TN-91-6
June 1991
(TN)

The Comparison and Selection of Programming Languages for High Energy Physics Applications

Bebo White

Stanford Linear Accelerator Center
P.O. Box 4349, Bin 97
Stanford, California 94309 USA

This paper discusses the issues surrounding the comparison and selection of a programming language to be used in high energy physics software applications. The evaluation method used was specifically devised to address the issues of particular importance to high energy physics (HEP) applications, not just the technical features of the languages considered. The method assumes a knowledge of the requirements of current HEP applications, the data-processing environments expected to support these applications and relevant non-technical issues. The languages evaluated were Ada, C, FORTRAN 77, FORTRAN 90 (formerly 8X), Pascal and PL/1. Particular emphasis is placed upon the past, present and anticipated future role of FORTRAN in HEP software applications. Upon examination of the technical and practical issues, conclusions are reached and some recommendations are made regarding the role of FORTRAN and other programming languages in the current and future development of HEP software.

I. Introduction

The programming language to be used for any software application is a critical determinant of the speed of software development, the ease of software maintenance and the portability of software to other systems. Many language comparisons have appeared in computer science and programming literature. A large portion of these comparisons have been conducted on the languages *in situ*. Little, it seems, has been written about how languages should be evaluated and assessed with respect to specific software projects.

Physicists have long been recognized as among the most knowledgeable of natural scientists with respect to the technical aspects of computing. Advancements in high energy physics have been driven by advancements in computer hardware and software technology and vice versa. There is significant cross-over of physicists into computer-related tasks. Physicists and developers of physics software generally have the expertise to make a choice of development programming languages.

This paper describes an exercise in which a number of computer programming languages were evaluated specifically for high energy physics (HEP) applications. The method devised for this evaluation is based upon a knowledge of the requirements of the applications involved, the data-processing environments expected to support those applications and additional surrounding technical and non-technical issues. This method, coincidentally, closely parallels the feasibility and requirements analysis common to many software engineering methodologies. Therefore, the goals of this study were,

Contributed to the International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics, Lyon Villeurbanne, France, March 19-24, 1990
Presented at the Conference on Computing in High-Energy Physics, Oxford, April 1989

° to systematically evaluate candidate programming languages specifically for high energy physics applications; and

° to evaluate the role of programming languages in future HEP computing environments.

II. Methodology

In order to perform a meaningful language comparison, it was necessary to define specific language evaluation elements. These evaluation elements define the language comparison environment and insure that only the meaningful features of the candidate languages are considered. A concise definition of these elements hopefully reduces the possibility of "programming language bigotry" and the comparison of irrelevant language features. For this study, the following elements were identified:

° Technical specification of a generic HEP programming application;

° Identification of candidate programming language features necessary to satisfy these specifications;

° Candidate programming language "fit" in HEP software environments;

° Anticipated growth and future development of the candidate programming language.

III. Identification and Comparison of Relevant Language Features

High energy physics data reduction and analysis programs provide good examples of highly numerical-intensive, batch-oriented scientific programming applications. An analysis of author-solicited, "typical" off-line programs used at SLAC and CERN was used to compile a list of common processes required by such applications. This list included

° input/output of binary files

° 64-bit floating-point arithmetic

° operations using complex data types

° vector and matrix arithmetics

° data structure manipulation

° access of common blocks of data

° direct addressing of dimensioned variables (as distinct from matrix operations and including non- zero lower bounds for arrays)

° separate or independent compilation of subprograms and cross module checking

- ° subprogram parameter passing by value and reference and the ability to pass routines as parameters
- ° access to libraries of mathematical functions
- ° access to histogramming services
- ° access to graphics services
- ° extended file or database services (e.g., particle tables).

This list of processes was used to identify the candidate programming languages to be evaluated in the study. It was expected that the candidate languages would either have features which would map directly to the processes on the list or would have access to toolsets which would supplement their functionality. It was also decided that the candidate languages would be well-established (i.e., standardized to some degree), scientific and algorithmic. The languages which met these criteria were Ada, C, FORTRAN 77, FORTRAN 90 (formerly 8X), Pascal and PL/1.

The next step was to derive a "wishlist" of language features from the list of typical HEP software processes. Specific candidate language features were then compared against this list. The result is illustrated in Table 1, "The Detailed Language Features Comparison." References in this table provide clarification and indicate alternatives to deprecated features. In this way each candidate language can be realistically evaluated in terms of its standard and non-standard features, implementation-dependent features and extensions and toolsets commonly available in HEP computing environments.

* Ada has the broadest base of features of the languages being discussed. It has an extensive range of array definitions and permits subprograms to use arbitrarily sized arrays. Ada does not permit the user to define the internal representation needed for multi-dimensional arrays. Intrinsic array operations are not defined within the language and are viewed as an extension in an Ada package if required. Ada offers extended capabilities in the area of bit manipulation on the order of enumerated types. Real data types are defined within the language and support for extended precision and complex arithmetic can be provided through packages in a reasonable way, since overloading of operators allows the mathematical syntax to be preserved. However, Ada does not recognize the need for at least two (single and extended precision) structurally different floating point arithmetics. Ada was obviously designed with conscious emphasis on sound software engineering. It has the best ability of the candidate languages to hide data and routines. It is especially strong in demanding a clarity of exposition of the data and routines which may be imported to, and exported from, any program unit by providing cross-module checking. Ada provides a

wide range of I/O facilities though a set of predefined packages, such as **DIRECT_IO**, **TEXT_IO**, or **LOW LEVEL_IO**. The **with** and **use** constructs can be used for computation.

* C is a language with functionality similar to Pascal, but it is much less strict in the consistency checking of program units. LINT must be used to achieve the kind of cross module checking that Ada and Pascal offer. Data hiding is available at the same level as Ada. C defines I/O through a standard library (library **stdio**). C defines real and double precision data types in the language, and support for complex arithmetic can be provided using the **typedef** and **struct** facilities, but without retaining the mathematical syntax. C manages fairly well in the area of bit manipulation. Bit fields of variable widths can be incorporated into user-defined structures, and a pointer type can be used to locate the structure at a given address in memory. Sets are not defined. Enumerated types are not defined in Kernighan and Ritchie but are in the ANSI draft standard . The language provides a very useful preprocessor for performing such operations as macro definition and library inclusion.

* FORTRAN 77 provides reasonable support for array handling and manipulation. It is the only one of the candidate languages which defines real, double precision and complex data types as part of the language. FORTRAN 77 contains a very clear definition for formatted and unformatted I/O to terminal, printer and mass storage. However, it is clear that FORTRAN 77 lacks a great deal of the functionality that the other languages offer, and little type checking and no cross module checking. There is no data hiding beyond local variables. In realistic HEP applications, FORTRAN 77 must be supplemented by additional software tools. At CERN, the whole concept of ZEBRA is a manifestation of one of FORTRAN 77's needs (i.e., data structure manipulation). FORTRAN 90 has directly addressed many of these deficiencies of FORTRAN 77.

* Pascal, as implemented by both DEC and IBM, is very similar to Ada from the stand-point of computing itself. It offers the ability to hide data and code at the same level and cross module checking. It does not offer multi-programming as part of the language. Pascal provides rather simplistic I/O facilities in the Jensen and Wirth definition which are not as comprehensive as those of FORTRAN 77 and do not provide for unformatted (binary) data. Extended I/O facilities are common in implementation-dependent compilers. Extensions in the area of array manipulation alleviate many of the problems which exist in the Jensen and Wirth definition. Bit manipulation is possible in an arcane, poorly documented manner. Real data types are defined in the language, but double precision and complex must be emulated in user-defined data structures.

* PL/1 is a very rich language in terms of its constructs, nearly on a par with Ada. It offers data hiding at the same level as Pascal. Like Ada, PL/1 offers a language feature for exception

handling. It offers rather little, however, in terms of type checking and no cross module checking. According to the language definition, concurrent computation is part of the language. However, this feature was not available in the IBM and DEC compilers used. Complex and double precision data types are defined as part of the language. Bit functions are also supported. Sets and enumerated types are not defined in the language. PL/1 does provide a very powerful preprocessor which provides such capabilities as including text from an external library, conditional compilation of sections of the source program, macro development and variable name replacement.

The technical issues alone would appear to suggest the superiority of any of the candidate languages over FORTRAN 77. However, it is clear that this would obviously not be a realistic conclusion. From a historical perspective no programming language evaluation could be conducted with the prospect of completely replacing FORTRAN in HEP applications. This presumption is substantiated in the evaluation of practical issues.

IV. Programming Language "Fit" (Practical and Non-Technical Issues)

The following categories of practical issues relevant to collaborative HEP software applications have been identified:

° The Historical Role of FORTRAN

"For all its inelegance, and lack of safety features, it seems certain that FORTRAN will remain the main language for HEP code well into the 1990s...."

Computing at CERN in the 1990s

The widespread adoption of FORTRAN by the physics community probably came about because it was the best approximation to the general purpose language, capable of abstracting most of the computer-oriented ideas that one physicist wished to express to another (i.e., the *lingua franca*). Millions of lines of program code written in FORTRAN are a valuable foundation of the role of computing in high energy physics. Much of this code has proven itself over the course of many years to be accurate, reliable, efficient and flexible enough to be used as the requirements of experiments and even as physics itself have changed. Concurrently, the shortcomings of the language have led to the production of software libraries that alleviate many of its defects, and represent a huge investment in accumulated expertise. From a historical perspective it is more realistic to evaluate programming languages in the HEP environment in such a way as to complement FORTRAN applications. To fit Ada, C, Pascal or PL/1 into a HEP application of any consequence

will depend on the ability of that language to communicate effectively with modules, libraries and other program units which were developed in FORTRAN.

° Code Portability and Maintainability

Code portability is one of the major *practical* issues in a HEP experimental environment. Experiment collaborators are literally "seconds away" from one another via network. Programs, subprograms, libraries, etc. are easy and necessary to share. Many of the advantages of a collaborative environment would be lost if software portability presented a major problem. The ideal programming language would operate independently of the hardware and the operating system within which it functions. Programs written in that language would execute with minimum modification on all collaborative systems and yield identical results. The portability of software written in a high level language depends upon the availability of the appropriate compilers on the target machines or on the existence of a compiler that is itself portable and which can be moved easily to new machines at a cost far less than that required to produce it initially. Moreover, for portability to be successful, a common subset (or dialect) of the high level language must exist among the compilers. Given this scenario, the programmer has a maximum degree of independence and the functionality allowed by the portability of software.

° Inter-Language Communication

It is not uncommon in the evolution of a software system to want to program a new application in one programming language while still maintaining the use of existing libraries that have been programmed in a different language. In most cases the recoding of existing software solely for compatibility cannot be cost justified. This concept of inter-language communication has played a very important historical role. Subprograms and libraries programmed in assembler language offer greater efficiency to applications programs written in higher level languages. The increasing demand for program portability across multiple computer systems has pushed the demand for inter-language communication to the higher level language level from the assembly language level. Separate compilation of program modules allows for compatibility at the source code level if the module interfaces are precisely defined. Effective inter-language communication deletes the need to "re-invent the wheel" and allows the capability to integrate the old and the new in a satisfactory manner. In a HEP environment, this is a measure of the fit to existing FORTRAN libraries, graphics and database facilities.

° Language Standardization

Language standardization activity must be considered for projects with lengthy life cycles. The technical direction assumed by the standardization committee for a programming language is of critical importance. Standards activities should attempt to preserve investments in software written in the language and to create new standards with as high a degree of compatibility as possible with previous standards. At issue is the question of object code compatibility and source code compatibility.

V. Preliminary Conclusions

A survey of the technical and practical features of the candidate languages led to the following preliminary conclusions:

- All of the candidate languages are mature;
- Ada has the broadest base of standardized features;
- C has broad base of features coupled with high portability and availability factors;
- FORTRAN 77 has all the HEP data types native; functionality has a strong dependence on extensions and additional software tools especially with data structures;
- FORTRAN 90 holds great promise, but when will reliable compilers be available?
- Many important Pascal features are implementation-dependent;
- PL/1 is a very rich in features; availability could be a major concern.

It is obvious from these conclusions that not one of the candidate languages is clearly superior for HEP applications when considering technical and practical issues. Yet, it does become clear that the size and complexity of HEP software systems is forcing the HEP community to confront the notorious "Software Crisis." It may have been more realistic to address the question " Can an evaluation of programming languages for HEP applications help improve programmer/physicist productivity and increase the reliability and maintainability of HEP software systems ?"

VI.What About Multi-Language Systems?

Instead of expecting (or hoping) a single programming language would be an obvious choice for HEP applications, perhaps a more realistic future lies in *multi-language systems?* As was indicated earlier, FORTRAN pioneered multi-language programs with Assembler routines. However, portability requirements have pushed the issue from assembler to higher order languages. All of the candidate languages have some capability for inter-language communication. Opening the door for multi-language systems allows complex and collaborative HEP software projects to:

- choose the best tool (language, toolset, library, software product, etc.) for the job at hand;
- choose the best programmer/software designer for the tool;
- take better advantage of the technology and expertise in current computing.

VII. Conclusions

The following general conclusions were reached as a result of this programming language comparison and evaluation:

- The common assumption that FORTRAN "will remain the main language for HEP code well into the 90's" is a valid, but conservative, one;
- HEP software development in alternative languages and multi-language systems should be encouraged if it can be proven to be a sound software design decision.

Goal of This Study

- To systematically evaluate candidate programming languages *specifically* for high energy physics applications; and

- To evaluate the role of programming languages in future HEP computing environments

"For all its inelegance, and lack of safety features, it seems certain that FORTRAN will remain the main language for HEP code well into the 1990s...."

Computing at CERN in the 1990s

"FORTRAN is probably the only perennial standard which will never be questioned."

Trends in Computing for HEP

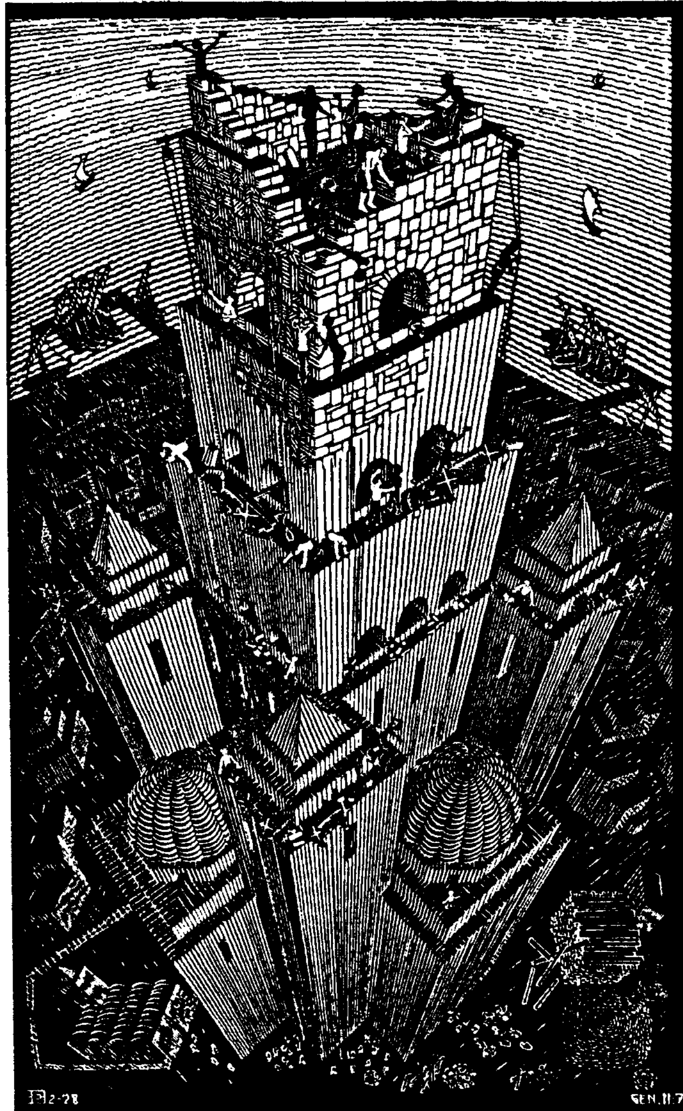
"I don't know what the language of the year 2000 will look like but I know it will be called FORTRAN."

Tony Hoare

"If HEP wishes to keep to its level of achievement, credibility and excellence, then it needs an injection of bright young computer-wise scientists and engineers."

Paolo Zanella

The Tower of Babel, Not a New Accelerator Design



Language Evaluation Elements

- Technical specification of the programming application
- Analysis of candidate programming language features
- Programming language fit in the application environment
- Growth and future development

Typical/Required Processes in HEP Applications

- input/output of binary files
- 64-bit floating-point arithmetic
- operations using complex data types
- vector and matrix arithmetics
- data structure manipulation
- access of common blocks of data
- direct addressing of dimensioned variables (as distinct from matrix operations and including non-zero lower bounds for arrays)
- separate or independent compilation of subprograms

- subprogram parameter passing by value and reference and the ability to pass routines as parameters

- access to libraries of mathematical functions

- access to histogramming services

- access to graphics services

- extended file or database services (e.g., particle tables).

Wishlist and Detailed Language Feature Comparison

Features	Ada	C	F77	F8x	Pascal	PL/I
address arithmetic	(1)	yes	no	no	yes	yes
arbitrary array bounds	yes	(2)	yes	yes	yes	yes
arbitrary function return value	yes	(3)	no	(4)	yes	(5)
argument by name or position	yes	no	no	yes	no	no
array bound checking	yes	no	(6)	no	yes	yes
array of structures	yes	yes	(7)	yes	yes	yes
array operations	(8)	no	no	yes	(9)	yes
binary I/O	yes	yes	yes	yes	no	yes
bit logic	(10)	yes	yes	yes	(11)	yes
call by reference	yes	yes	yes	yes	yes	yes
call by value	yes	yes	no	(12)	yes	no
complex variables	yes	(13)	yes	yes	(14)	yes
concurrent computation	yes	no	no	no	no	(15)
cross module argument checking	yes	(16)	no	(17)	yes	no
data hiding	yes	yes	no	yes	yes	(18)
descriptive variable names	yes	yes	(19)	yes	yes	yes
double precision	yes	yes	yes	yes	(20)	yes
dynamic memory	yes	yes	(21)	yes	yes	yes
enumerated data types	yes	(22)	no	no	yes	no
exception handling	yes	no	(23)	(24)	(25)	yes
formatted I/O	yes	yes	yes	yes	yes	yes
inhomogeneous data structures	yes	yes	(26)	yes	yes	yes
inter-language communication	(27)	(28)	(29)	(29)	(30)	(31)
local procedures	yes	no	no	yes	yes	yes
multiple subprogram entries	(32)	no	yes	yes	no	yes
pass arbitrary matrix	(33)	(34)	yes	yes	(35)	yes
pass arbitrary 1-dimensional array	yes	(36)	yes	yes	(37)	yes
pass structure to subprogram	yes	yes	(38)	yes	yes	yes
pointers to functions	(39)	yes	no	no	no	no
pointers to variables	yes	yes	(40)	yes	yes	yes
preprocessor	(41)	yes	(42)	no	(43)	yes
recursion	yes	yes	no	yes	yes	yes
routine hiding	yes	yes	no	yes	yes	(44)
sets	yes	no	no	no	yes	no
static variables	yes	yes	yes	yes	(45)	yes
strings	yes	yes	yes	(46)	(47)	yes
strong type checking	yes	(48)	(49)	(49)	yes	no
subprogram argument checking	yes	(50)	no	(17)	yes	no
user generic functions	yes	no	no	yes	no	no
variable equivalence	(51)	(52)	yes	yes	(53)	yes
variable initialization	yes	yes	yes	yes	(54)	yes
variable range checking	yes	no	no	no	yes	no

References

(1) unknown
(2) low = 0
(3) in ref. [7] only simple variables or pointers; ANSI draft standard also includes structures [8]
(4) user generic functions may be written
(5) scalars, bit strings, string, entries, pointer, some expressions
(6) DEC – yes; IBM – with IAD facility
(7) DEC – yes; IBM – no
(8) use operator overloading (external in a package)
(9) can assign an array to another, but no operations on arrays as units
(10) unknown
(11) implementation dependent
(12) the INTENT of a dummy argument may be specified
(13) can use typedef and struct, but without mathematical syntax
(14) no, must be emulated with records
(15) defined in the language definition; not implemented by DEC or IBM
(16) no, necessary to use LINT
(17) yes, if INTERFACE blocks are used
(18) no, only in local procedures
(19) DEC – yes; IBM – no (only 8 characters allowed)
(20) no, must be emulated
(21) no, use ZEBRA
(22) no, in ref. [7]; yes in the ANSI draft standard [8]
(23) implementation dependent
(24) yes, for I/O
(25) implementation dependent
(26) DEC – yes; IBM – no, use ZEBRA
(27) defined via INTERFACE PRAGMA; not always implemented
(28) DEC – yes; IBM – yes; standard definition – unknown
(29) EXTERNAL declaration
(30) defined via EXTERNAL and FORTRAN declarations; implementation dependent
(31) yes, with parameters of the DECLARATion and PROCEDURE statements and only with specific languages
(32) unknown
(33) unknown
(34) pass array of pointers to arrays; both must start at 0
(35) pass array of pointers to arrays; turn off range checking
(36) lower bound must be 0; no bound checking possible
(37) DEC(ISO) – yes; IBM(ANSI) – no; turn off range checking
(38) DEC – yes; IBM –no
(39) not in the sense of C; pass routine at FORTRAN level
(40) no, use ZEBRA
(41) conditional compilation, declaration import, and simple lexical substitution of constants with arithmetic are part of the language
(42) yes, MORTRAN is one example
(43) substitution of constants with arithmetic is supported by DEC and IBM
(44) no, only as local procedures
(45) implementation dependent
(46) can be coded as a MODULE
(47) implementation dependent
(48) yes for simple variables, no for routines
(49) if IMPLICIT NONE is used
(50) no, must use LINT
(51) unchecked-conversion or variant record
(52) variant record or union
(53) variant record or union
(54) implementation dependent

Practical Issues

- The FORTRAN legacy
- Portability
- Maintainability
- Standardization (features vs. extensions)
- Availability of knowledgeable users
- Fit with current technology, available tools and projected hardware environments

Preliminary Conclusions I

- All of the candidate languages are mature

  - Ada has the broadest base of standardized features

  - C has broad base of features coupled with high portability and availability factors

  - FORTRAN 77 has all the HEP data types native; functionality has a strong dependence on extensions and additional software tools esp. with data structures

  - FORTRAN 90 holds great promise, but when??

  - Many important Pascal features are implementation-dependent

  - PL/1 is a very rich in features; availability could be a major concern

HEP Programming Languages and the "Software Crisis"

Can an evaluation of programming languages for HEP applications

- Improve programmer productivity?

- Increase the reliability and maintainability of HEP software systems?

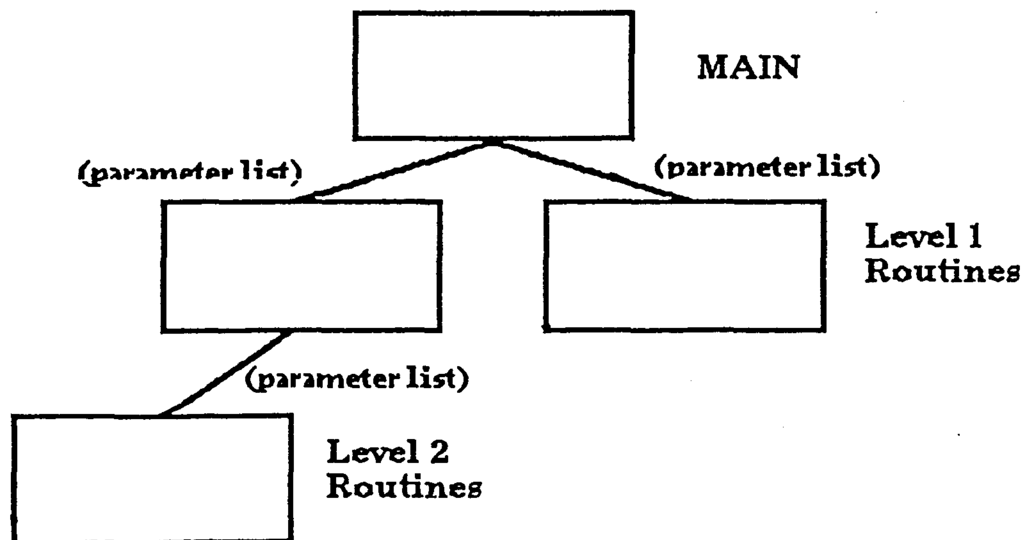
Can Software Engineering Help?

The software project life-cycle:

1) Feasibility and Requirements Analysis
2) Logical Design
3) Detailed Design
4) Coding
5) Implementation
6) Maintenance

Structure Charts

- A tool often used in the Detailed Design phase
- Defines the skeleton of the final system in terms of subprogram structures, data structures, etc.
- Helps to separate interface specification from implementation details
- Demonstrates the "maintenance view" of the system



What About Multi-Language Systems??

- FORTRAN pioneered multi-language programs with Assembler routines

- Portability requirements have pushed the issue from Assembler to Higher Order Languages

- Allows:

  - The best tool for the job

  - The best designer for the tool

Inter-Language Communication

Ada	defined via INTERFACE PRAGMA; not always implemented
C	DEC - yes; IBM - yes; standard definition - unknown
FORTRAN 77	EXTERNAL declaration
FORTRAN 90	EXTERNAL declaration
Pascal	defined via EXTERNAL and FORTRAN declarations; implementation-dependent
PL/1	yes, with parameters of the DECLARATion and PROCEDURE statements and only with specific languages

ILC Compatibility Issues

data type conventions: it is imperative that data type matching occurs with parameters passed to modules;

array conventions: it is necessary to know how multi-dimensional arrays are mapped into memory by the called and calling languages;

calling conventions: it is necessary to know how parameters are passed to modules, how values are returned and how register and memory management is handled.

FORTRAN ⇔ C Data Typing

C Type	FORTRAN Type
int *	INTEGER*2
long int *	INTEGER*4
float *	REAL*4
double *	REAL*8
float[2]	COMPLEX
Struct Complex *	COMPLEX
double[2]	COMPLEX*16
Struct Double Complex *	COMPLEX*16
char *	LOGICAL*1
struct CHARACTER{ char * text; int length; }	CHARACTER*(*)

Conclusions II

- The assumption that FORTRAN "will remain the main language for HEP code well into the 90's" is a valid, but conservative, one;

- HEP software development in alternative languages should be encouraged *if it can be proven to be a sound software design decision.*