NRC-CNRC

Elementary Mortran3

Walter R. Nelson
Stanford Linear Accelerator Center

Introductory Remarks

- EGS runs most effectively in the language in which it is written – ***Mortran3***
- ***Mortran*** is simply a string processor that produces Fortran77 code
- Users should learn ***Mortran*** in order to take advantage of the more advanced features of EGSnrc
- EGSnrc can be run directly in Fortran77, but we don't advise doing it this way. ***Mortran*** is easy to learn…and it's fun

What is Mortran?

- The term ***Mortran*** has several meanings:
  - A structured language
  - A translator for that language
  - A macro processor
- The macro-processor facility of ***Mortran*** will be discussed in a subsequent lecture
- In this lecture, we will concentrate only on those things one ***really*** needs to know in order to create a User Code for EGSnrc

Mortran as a *String Processor*

- The Mortran string processor is an ANSI standard Fortran77 code (mortran3.f)
- The job of the user is to write a ***User Code*** that gets "sandwiched" within a set of EGSnrc files

  egsnrc.macros + *User Code* + egsnrc.mortran

- This package then gets read in and ***string processed*** by mortran3.f using a set of conversion rules

… a *String Processor*

- The output is a large Fortran (.f) file consisting of the User Code plus the necessary parts of EGSnrc itself
- This large program is then compiled, linked and executed like any other Fortran code
- A script is generally employed to facilitate putting together the **“sandwich”**, creating the executable and running the job

Mortran as a Structured Language

- The primary Mortran3 reference is:

  A. J. Cook, *The Mortran3 User's Guide*,
  SLAC Internal Report CGTM-209 (1983)

  which is too difficult for the beginner (so we won't use it)

- A more useful ***Mortran*** manual entitled

  EGS User Guide to MORTRAN3

  is provided in Section 7 of the EGSnrc manual (PIRS-701)

- This lecture will not cover all the rules of Mortran, but will simply provide enough examples to illustrate the basics

… a Structured Language

- In the examples that follow we will interlace commentary to explain various features. Here are the rules for this:
  - Comments are placed inside of double quotes (e.g., “string”)
  - Comments may be inserted anywhere, except in character strings
  - Also, avoid placing them inside macros until you become an EGSpert

Example 1

```
XSUM=0.0;   X2SUM=0.0;
DO I=1,10 ["Start of DO-loop"
  X=I;
  XSUM=XSUM + X;
  X2SUM=X2SUM + X*X;
  ] "End of DO-loop"
OUTPUT XSUM,X2SUM;  (' XSUM=',E10.3,5X,'X2SUM=',E10.3);
STOP;  END;
%%    "Signals end of Mortran3 input"
```

- Statements terminate with a semicolon (**;**)
- More than one statement on a line
- Statements start in *any* column
  - **DO**-loop is simplified…just use brackets: **[** and **]**
  - No need for statement number or **CONTINUE** statement
- **OUTPUT** is easy way to say **WRITE(6, etc.)**

  …with **FORMAT** statement following immediately

Example 1 (cont.)

Mortran Code:

```
XSUM=0.0;   X2SUM=0.0;
DO I=1,10 ["Start of DO-loop"
  X=I;
  XSUM=XSUM + X;
  X2SUM=X2SUM + X*X;
  ] "End of DO-loop"
OUTPUT XSUM,X2SUM;  ('
   XSUM=',E10.3,5X,'X2SUM=',E10.3;
STOP;  END;
%%    "Signals end of Mortran3
   input"
```

Fortran Code:

```
 XSUM=0.0
      X2SUM=0.0
      DO 11 I=1,10
      X=I
      XSUM=XSUM + X
      X2SUM=X2SUM + X*X
11    CONTINUE
12    CONTINUE
      WRITE(6,20)XSUM,X2SUM
20    FORMAT('
   XSUM=',E10.3,5X,'X2SUM=',E10.3)
      STOP
      END
```

Example 2

```
IF(IRL.EQ.1) [A=B;]
ELSEIF(IRL.EQ.2) [C=D;]
ELSE [X=Y;]
Z=10;
```

- Mortran easier to read than Fortran (kind of like C)
- **IF-ELSE** statements may be nested to any depth
- Could also have written:

  `IF IRL.EQ.1 [A=B;]` or `IF IRL=1 [A=B;]`
- Caution – do not mix methods:

  `IF(IRL=1 & IRL=2)` is **OK**…but

  `IF(IRL=1.AND.IRL=2)` is **not OK**

Example 2 (cont.)

Mortran Code:

```
IF(IRL.EQ.1) [A=B;]
ELSEIF(IRL.EQ.2) [C=D;]
ELSE [X=Y;]
Z=10;
```

Fortran Code:

```
IF ((IRL.EQ.1)) THEN
A=B
ELSE IF((IRL.EQ.2)) THEN
C=D
ELSE
X=Y
END IF
Z=10
```

Loops – Other Than DO-loops

- In the following: **e** = logical expression, **[…]** = block of statements
  - **WHILE e […]**

    e is tested first – block executed if **e** true
  - **LOOP […] WHILE e**

    e is tested last – block re-executed if **e** true
  - **UNTIL e […]**

    e is tested first – block executed if **e** false
  - **LOOP […] UNTIL e**

    e is tested last – block re-executed if **e** false
  - **WHILE e […] UNTIL f**

    Test **e** first *AND* test **f** last , etc. etc. etc.

Loops (cont.)

FOR v=e TO f BY g […]

where **e**, **f** and **g** are expressions and **v** is a control variable

Note: **v** can be **REAL**, **INTEGER** or an array

- Example 3 (taken from pegs4.mortran)

```
"***NOW FILL UP MSMAP."
      FOR IS=1 TO MSTEPS-1 [
      FOR J=FSTEP(IS) TO FSTEP(IS+1)-1 [MSMAP(J)=IS;]]
      MSMAP(JRMAX)=MSTEPS;
```

Loops (cont.)

Mortran Code:

```
"***NOW FILL UP MSMAP."
      FOR IS=1 TO MSTEPS-1 [
      FOR J=FSTEP(IS) TO
   FSTEP(IS+1)-1 [MSMAP(J)=IS;]]
      MSMAP(JRMAX)=MSTEPS;
```

Fortran Code:

```
C     ***NOW FILL UP MSMAP.
            IS=1
              GO TO 993
      991     IS=IS+1
      993     IF(IS-(MSTEPS-1).GT.0)GO TO 992
              J=FSTEP(IS)
                GO TO 1003
      1001      J=J+1
      1003      IF(J-(FSTEP(IS+1)-1).GT.0)GO TO 1002
                MSMAP(J)=IS
              GO TO 1001
      1002    CONTINUE
            GO TO 991
      992   CONTINUE
            MSMAP(JRMAX)=MSTEPS
```

DO-loops

```
DO I=1,J,K,N [...]
```

is typical, where all must be integers

Also available:

```
[I=J,K,N; ...]
```

which is called the compact DO-loop

Forever-loops

LOOP [...]

or

LOOP [...] REPEAT

(the REPEAT is simply a “visual aid”)

How can you get out of loops?

Answer: Using the following statements with conditionals

```
NEXT;
EXIT;
GO TO :label:;
```

Example 4

```
:START:
LOOP [“Start of infinite loop”
  IF        e  [EXIT;] [ “Automatically exits to :HERE:”
  ELSEIF  f  [GO TO :THERE:;]
  ELSEIF  g [GO TO :Neither_HERE_nor_THERE:;]
]  “End of infinite loop”
:HERE:  “…actually, this label is not required”
:THERE:
:Neither_HERE_nor_THERE:
```

Example 5

```
DO I=1,10 [
  IF       e  [NEXT;]
  ELSEIF  f   [EXIT;]
  …miscellaneous code…
]
```

Multiple Assignment – Example 6

- Assigning value to several variables in the same statement

```
/MED(1),MED(5),MED(6)/=0;
```

produces the following Fortran

```
MED(1)=0;
MED(5)=0;
MED(6)=0;
```

Multiple Assignment – Example 7

```
/I,A(I,K),J/=SQRT(X/2.0);
```

produces the following Fortran

```
I=SQRT(X/2.0)
A(I,K)=SQRT(X/2.0)
J=SQRT(X/2.0)
```

Note: `/MED(1)/=0;` (i.e., a ***single*** assignment) will not work – use must explicitly use

```
MED(1)=0;
```

Input/Output – Example 8

```
INPUT A,B,C;   (3E15.5);
OUTPUT X,Y,Z;   ('X,Y,Z=',5X,3(F10.2,1X));
```

produces the following Fortran

```
      READ(5,10) A,B,C
10    FORMAT(3E15.5)
      WRITE(6,20) X,Y,Z
20    FORMAT('X,Y,Z=',5X,3(F10.2,1X))
```

where statement numbers 10 and 20 were generated by the ***Mortran*** (and may be initialized by the user)

Input/Output – Example 9

```
READ(5,:FMT1:) A,B,C;
:FMT1: FORMAT(3E15.5);
WRITE(6,:FMT1:) X,Y,Z;
```

produces the following Fortran

```
      READ(5,10) A,B,C
10    FORMAT(3E15.5)
      WRITE(6,10) X,Y,Z
```

which is standard in Fortran – allowing for further use of :FMT1: elsewhere