

**PORCTIONS
OF THIS
DOCUMENT
ARE
ILLEGIBLE**

32

LA-UR-79-3294

CONF-801001--2
CONF-801012--2

TITLE: FORTRAN FOR THE 1980's

AUTHOR(S): Jeanne Adams
Walt Brainerd

SUBMITTED TO: IFIP '80

MASTER

University of California

DISCLAIMER

This report was prepared as an account of work sponsored by the United States Government. It is not to be distributed outside the government except by special permission. The views and opinions contained herein are those of the author(s) and do not necessarily represent those of the United States Government. This report is the property of the United States Government and is loaned to you; it and its contents are not to be distributed outside the government except by special permission. This report is the property of the United States Government and is loaned to you; it and its contents are not to be distributed outside the government except by special permission.

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes.

The Los Alamos Scientific Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.



LOS ALAMOS SCIENTIFIC LABORATORY

Post Office Box 1663 Los Alamos, New Mexico 87545

An Affirmative Action/Equal Opportunity Employer

20

FORTRAN FOR THE 1980's

Jeanne Adams
National Center for Atmospheric Research
P.O. Box 3000
Boulder, Colorado, 80307, USA

Walt Brainerd
University of California
Los Alamos Scientific Laboratory
P.O. Box 1663
Los Alamos, New Mexico 87545, USA

Technical area: Software

"Neither this paper nor any version close to it has been offered elsewhere for publication and, if accepted, the paper will be personally presented in at least one of the 8th World Computer Congress locations by one of the co-authors."

Walter J. Brainerd

Preference for presentation: Tokyo, Melbourne, or both

ABSTRACT

The new standard Fortran 77 has not been available long but the American National Standards Institute (ANSI) committee X3J3 responsible for Fortran standardization is already working on the next revision. Since the result of this work will be a candidate for an international (ISO) standard, it is important that work being done now become known to all persons interested in Fortran.

A new set of problems related to the accommodation of related standards in data base management and real time process control, as well as the ever-increasing size of the language, have caused the standardization committee to consider some new approaches to the development of the next standard. These new approaches and many of the new features that probably will be in the next Fortran standard are described. It is hoped that this presentation will stimulate comments and suggestions in time to include them before the next standard is finalized.

The authors are the chair of the ANSI Technical Committee X3J3 and the officer of the committee responsible for the management of the technical work, respectively.

1. INTRODUCTION

A revised Fortran standard was adopted as an American National Standard in 1978 [1,2] and the identical specification was adopted as an international (ISO) standard in 1979. This language is called "Fortran 77" because its development was completed in 1977. The technical committee X3J3 of the American National Standard Institute (ANSI) is currently working on the next revision of the Fortran standard.

If history repeats itself, the revision that X3J3 is now preparing eventually will be adopted as an international standard. For this reason, the members of X3J3 realize that it is very important to keep all of those persons throughout the world that are interested in Fortran informed about the problem areas and possible solutions that are being considered by X3J3. Even more important are the opinions and suggestions received from people and organizations. Members of X3J3 have met informally on three occasions with members of ISO in western Europe. These meetings have been very valuable to X3J3 and the authors hope that this presentation will stimulate more interaction between X3J3 and people in other parts of the world.

One of the reasons that international interaction is so important during this revision of the Fortran standard is that language developers are facing a set of problems very different from those faced during 1970's and before.

One of the phenomena affecting language development is the view that a programming language is one tool that must be

integrated with other software tools, such as editors, graphics systems, and data base management systems. This is complicated by the fact that these software tools often are associated with hardware systems that are distributed. The total software system, including the programming language Fortran, must operate effectively in this new environment. Indeed, there are already standards for real-time process control in a Fortran environment [3,4,5]. A proposal is now being drafted for Fortran language extensions to implement the CODASYL model of a data base management system [6]. How are these systems to interface with the Fortran language system?

Another reason that this development cycle for Fortran is different is that it is becoming recognized that the Fortran language has become too large. It contains many features that are obsolete or redundant. One of the difficult tasks faced by X3J3 is to provide a way to remove obsolete language features and still protect the tremendous economic investment in Fortran programs and Fortran programmers.

2. LANGUAGE ARCHITECTURE

X3J3 is attempting to address the two major problems of collateral software systems and an overweight language by developing a more complex architectural model for the Fortran language.

One approach that is being considered is a "core-plus-modules" organization of features. The core is to be the nucleus of Fortran consisting of a complete set of language

features. It is intended that most general purpose applications could be written in Core Fortran. Modules contain features that are not in the core, but are used with the core to provide an enhanced facility.

Three kinds of modules have been identified.

1. an extensions model
2. an obsolete features model
3. applications modules

2.1 Core Criteria

In order to achieve correct placement of various proposed facilities in the core or a language module, it is necessary to have a set of criteria for the decision process. The set of criteria for the placement of features in the core includes the following.

1. The core must be a complete language suitable for most general purpose applications.
2. The core must be internally consistent and regular.
3. It must consist of features characterized as reflecting modern software technology.
5. Programs written in Core Fortran should compile and execute efficiently.

6. Features in the core must be portable (i.e., machine independent).
7. Features should not provide duplicate functionality.

2.2 The Obsolete Features Module

It should be possible to retire features that no longer serve the Fortran users well. However, this must be done in a way that does not require that all programs using these features be converted immediately. This may be accomplished by placing candidates for retirement in an obsolete features module. This will serve as a notice that these features will probably not appear in future versions of the standard, so that the features can be removed gradually from old programs and not be used in new programs. It is assumed that marketplace pressures will keep these features in Fortran implementations as long as it is economically advantageous to do so.

2.3. The Extensions Module

The extensions module provides a means of introducing features that may or may not survive in the language, but will be identical in all implementations that contain the extensions.

The extensions module also provides a place to put language features that may be of special importance to a particular class of problems. These features may not satisfy the criteria for inclusion in the core, but are common enough that it is worth the effort to specify that all implementations of the feature

should be identical. A good example of this type of feature is a collection of sophisticated array processing facilities. These may be of interest only to those doing large numerical computation problems on machines with special array processing capabilities. It may not be practical to implement the features on small machines.

The extensions module and obsolete module features provide a way for features to enter the language and leave the language at the end of their useful life with a minimum of impact on the large software investment in existing programs. Extensions that prove not to be useful to the users of Fortran may never appear in the core language. They may be dropped from the extensions module or moved to the obsolete module.

2.4 Applications Modules

The new architecture allows for the possibility of many kinds of applications modules interfacing with the rest of the language. Most applications modules will specify a standard set of external procedures to introduce special purpose facilities. This will be made easier by the enhanced procedure call mechanism discussed in Section 3.4. Some applications may introduce extensions such as those required by the CODASYL Data Base Journal of Development [6]. Some of the possible applications modules are those for data base management, graphics, and real time process control.

The model in Figures 1 and 2 identifies the proposed processes for standardizing various Fortran features. It is

hoped that the architecture will allow the development of the Fortran language to proceed dynamically well into the future. The plans for features to be introduced cautiously and with attention to the current usage, and the plans for potential obsolescence of archaic features should provide a constructive method for managing growth and change in Fortran.

3. NEW LANGUAGE FEATURES

At this time, it is impossible to describe all of the features of the next Fortran language standard. However, there are a few major features that are quite likely to be included. These are discussed with a few examples of how some of these features might be used in programs.

It must be emphasized that these features and their descriptions are tentative. The committee X3J3 may change the form or content of any of these features, particularly if significant comments about them are communicated to the committee.

The major features that will be discussed are the following.

1. control structures
2. data structures and data types
3. array processing
4. enhanced procedure calls
5. program form

6. precision and environmental inquiry

7. dynamic storage allocation

3.1 Control Structures

New control structures are being considered that will encourage structured programming techniques. Among these are looping extensions, a CASE statement, and a means of grouping procedures.

The form of a DO statement has been modified to enable a loop to be bracketed by a new DO statement and a REPEAT statement. Examples of the new form of the DO statement are

```
DO
DO (N+1 TIMES)
DO (I = 1, N+1)
```

where the keyword DO by itself means DO forever (i.e., until an exit is taken). There is an EXIT statement that causes a branch out of the innermost loop. No DO WHILE or DO UNTIL constructs have been added because these can be built by putting an EXIT statement at the appropriate point in the loop. For example

```
DO (WHILE X .GT. EPS)
...
```

can be written as

```
DO
IF (X .LE. EPS) EXIT
...
```

An example of how a case construct might look is the following.

```
SELECT CASE (N+1)
  CASE (1,3,9)
    CALL A
  CASE (7,11)
    X=7.6
    CALL C
  OTHERS
    CALL D
END SELECT
```

Two schemes for grouping procedures have been considered. One is to allow procedures internal to other procedures, such as having a function be internal to a subroutine. Variables in the outer procedure would have a scope that includes the inner procedure unless that same variable name were also declared in the inner procedure. The other scheme involves collecting procedures into a group and allowing variables to be declared within the group and have a scope that includes all of the procedures in the group. It would be possible to export declarations and procedures for use outside the group and to import declarations and procedures from other groups.

3.2 Data Structures and Data Types

There is a new capability for defining collections of heterogeneous types of data so that manipulation of the structure and the various components of the structure is possible. A

form definition indicates the form of a structure and the names and attributes of its fields. Variables may then be declared to have that form.

For example, a structure consisting of a name and identification number could be declared by

```
FORM: PERSON
      CHARACTER*20 NAME
      INTEGER ID_NUMBER
      END FORM
```

```
PERSON: KUKO, IAN, TAXPAYER (100)
```

The identification number of taxpayer 43 would then be referenced by TAXPAYER (43).ID_NUMBER.

One new data type, bit data, has been included. It is modeled on the character data type. Included are a new type statement, bit substrings, bit string concatenation, and several operations, such as "and", "or", "exclusive or", and "complement". A number of bit intrinsic functions have been added also. Examples of bit constants are '0'B and '011101'B.

The IMPLICIT NONE statement removes the default integer and real typing within a program unit. This means that all variables in that program must be declared explicitly.

3.3 Array Processing

A basic set of extensions for array processing has been proposed. In this proposal, Fortran 77 operators (arithmetic,

relational, logical, character, and assignment) are extended to operate element-by-element on arrays that are the same size and shape. This allows manipulation of arrays as fundamental objects and facilitates "formula translation" for array and matrix oriented problems. For example, it is possible to replace

```
REAL A(10,30), B(10,30), C(10,30)
DO 10 I = 1,10
DO 10 J = 1,10
10 A(I,J) = B(I,J) * ((I,J) + 2.7
```

by

```
REAL A(10,30), B(10,30), C(10,30)
A = B * C + 2.7
```

As is illustrated by the example above, a scalar is an exception to the rule requiring that operands in an expression all be the same size and shape. A scalar is treated as an array of the correct dimensions, all of whose entries are equal to the scalar value.

Intrinsic functions have been added to accept arrays as arguments. An example is

```
A = SQRT (ABS(B) + 1.0)
```

where A and B are the arrays declared above.

Other intrinsics special to arrays are proposed, such as SUM (A), which yields a scalar sum of all the elements of an array A.

Many other array processing features, such as programmer-defined array-valued functions and subarrays are being considered to determine which set of facilities should be included in the next Fortran standard.

3.4 Enhanced Procedure Call

Many useful extensions to Fortran have been provided by using external procedures. Sometimes these procedures are written in Fortran and sometimes they are written in other languages. This technique provides additional facilities to the user without extending the language.

It is recognized that further enhancements of the procedure call mechanism will permit even more sophisticated applications procedures to be made available with only modest extensions to the language. Some of the ideas being considered are now described.

An extended form of the procedure call would permit arguments to be specified by keywords, as well as position. For example

```
CALL S (ARG3 = K, ARG2 = 2 * C + 1)
```

Such a call requires that the form of the arguments for S be described in the calling program. It is proposed that some kind of declaration such as

```
PROCEDURE S (REAL ARG1 = 7.6, REAL ARG2, INTEGER ARG3)
```

be used to indicate the type and keyword for each argument. It would also be possible to specify a default value for an argument omitted in the procedure call. In this example, ARG1 would have the default value 7.6.

Other ideas are being considered. These include permitting user-defined generic functions, intrinsic procedures, and a means for specifying if an argument is read only or read/write.

3.5 Program Form

It has long been recognized that Fortran's fixed source program form with labels in columns 1-5, continuation indicator in column 6, and statements in columns 7-72 is not convenient, particularly when entering programs at a terminal.

It is almost certain that the next standard will provide for programs written in a different form and it is likely that this form will be similar to that described here.

Fortran programs will be written on lines that may be of different lengths, but any line up to 72 characters must be accepted. None of the columns in a line have any special significance. However, the end of the line is still significant; it terminates a statement, unless continued, and terminates comments.

An exclamation mark (!) that is not in a character string begins a comment, which is terminated by the end of a line. A statement may be continued by placing an ampersand (&) as the

last character on the line to be continued. The continuation mark must precede a comment, if there is one.

More than one statement may be placed on one line if separated by a semi-colon (;). Also, local names may contain up to 31 letters, digits, or underscores (_).

This new form is illustrated by the following example.

```
REAL A (4,3)
DATA / &
    11, 21, 31, 41, & ! COL1
    12, 22, 32, 42, & ! COL2
    13, 23, 33, 43 / ! COL3

IF (X.GT.Y) THEN
    T=X; X=Y; Y=T ! SWAP X,Y
END IF

I=3;;; X=Y& ! BAD FORM!;
    + Z !!; 77;!
```

3.6 Precision and Environmental Inquiry

One of the severe difficulties encountered when attempting to move Fortran programs from one machine to another is that of precision. The word sizes of two machines may be very different, resulting in the necessity to change computations from REAL to DOUBLE PRECISION or vice versa. Also, some machines have more than two floating point precisions (e.g., quadruple) and there is no way to specify these precisions in Fortran 77.

A means of specifying the precision of any real variable or constant in decimal digits is being proposed.

For example

```
REAL X*4, Y*4, D*8
```

would indicate that X and Y must have at least 4 decimal digits of precision and that D must have at least 8 decimal digits of precision. The constant 2.7P9 would have to be represented with at least 9 decimal digits of precision. With specifications such as these, a program can be moved without change to a different machine with the guarantee that the desired results will be achieved, even if the program is actually run using what one machine calls "single precision" and what the other calls "double precision".

In order to evaluate the numerical stability of certain computations, it is necessary to ensure that some calculations are done using more precision than others. To accommodate this, a concept of machine precision level is introduced and a mechanism can be provided so that the programmer can say, in effect, "Do computation A using at least 6 digits of precision and do computation B using a machine level of precision that is greater than that used for computation A. Also, let me know how many digits of precision were actually used for each computation." The mechanism for specifying these computations was not worked out in detail at the time this was written.

A set of intrinsic functions is being added that will allow the program to determine characteristics of the environment in

which the program is running. These functions will include the date, the time of day, the largest and smallest integers, the radix used to represent a real number, etcetera.

3.7 Dynamic Storage Allocation

One of the criteria for the development of Fortran 77 was that no feature would be included if it required dynamic storage allocation. This criterion has been abandoned, primarily because it is recognized that there can be no significant array processing with such a restriction. Consider, for example,

```
REAL A(0:100)
A(1:N) = A(K+1: K+N)
```

This is not necessarily equivalent to the DO loop

```
DO (I = 1,N)
  A(I) = A(K+I)
REPEAT
```

Consider the case where $K = -1$. The DO loop will be equivalent to

```
A(1) = A(0)
A(2) = A(1)
A(3) = A(2)
...
A(N) = A(N-1)
```

which is equivalent to

```
A(1:N) = A(0)
```

However the assignment

```
A(1:N) = A(0:N-1)
```

must produce the same result as

```
REAL TEMP(1:100)
TEMP = A(0:N-1)
A(1:N) = TEMP
```

To accomplish this, it may be necessary (or at least desirable) to have a temporary array be created at run time.

Fortran 77 restrictions on the character data type have been eliminated. For example, it is now possible to write

```
CHARACTER C*100
C(1:N) = C(K+1 : K+N)
```

This is prohibited in Fortran 77 for the reasons cited above.

Also the following is prohibited in Fortran 77 because the amount of storage needed to hold the actual argument cannot be determined at compile time and so may require a temporary to be created dynamically. This would be permitted in the next standard.

```
SUBROUTINE S1 (C)
CHARACTER C*(*)
CALL S2 (C // 'Q')
END
```

Within a subroutine or function a local array, in addition to a dummy argument or one in common, may have dimension bounds declared using values that are known when the subprogram is entered. These values may be arguments, in common, or in DATA statements in the subprogram. This is illustrated in the following program that swaps two real vectors using a local temporary vector.

```
SUBROUTINE SWAP (A, B, N)
REAL A(N), B(N), TEMP(N)
TEMP = A; A = B; B = TEMP
END
```

Recursive subprograms have been excluded from Fortran, partly on the grounds that dynamic storage allocation is required. This feature is now being considered for inclusion in the next standard.

4. THE OBSOLETE FEATURES MODULE

With the introduction of the features described in section 3, there is a certain amount of redundancy in the language. One of the criteria for the core is that it should not contain redundant features. Therefore, the old features that are replaced in functionality by new features are candidates for the

obsolete module. We now look at some of these features and indicate which features replace them.

There is one concept in Fortran that is much better left out of the core entirely: It is the concept of storage and storage association. For example, there is no reason that a double precision variable must occupy exactly twice as much storage as a real variable, or that a real and integer variable must occupy exactly the same amount of storage. This was done in the past to explain the consequences of equivalencing variables of different types (via the EQUIVALENCE statement, COMMON blocks, or actual-dummy argument association). The useful features of these language constructs (e.g., global data) have been replaced, but the concept of storage association is being eliminated from the core.

Several obsolete control mechanisms are being moved to the obsolete module. These include the arithmetic IF, the assigned GO TO, the computed GO TO, alternate returns, the ENTRY statement, and the statement function. These are being replaced with the new DO statement with EXIT, the CASE statement, grouping of procedures, and, of course, the block IF statement of Fortran 77.

The old fixed program source form will be replaced by the new free form. Incidentally, this is the only case in which a proposed new feature conflicts with Fortran 77. It is impossible to have a free form and to interpret column 6 as a continuation indicator.

The EQUIVALENCE statement and other storage association features such as ENTRY association and COMMON association are being put in the obsolete module. The useful aspects of EQUIVALENCE have been incorporated into data structures, procedure grouping, and dynamic local arrays.

The DOUBLE PRECISION data type is unnecessary with the ability to specify precision in decimal digits; therefore, it is being moved to the obsolete module.

In addition there are several obvious redundancies that already existed in Fortran 77. The ERR= and END= specifiers are duplicated by the IOSTAT specifier. The H, X, and apostrophe edit descriptors are not needed with the T edit descriptor and the ability to put character expressions in an input/output list. With generic functions available, the specific names are needed no longer.

5. CONCLUSIONS

It should be obvious from this presentation that the next Fortran standard will be significantly different from Fortran 77. We consider it very important that all interested persons become aware of these developments before it is too late to modify them easily, and that any opinions and suggestions be conveyed to the committee. Either of the authors would be pleased to receive comments.

REFERENCES

1. American National Standard Programming Language FORTRAN, ANSI X3.9-1978, American National Standards Institute, New York, NY, 1978.
2. Brainerd, Walt, editor, Fortran 77, Communications of the ACM, 21, 10 (October, 1978), 806-820.
3. ANSI ISA 61.1, Industrial Computer System Fortran Procedures for Executive Functions, Process Input/Output, and Bit Manipulation.
4. ANSI ISA 61.2, Industrial Computer System Fortran Procedures for File Access and the Control of File Contention.
5. ANSI ISA 61.3, Industrial Computer System Fortran Procedures for the Management of Independent Interrelated tasks.
6. CODASYL FORTRAN Data Base Facility Journal of Development, Version 2.0, Ontario, Canada, 1980.

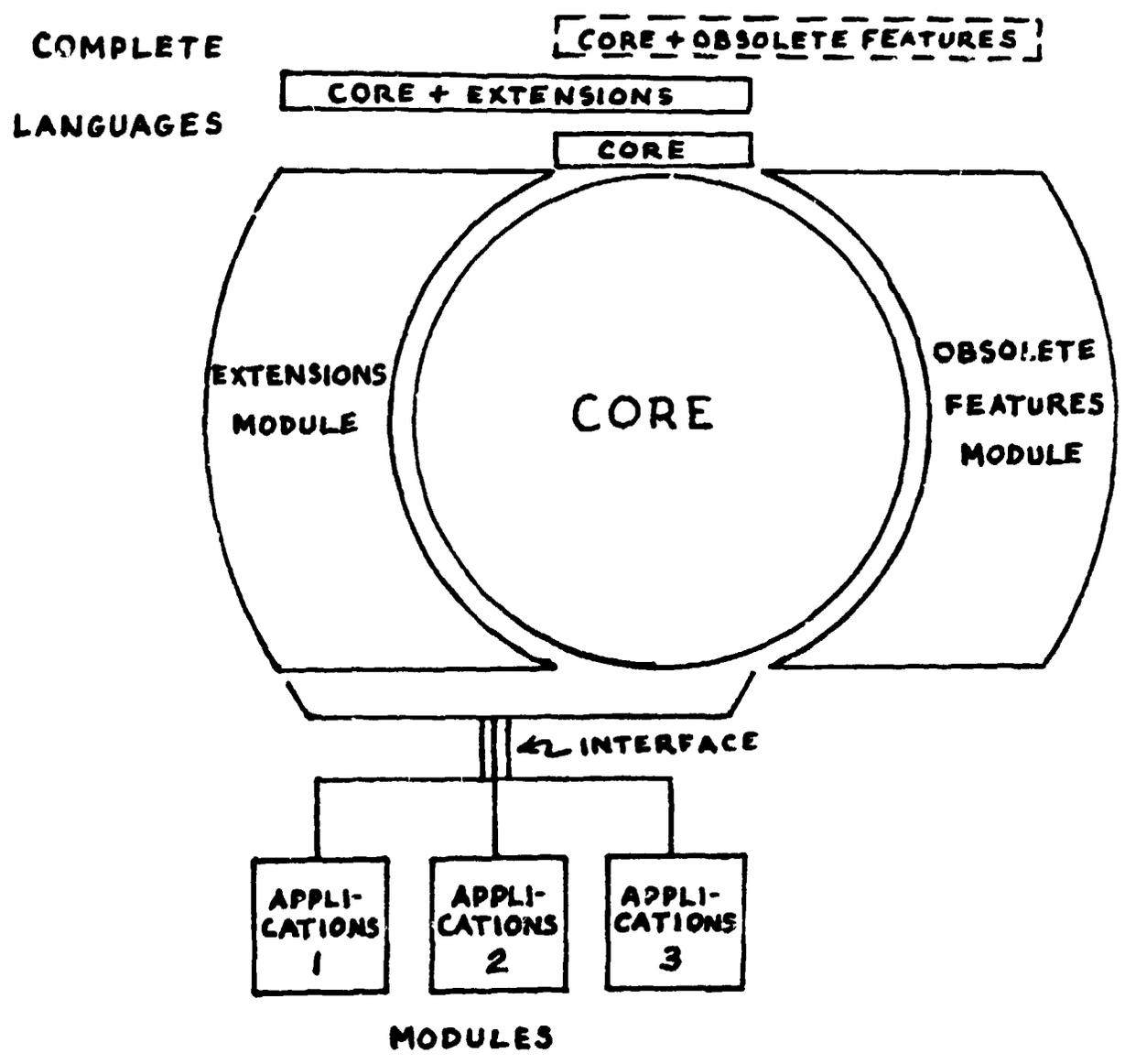
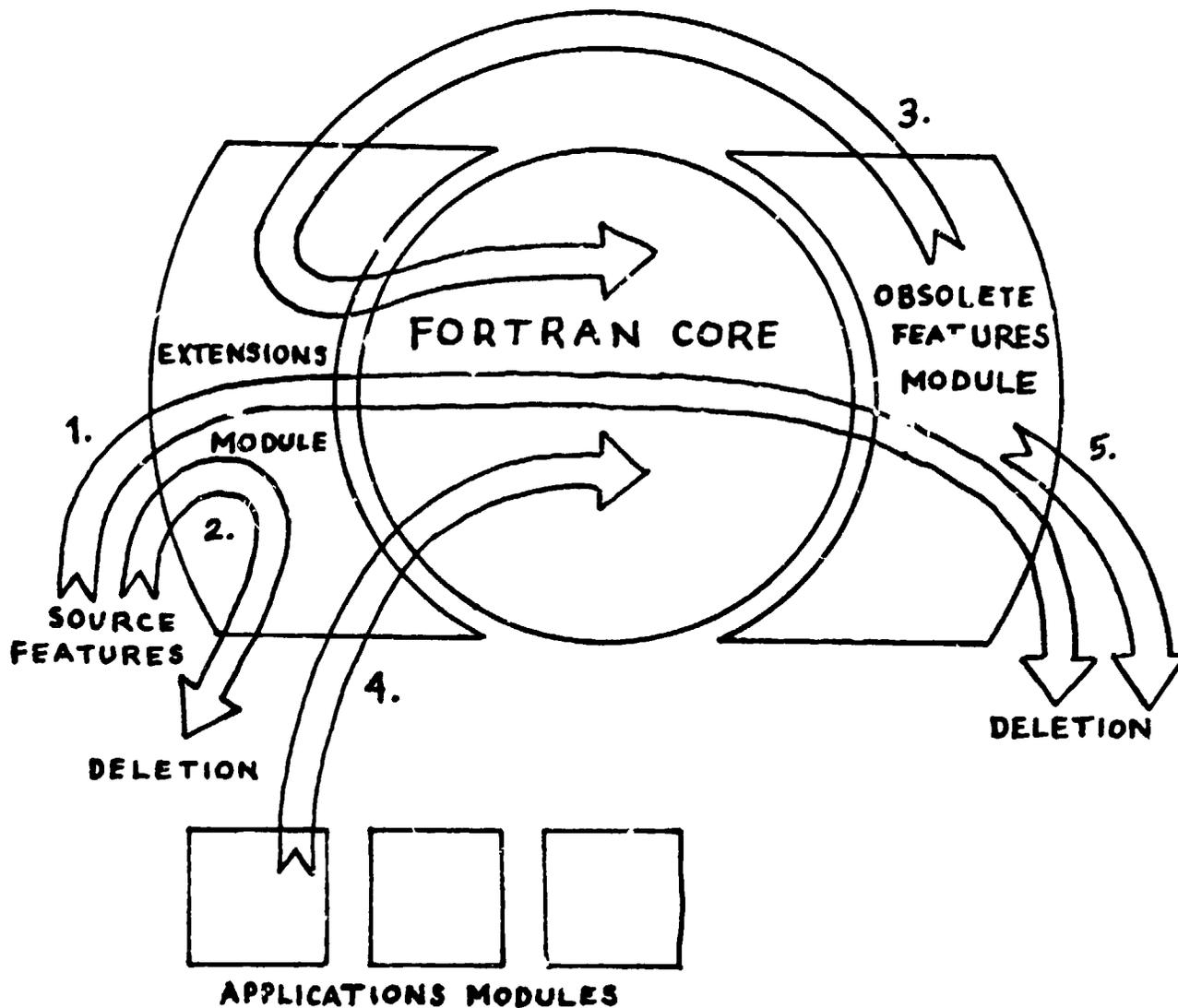


FIGURE 1: RELATION OF MODULES TO CORE



SAMPLE PATHS OF FORTRAN FEATURES:

- 1. EXPECTED PATH OF MOST FEATURES
- 2. IMMEDIATE DELETION OF UNSUCCESSFUL FEATURES
- 3. RESTORATION OF OBSOLETE FEATURES
- 4. APPLICATIONS SYNTAX
- 5. OBSOLETE FEATURES DELETED

FIG.2 DYNAMICS OF FEATURE PROCESSING