FORTRAN

Overview

FORTRAN (short for FORmula TRANslation) is a problem-oriented language concerned with problems of numerical calculations in scientific and engineering applications.

Why FORTRAN? FORTRAN as a programming language is decades old, but until recently it has been the programming language HEP experiments have been using. Software written in FORTRAN is still in active use, like

- PYTHIA6
- Alpgen
- Madgraph5_aMC@NLO (NLO calc)
- HDECAY

The FORTRAN version in use is FORTRAN77 or FORTRAN90

There is a transition to C++ in HEP software, programs are being rewritten in C++ and new programs are mostly written in C++. Not everything, though.

There are some peculiarities in F77. The statements start at the 7th position of a line, the first 5 positions are for labels when needed, and the 6th position signals a continuation line for long statements. This feature is error-prone so be careful to start a new statement line with 6 blanks.

The line must not be too long either. If the line is longer than 72 chars, the rest of the line is truncated. This can also result in errors which are not easy to find. The command can be extended to the next line by e.g. & char as the 6th char in the line

```
      COMMON/A/B,C,D,
     & E,F,G
```

Comments can be made by starting the line with C, c or * , or with ! when commenting only a part of the line.

FORTRAN

Variables

The data types in FORTRAN are

- INTEGER
- REAL
- DOUBLE PRECISION
- CHARACTER
- LOGICAL
- COMPLEX

The variable declarations are done in the beginning of the program. The variables can be separated with commas:

```
INTEGER B,C,D
REAL dummy
INTEGER I
```

Usually variables starting with letter I are implicitly INTEGERs. This can be changed by command IMPLICIT,

```
IMPLICIT REAL (A-H,O-Z)
```

If no implicit types are wanted, one can use

```
IMPLICIT NONE
```

If this is used, all the variables must be declared explicitly.

In the old FORTRAN (F66) the variable names could be only up to 6 characters long. This restriction is not there any more in F77.

String variables are constructed from chars

```
CHARACTER*10 NAME
NAME = 'abc'
```

Arrays are initialized by enclosing the dimension in parenthesis

```
CHARACTER*10 NAME(50)
INTEGER IMATRX(3,3)
```

In FORTRAN the indexing starts from 1 (while in C++ it starts from 0).

```
IMATRX(3,3) = 1
```

FORTRAN

Control flow

The basic loop in FORTRAN is a DO loop

```
      DO 999 I=1,10
      ...
  999 CONTINUE
```

Here the number 999 is a label which must be uniquely chosen. No two lines can have the same label. Another way to make a DO loop is

```
      DO I=1,10
      ...
      ENDDO
```

In FORTRAN programs one can also see often IF...GOTO conditions, although using GOTO is often said to be bad programming style. Here a label is again used

```
      IF(I.eq.1) GOTO 999
```

which jumps to line 999 CONTINUE when the IF condition is fulfilled

The conditional statements are constructed with IF..THEN..ELSE..ENDIF. The logical operators are .EQ., .LT., .GT., .GE., .LE., .NE. and multiple expressions can be combined with .AND. and .OR. operators. The . separates the operator from the variables and values. Parenthesis (,) can be used to change the order how the logical expression is evaluated.

```
      IF(I.eq.1.OR.(J.GT.0.AND.k.ne.0)) THEN
        ...
      ENDIF
```

The logical operators for character and string variables are the same

.LT.	comes before
.GT.	comes after
.LE.	comes before or is equal
.GE.	comes after or is equal
.EQ.	equal
.NE.	not equal

One can use either " or ' for strings.

FORTRAN

Input/Output

Printing in FORTRAN is done with keyword PRINT, followed by asterisk, comma and what to print.

```
      PRINT *,"Hello world!"
```

The asterisk in the PRINT statement tells the compiler that the format of the printing is to be directed by the list of printed items.

```
      PRINT *,"1+2=",1+2
```

Another option for printing is using the keyword WRITE. WRITE can also be used to write directly into a file. The syntax is

```
      WRITE(6,*)"Hello world!"
```

Here the number 6 tells that the output is standard output, and asterisk is as in the PRINT statement. The format can be determined explicitly

```
      WRITE(6,999)"Hello world!"
  999 FORMAT(A12)
```

The format options are Aw for char/string, Iw for integer, Fw.d for float, Ew.d for float in exponential notation. Gw.d uses F format to values between 0.1 and 10^ d, outside that range format is like E. Here the w is an integer determining the width of the field, and d digits after the decimal point. X is used for skipping.

```
  123 FORMAT(A12,2I4,G15.6,I4)
```

Here the format is string with 12 char field, 2 integers with field 4 each, one real with field 15 and decimal precision of 6 digits, and one integer with field 4.

Analogous to WRITE is READ. It can be used to read input from standard input or from a file.

File is opened with command OPEN, and closed with CLOSE.

```
      OPEN(87,FILE='hdecay.in')
      READ(87,101)IHIGGS
  101 FORMAT(10X,I30)
```

FORTRAN

Subprograms

In FORTRAN there are two kinds of subprograms: SUBROUTINE and FUNCTION. The difference between subroutines and functions is that subroutines have no return type while functions do have. A subprogram is defined with a line like

```
        SUBROUTINE NAME(LIST)
```

or

```
        REAL FUNCTION NAME(LIST)
```

The LIST is the list of variables. The function return type can be any of the fortran data types.

The variable list LIST may contain both input and output variables. Therefore functions can be written as subroutines, with one extra variable in the list returning the function value. Subprograms end with statements RETURN and END.

Once the subroutine has been defined, it can be used via the CALL statement just like any other FORTRAN statement

```
        CALL NAME(LIST)
```

The function is used like built-in functions

```
        VALUE = NAME(LIST)
```

COMMON block

There is another way for programs to communicate: it is done through a special area of storage called COMMON. When variables are placed in COMMON blocks they may be accessible to two or more subprograms. The syntax for COMMON block definition is

```
        COMMON/name/list of variables
```

Example: part of PYTHIA COMMON block

```
    COMMON/PYDAT1/MSTU(200),PARU(200),MSTJ(200),PARJ(200)
    COMMON/PYDAT2/KCHG(500,4),PMAS(500,4),PARF(2000),VCKM(4,4)
    COMMON/PYDAT3/MDCY(500,3),MDME(8000,2),BRAT(8000),KFDP(8000,5)
    COMMON/PYDAT4/CHAF(500,2)
    CHARACTER CHAF*16
    COMMON/PYDATR/MRPY(6),RRPY(100)
    COMMON/PYSUBS/MSEL,MSELPD,MSUB(500),KFIN(2,-40:40),CKIN(200)
    COMMON/PYPARS/MSTP(200),PARP(200),MSTI(200),PARI(200)
```

FORTRAN

Simple example programs

```
PROGRAM EXAMPLE1
PRINT *,"Hello world!"
STOP
END
```

The program can be compiled with g77, or in the latest versions of gcc, gfortran. In the Makefile you can use an implicit variable $(FC), and suffix rules.

```
$ gfortran example1.F -o
example1.exe
```

In the example program the first and third lines are not mandatory, the same program can be written as

```
PRINT *,"Hello world!"
END
```

```
PROGRAM EXAMPLE2
INTEGER i
COMMON/TEST/i
i = 1
CALL INCREMENT1(i) ! i = 2
CALL INCREMENT2 ! i = 3
i = INCREMENT3(i) ! i = 4
END

SUBROUTINE INCREMENT1(input)
INTEGER input
input = input + 1
END

SUBROUTINE INCREMENT2
INTEGER i
COMMON/TEST/i
i = i + 1
END

INTEGER FUNCTION INCREMENT3(in)
INTEGER in
INCREMENT3 = in + 1
END
```

FORTRAN

Real world example: PYTHIA

PYTHIA source code and documentation can be found in
https://pythia6.hepforge.org/

Let's download the source code pythia-6.4.28.f and compile it

```
    $ gfortran pythia-6.4.28.f -c -O4
```

There are a number of warnings, but they can be ignored... Then let's create a library from the object file

```
    $ ar -r libpythia-6.4.28.a *.o
```

With command ar -t libmylib.a you can list the contents of the lib file. More about the ar command, read the man pages.

Next we need the main program. There are several examples in the pythia web page, let's take the Z0 production at LEP1 and download it. The file seems to be named main63.f. The main program consists of three parts, initialization, event loop, and output creation. The output here is histograms and a cross section table. Let's compile and link:

```
    $ gfortran main63.f -o Z0lep.exe \
-L. -lpythia-6.4.28
```

Now the executable Z0lep.exe can be run by the command: `./Z0lep.exe`

Next, let's make some modifications and have a new main. Copy the main63.f to a new name, e.g. main63b.f. Then open the file with editor. Let's change the machine from LEP to LHC. For that two lines need editing:

```
      ECM=13000.D0
      CALL PYINIT('CMS','p','p',ECM)
```

Let's also print the Z0 transverse momentum. Inside the event loop after calling PYEVNT add

```
      DO J=1,N
        IF(K(J,2).EQ.23) THEN
          PRINT *,DSQRT(P(J,1)**2+P(J,2)**2
        ENDIF
      ENDDO
```

FORTRAN

Here the variables used come from the PYTHIA common blocks. All this is documented in the PYTHIA manual.

- N number of particles in the event
- K(J,2) KF code of particle J (just a label)
- P(J,1) PX of particle J
- P(J,2) PY of particle J

The Z0 pt seems to be mostly small. It means the particle doesn't move much in the transverse plane. However, it can have quite a sizable momentum along the beam pipe (Z-coordinate).

Let's next look at the decay products. The KF code for b quarks is ± 5, sign indicating particle/antiparticle.

```
      DO J=1,N
       IF(IABS(K(J,2)).EQ.5.AND.
     & K(K(J,3),2).eq.23) THEN
       print*,dsqrt(p(j,1)**2+p(j,2)**2)
       ENDIF
      ENDDO
```

Here the K(J,3) returns the line of the parent particle, so parent particle KF code is K(K(J,3),2). This selects all b and bbar quarks which originate from Z0 decays. As you can see in the output the transverse momentum for b's is much larger than for Z0's, as expected.

Normally this kind of information is put into histograms. Prints are used only for testing and debugging. Booking histograms is done in the initialization

```
      CALL PYBOOK(3,'b pt',100,0.D0,
     & 100.D0)
```

Filling histograms inside the event loop

```
      CALL PYFILL(3,PT,1.D0)
```

and writing the histogram in a file in the output creation part.

```
      OPEN(87,file='histo.txt')
      CALL PYDUMP(1,87,0)
      CLOSE(87)
```

The resulting text file contents is documented in the PYTHIA manual.