.3 The code is more readable mainly due to the ability of defining types with well defined logical properties.

4.5 The diversity of features offered by the language make it difficult to make a choice in certain cases.

4.6 Strongly in favor. ...the features of Ada... seem to fit the direct needs of the embedded computer software group.

TER #32

2.2 Worse. Interfacing between units was complex. More complex and error prone....
3.1 No real difficulty using or applying features correctly once they were understood.
3.4 ...the many restrictions and qualifications, and the lack of a pattern or intuitive parallelism for them creates a burden for the programmer....
4.1 Ada is more complex than any other language with which I am familiar.
4.2 The greater specification of data and the need for explicit conversion of types is a nuisance. [It] added unexpected conversion errors; fixed point arithmetic was a major problem.
4.3 No
4.4 None
4.5 Poorer quality of software development due to complexity of Ada.

4.6 ...Ada is complex, awkward and imposes unnecessary burdens upon the programmer.
4.7 The deletion of redundant features would probably impair the usability of the language for many programmers.
4.10 It is difficult to identify a small basic subset within Ada to use as a starting point for learning the language and for beginning to program in it. Ada seems to be made up of a set of sub-languages that are partially disjoint, rather than being concentric, as is the case with other programming languages.

TER #33

2.2 It is better because it more clearly expresses my intent.
2.5 No problem - infact, it was a relief to be able to [translate to Ada]
0 ...Ada was found to provide the basis for much greater ease and clarity of formulation and communication of concepts....
...Ada's expressiveness will contribute substantially to software maintainability....

TER #34

4.1 ...we doubt that debug time will be significantly different....
4.4 an increase in reliability; possibility of re-using code; decrease in dependence on proprietary software.
4.6 Apprehensive at the commercial risks of depending on as yet unproven technical features.

TER #35

0 Overall, our impression of the language was a very positive one.
3.3 There were two major omissions for us, variable length strings and formatted I/O.
4.1 ...Ada would be yet better. It is so much better than languages such as Fortran or assembler language as to make the comparison laughable.
4.2 ...a very useful feature.
4.3 Yes, for a number of reasons including enumeration types, ability to overload procedures and operators, ability to have functions returning any type, and etc.
4.6 Love it.

TER #36

2.1 What can be done [currently] can be done in Ada with only minor modification.
2.2 It is possible to describe the same structure in a shorter, nicer and more readable way in Ada.

TER #37

2.2 The Ada is more readable than the assembly or the flow charts.
4.1 Development of a debugged program would be faster than in assembly language, and given that the interrupt priority problem is solved, faster than in Fortran.
4.2 ...Ada seems to discriminate by requiring the fixed point programmer to live with stronger typing than the floating point or exact integer programmer must endure.
4.3 ...the rest of the language is almost obvious without explanation, and represents a viable presentation language as is.
4.4 ...it would probably be easier to teach how to read Ada than to teach the structure and format of a sizable application program.

TER #38

4.1 ...what Ada forces you to do is to spend more time in the analyses and design phases.
4.3 It looks to be as readable as most other high order languages.
4.4 The concepts of information binding, abstract data type, would be used extensively — thus allowing better defined partitioning of work.

TER #39

3.2 Mapping an existing concurrency mechanism into Ada is difficult.

4.1 Given a decent programming environment, I wouldn't expect program development to take any longer in Ada than in another high order language.

4.2 The Ada code, particularly if formatted properly, is very readable compared to that of other languages.

4.4 The use of Ada would benefit our project in two major ways. First, Ada has several [useful] features. Second, Ada is expected to be a military standard. Our experimental work in computer security will thus be more easily applied to real programs.

4.5 Verifiability may turn out to be a problem for security work although it may be possible to generate a verifiable subset....

<u>TER #40</u>

4.3 More readable than Fortran!! Not necessarily more readable than Algol 60, but this may be a familiarization problem. On the whole Ada gives a good algorithmic description.

4.4 Portability of numerical software across machine ranges. Debugging and error detection should be much easier in Ada.

<u>TER #41</u>

4.1 ...no significant difference is expected.

4.2 strong type checking is important not only fr detecting programming errors but for enhancing program readability.

4.3 However, Ada code is understandable by anyone who has knowledge (limited) of Ada constructs or good knowledge of any other HOL.

4.6 Other than questioning compiler availability,, bullish

4.9 As a general rule, the program and control structures of Ada are good.

<u>TER #42</u>

3.6 It did seem possible to write clear yet efficiently compilable code except in isolated instances....

<u>TER #43</u>

2.5 Ada is not the kind of language you can read a manual on and then easily begin generating code. It is too complex for that.

3.6 I do not feel that Ada permits very clear expression of a program.

4.1 Ada is unnecessarily complex and restrictive. For many microprocessor applications I feel that Assembler or Fortran would be better.
4.3 The multitude of program types seems artificial.
4.6 I would not like it at all because I feel that Ada in its present form, is not suitable for [my] microprocessor applications.

TER #44

3.1 Many constructs seem to also have a lot of extra keywords.
4.3 ...program readability seems to be significantly improved over other languages.... Subprogram specifications are more easily understood in Ada in some cases than in other languages.
4.6 As long as the project did not require a great deal of bit pattern manipulation... doing a project in Ada would be favored.

TER #45

1.4 The number of statements (and types) should be much the same. The area in which the Ada will require more is in the declarations and the amount of visibility allowed. Little experience exists with the complex Ada scoping and visibility rules. This is an area which may add to the maintenance effort for large programs.

TER #46

2.2 The new code is more readable at the executable level and does not require a long introductory prologue primarily because of the Ada declarative requirements.
4.4 Language is relatively easy to learn for the type of language features required; large common data blocks will be simpler and less error-prone with control/facility of packages and "use" statements.
4.6 ...I would welcome the opportunity to use Ada in an embedded software project.
4.9 Typing and packaging features: the readability and traceability that results is worth the additional efforts.

TER #47

2.2 The Ada version would be easier to maintain since the code is more descriptive.
4.9 Ada forces a more structured software design, first on an overall program level by means of the specification section and then in a more detailed body section.

TER #48

4.1 ...I feel that the time to debug a program will be less.
4.3 The code is definitely more readable. The program logic was much easier to follow when it came to concurrent tasks due to statements which facilitate such....
4.6 I would look forward to doing my next embedded computer project in Ada. Although it requires more writing, I feel in the long run I'd get the job done sooner.
4.9 Although somewhat tedious, I would not like to see the typing of data changed.

TER #49

2.6 The design was not significantly better; I primarily followed original design which was structured and mapped easily into Ada.
2.7 Planned to follow original design; in using Ada I found simpler mechanism and constructs in some cases, e.g. initialization.
4.1 ...the numerous programming techniques available with Ada could make program management difficult.
4.3 The Ada program was definitely more readable. Ada is definitely the most readable language I have ever encountered.

TER #50

3.6 I have great reservations about the ability to optimize Ada code as the language is currently defined.
4.1 In general, I think that Ada facilitates the program development process. It is difficult to make predictions....
4.3 Generally, an Ada program in not very readable: declarations must be present bottom-up; begin-loop-select demands too much nesting. Separating logical declarations from physical representation specification is awkward. A heavy language (no abbreviations).

TER #51

2.2 The new design offers better protection of the data objects because of the strong typing used.
2.5 Two major areas of difficulty had to do with handling exceptional conditions and with choosing the best data representations.
3.1 Surprisingly, text layout was a problem. Ada does not allow presentation in top-down fashion [of structures], but in fact requires programmers to perform a topological sort so that no forward references occur. Not only is this a nuisance to the

writer, but I don't see it as useful for the reader.

3.3 The critical feature missing in Ada for our application was a well-defined scheduling policy and the lack of facilities for introducing one.

4.3 In some ways it is less readable. The procedural code is probably as good in Ada as in other high level languages. Non-procedural code, such as type declarations, has a very severe restriction placed on its presentation. The no-forward-reference rule enforces the bottom-up order.

4.4 The possibility of designing all interfaces and compiling them independently of implementations will be a plus.

4.6 I would not be reluctant to use Ada on production programs. I think it is, in the large, a well-constructed language.

<u>TER #52</u>

3.2.1 Access types could have been used to greatly improve access performance but they are not capable of denoting static variables.

4.4 The Ada source code would be self documenting and much easier to read and follow. A maintenance programmer would have a greater understanding of the program with less effort.
Ada promotes top-down structured programming.
The job of transporting programs from one machine to another would be easier.
Strong typing would prevent subtle type errors.... packages will prevent procedures from accessing and modifying data that they should not have access to.

4.6 Ada is a very capable language. It permits good structured code and the strong typing helps maintain data integrity. AFAL would like to use Ada in future projects. However, inefficiencies in the language may force the continued use of JOVIAL for real time applications unless the problems are resolved.
The greatest concern is with access types. The basic concept is excellent but the associated problems make them of limited value for AFAL applications.

<u>TER #53</u>

4.0 Overall impressions of the Ada language are very favorable. The package concept is perhaps the cleanest solution to date for the development of general purpose library routines.
There was no major difficulty in learning to use a convenient subset of Ada. The only significant problems encountered were in the tasking facilities.
Ada is an extremely verbose language.

TER #54

2.2 Forcing the use of Ada and thereby the interfaces supported by the Ada runtime system, contributes to making an applications program more manageable.
2.7 Originally some of the data was represented by enumeration types, but due to the limited number of operations that could be performed with enumeration type data, record and array structures were chosen instead.
4.1 Ada may have an advantage in the debugging area because of its readability and because most programs written in it tend to be structured so that it is not difficult to follow the path of data through a program.
4.4 The greatest advantage accrued from use of Ada on a project would probably be program maintainability.
4.9 The data structures, program structure and separate compilation facility in Ada are its prime assets as programming tools.

TER #55

4.9 I particularly like the ability to restrict visibility in Ada programs, because I have worked on projects where this feature would have prevented problems caused by multiple programmers making multiple uses of a particular data field.

TER #56

2.2 The new design is more portable because the Pascal version had to use non standard I/O procedures. The new design is more efficient thanks to exception handling for errors and exits in the middle of loops. The new design is easier to understand and uses shared packages.
4.1 Although ada is slightly more verbose than Pascal, and would therefore require more original coding work, it would take less time to develop a debugged Ada program. This is be because Ada is easier to understand, and it has more safety features.
4.3 Ada is much more readable than other languages.

TER #57

4.0 Our impression of Ada was that it was an exceedingly complex language. We now feel that although it is complex, Ada is not significantly more so than several other HOLs.

TER #58

0 ...with the appropriate change [interrupts] to meet real-time requirements, we given Ada an "A"....
One can hardly imagine a better conceptual modeling tool than Ada tasking, but the mapping into implementation seems latent with difficulties.

4.3 Another emphatic yes, primarily because it reads more like English text than a program language.

4.10 We like Ada!

<u>TER #61</u>

4.4 I believe that using ada will speed up project completion and reduce costs.

<u>TER #62</u>

4.1 I would say that a debugged Ada would take less time to develop because type-checking would ensure clean interfaces and avoid type mixing (producing garbage).

<u>TER #63</u>

2.2 New design considered conceptually clearer because of fewer interfaces (subroutines) where actions take place remotely.

4.3 the standard package with names for all character makes programs of this type more portable....

4.4 Dimension analysis problems caught at compile-time.

<u>TER #64</u>

2.5 In general the design mapped easily. The areas of trouble were machine dependent.

3.1 Finding the best way to divide a program up between packages, procedures, and tasks was the problem. This will require a different design outlook than we've used in the past.

4.1 Ada would simplify the implementation of debugged code to specification. Ada would allow testing better.

4.4 The parallel tasks of Ada would simplify the design of complex machine simulations. This was a problem in the current system.... The convolutions which were necessary for this were unbelievable. The recursive nature of Ada would allow easier implementation of some algorithms. We can obviously do the calculations non-recursively but not as neatly.

4.5 Ada should [help] by allowing us to design standard conventions at a high level in packages and then requiring the use of these definitions.

4.6 Ada would make life easier by allowing us to define the data and processing interfaces in the early conceptual design, and then implement the details within these constraints. Also the review of program design and style would be easier in Ada because of its structure and readability.

TER #66

3.3 There is no question that the redesign is better than the original in many ways — conceptual simplicity, readability, maintainability, modularity, machine independence, etc.
2.3 The task facility is a natural for this application despite the [priority] flaw. The use of packages and tasks provides an effective means of decoupling the different parts of the system. Abstract types, especially enumerated types, were useful in making the design more readable. Representation specifications were effectively employed to eliminate obscure bit manipulation and to make the whole design less machine dependent
2.4 The representation specifications enable the redesign to meet the storage requirements of the application.
3.1 The inability of the task facility to suspend and resume background (or lower priority) tasks to service higher priority tasks is viewed as a severe limitation....
3.6 My only concern for the optimization of any construct in my program is the optimization of tasks. The Hassi and Haberman [technique] shows promise....
4.1 Any increased time spent in coding an Ada program is more than offset by reducing problems due to type mismatches and procedure interface mismatches, common sources of problems. In addition, programs in Ada are definitely more readable....
4.3 The Ada program is more readable than the original program in every possible way! By using abstract types, especially enumerated types, the programmer can produce a program that is more descriptive and problem oriented.
4.4 The advantages of Ada are many and well known: machine independent; more readable and maintainable; more reliable; structured program design.
4.9 I am very enthusiastic about the whole language. In particular: the separation of the logical and physical properties of a program are supported by the language syntax; despite the flaw discovered in the task facility (see Response 3.1), the task facility provides an excellent conceptual approach....

TER #67

2.2 The new design is better than the original design because the Ada design is structured.... The use of Ada reduced the volume of source code.

2.3 The Ada tasking concept so naturally fitted the problem that it was impossible to conceive of any other approach; generics appeared natural.

4.1 The language so naturally supports tasking (logical), that my only complaint is that there is a bit of conceptual overloading of the term task.

4.2 In addition to detecting type errors at compile time, Ada forces an early focus on data as a general thing to be designed.

4.3 Ada is very readable because enumeration types fit so naturally into problems.

4.4 Advantages from using Ada are probably going to come from its readability initially, and portability in the longrun. Other areas where Ada appears valuable are more subtle. It seems reasonable to expect the quality of Ada code be better than other languages, the use of Assembly code to be reduced and to see more attention to data design.

4.5 The advantages may be offset by Ada's wholeness. A programmer must understand concurrent processing to understand tasking, macros to use generics and "type" to code at all.

TER #68

2.6 Using the Ada language resulted in a better design. The typing of each object gave more information about the data being used. Enumerated typing encouraged more descriptive assignments. The semantics of the language added to its readability.

4.4 Ada could be used very effectively as a design language. The language requires a strong, clear specification of all the objects being used. The readability surpasses most (all?) other languages.

TER #69

2.2 Worse, one NULL statement had to be introduced after a label in order to avoid assigning the label to the following FOR ... LOOP statement and then having to spuriously repeat the label in the END LOOP statement, which would have made the program more difficult to maintain.; but better in some respects, such as there are no anonymous END statements, there are many fewer BEGINs than in Pascal because Ada provides the END IF which is missing in Pascal.

3.6 Not as clearly or as concisely as in Pascal.

4.4 At present Ada is incapable of supporting the applications we are interested in....

TER #70

2.2 The new design is more straight forward and far more readable, possibly at the cost of memory space. It is hoped that the Ada version may be able to execute slightly faster.
2.3 The general goal of the Steelman requirements for more readable, more easily maintainable code. It is significant that in trying to understand the assembler version in order to redesign it, two relatively major flaws in the logic were discovered which could return erroneous data in some circumstances. It is felt that this would not have happened in Ada.
2.6 The problem in the past has been twisting the Program Development Language around to match the HOL.
4.0 ANY good structured design could be very easily implemented in Ada. On the other hand, a non-structured design would be harder to implement. The program evolves easily from the fresh design, however, it is not at all easy to transliterate from an unstructured existing HOL.

<u>TER #71</u>

1.0 It is felt that Ada should receive additional time for redesign and development which concentrates on Orderly Development of Reliable Software for Systems with embedded computers; Enhancing and clarifying Ada semantics; Simplifying Ada's syntax; Incorporating additional real-time capabilities.

3.0 Ada syntax is extensively verbose and in many cases grammatically incorrect.

3.1 Access types and allocators are very difficult to understand and use.
3.2 The IF-statement nested with itself and/or Loop-statements created unexpected difficult upon application. Many levels of nesting were virtually unintelligible. With each level, confusion increased. In following the logic of the program, one is never quite sure where one series of statements ends and another begins. Additional commenting was necessary to help alleviate these problems. Debugging is difficult at best.
4.1 Developing a debugged program with standard I/O requirements would take no longer nor shorter in Ada.
4.3 Ada produces less readable code than other high-level languages. Verbosity seems to be the keyword. Ada, in its attempt to provide more readable code, has gone to the other extreme. Additional keywords are attached to basic program constructs which are unnecessary. They convey no additional significant meaning that could not be picked up by the use of delimiters.

<u>TER #72</u>

4.1 I do not believe that using Ada instead of Concurrent Pascal will increase development time.

4.2 A number of programming errors were detected by the Ada Test Translator

4.3 The Ada source code is exceptionally clear. I have felt this way about every program I have written in Ada. However, (this) depends on the identifiers and constructs used....

4.4 Ada could be used to develop an entire system, eliminating the need for excessive machine code or assembly language insertions, and would therefore increase productivity and reduce the costs associated with program maintenance.

TER #73

2.6 I do not feel that knowledge of Ada helped me arrive at a better design but the language allowed me to represent the design better in program's implementation. the program structure became more closely tied to the functional requirements of the problem.

2.5 The translation of the mathematics in this example from a specification into the Ada language was extremely straight forward; However, using the Ada fixed point representation would probably be much more difficult.

2.1 ...packages, access types, and private types... should ease [our] programming job required....

2.6 A knowledge of Ada did not produce a better design as we had hoped. Perhaps if we could obtain a better understanding of how to use access types, we might find a way to successfully use them to improve the current design.

41. ...writing programs will take longer. There is more error checking, but the language consequently requires a great deal more writing effort....

TER #75

2.2 The new design is much easier to read. It has a more logical downward flow.

4.3 Just as Jean Ichbiah mentioned at the Ada Orientation, you can actually read an Ada program.

TER #76

4.0 After allowing for familiarization with Ada, I believe developing a debugged program similar to mine would take less time in Ada. This would be a result of the ability to detect programming errors and subroutine interface inconsistencies at compile time. The strong type checking of Ada that require greater specification of data and its usage is needed for embedded systems.

TER #77

4.3 The Ada coding is written in more of an english manner which makes it easier to understand.
4.4 With Ada's detailed data type definitions and its run time type checking and structured programming architecture more problems will be eliminated in the design and debug phases of program development. This should reduce the problems in the verification and validation phases.
4.5 It will be very difficult to convert the existing METAPLAN code without a major redesign..... It is not structured in the formal sense and does not flow from top to bottom as Ada programs must. ...a major redesign... is almost mandatory....
4.10 The Ada language is certainly a language of the 1980's. Its structured and highly readable constructs will provide cheaper and more reliable software in the future.

TER #78

0 My experience with Ada has been disappointing. It is a well thought out language useful in a teaching environment but of limited value for use in small real-time computers. The two major flaws in the Ada design are its unusual delta or fixed type variable notation and its complex interface with assembly language.
2.5 In general, Ada is a sledge hammer where only a tack hammer is needed. It seemed to take more time to set up a procedure and code its boiler plate specification than to actually develop and implement the real operation.
4.6 The extra time it would consume would not make it worthwhile.

TER #79

4.1 ...coding would certainly be done quicker in Ada. The debugging process would probably be quicker for an assembly language version.
4.3 If extensive nesting of different types of modules is done, the program can be very confusing.

TER #80

2.3 The appropriateness of the Ada record, aggregate, and package constructs for this application made them convenient candidates and therefore these concepts influenced the redesign significantly.
2.5 Much of the redesign was accomplished with certain Ada

features in mind. However, after the PDL was written, it was very easy to translate the PDL directly into Ada code.

2.6 In many cases, the Ada constructs were very appropriate; i.e. aggregates, records.

4.1 The specific problem could have been resolved in a shorter time by using languages with constructs similar to the SIMSCRIPT attributes, entities and sets, although this language would lack some of the highly desirable features of Ada.

<u>TER #81</u>

2.5 ...we couldn't figure out how to do string conversion in Ada.

2.7 We changed several of our design ideas during the Ada coding primarily to take advantage of the advanced features of Ada and only once to escape a serious difficulty. The specific feature that had the strongest influence on our redesign was packaging and visibility rules.

4.5 ...I/O in Ada seems to be either sadly lacking or at least badly explained, especially for input ASCII string conversion.

<u>TER #82</u>

2.6 Having knowledge of Ada does help to arrive at a better design. This was illustrated by the history of my redesigns. As my understanding of Ada increased, so did the quality of my design.

4.4 It is likely that the consequences of changes to Ada programs will be more quickly and accurately understood than for programs written in other languages.