NRC-CNRC

Advanced Mortran3

Macros & Other Tricks

Walter R. Nelson
Stanford Linear Accelerator Center

Macros – Simple String Replacements

- The Mortran3 ***macro-processor*** may be regarded as a device that accepts and applies transformation rules
- The simplest macro is ***string replacement***:

```
REPLACE {pattern} WITH {replacement}
```

Note other names: pattern → template
replacement → value

- Macro definitions are not statements and therefore need not be terminated with semicolons (they will be ignored)

Example 10 – String Replacement

```
REPLACE {$MXREG} WITH {2000}
REPLACE {;COMIN/BOUNDS/;} WITH
   {;COMMON/BOUNDS/ECUT($MXREG),PCUT($MXREG),
    VACDST;}
```

The macro-processor will search both the User Code ***and*** the EGSnrc code…and will replace every occurrence of the string

```
    ;COMIN/BOUNDS/;
```

with the following Fortran

```
    COMMON/BOUNDS/ECUT(2000),PCUT(2000),VACDST
```

Assembling and EGSnrc Deck*

1) egsnrc.macros – Contains ***default*** macros
2) User Code – May contain ***override*** macros plus ***templates***
3) egsnrc.mortran – Contains ***templates***

* Also called a **“sandwich”**

Example of a *Default* Macro

In the file called egsnrc.macros we have

```
REPLACE {$MXREG} WITH {2000}
REPLACE {;COMIN/BOUNDS/;} WITH
{;COMMON/BOUNDS/ECUT($MXREG),PCUT($MXREG),
    VACDST;}
```

The string $MXREG gets replaced by the number 2000 in all code that follows the first replacement macro…unless there is an ***overriding*** macro further down in the **“sandwhich”**.

Example of an *Override* Macro

- We can add the following line to our User Code

  ```
  REPLACE {$MXREG} WITH {20}
  ```

  and this will force 20 to be used instead of 2000 as a replacement for $MXREG in all code that ***follows***.

- This applies to the User Code itself, where we might want access to ECUT and PCUT and have included the statement

  ```
  ;COMIN/BOUNDS/;
  ```

- And it also applies to…

…Example of an *Override* Macro (cont.)

…the BLOCK DATA (in egsnrc.mortran):

```
;COMIN/BOUNDS/;
DATA ECUT/$MXREG*0./,PCUT/$MXREG*0./,
     VACDST/1.E8/;
```

The appropriate COMMONs will get expanded and the initialization will get done using 20 regions (instead of 2000, the default value for EGSnrc).

Example of Templates in the EGSnrc

Throughout egsnrc.mortran you will see templates, such as

```
COMIN/BOUNDS/;
```

and

```
DO JR=1,$MXREG [MD=MED(JR);]
```

Most typically these “strings” can be recognized by a $ prefix, or by an unfamiliar combination of letters and words, such as

```
$RANDOMSET RNNO01;
```

Control Cards

- More properly called ***“processor-control directives”***, Mortran control cards may appear anywhere within the program
- There is a much more complete discussion of control cards in Section 7.6 of the EGSnrc manual (PIRS-701)
- They fall into two categories:
  - ***Free-form directives***
  - ***Column-one-restricted directives***

…Control Cards (cont.)

- ***Free-form directives*** may appear anywhere on any line and are not limited by number—we will talk them later on in this lecture
- ***Column-one-restricted-directives***, on the other hand, MUST begin with a % in column one and only ONE directive per line is recognized

…Control Cards (cont.)
%I, %F, %M and %%

- The only ***required*** “control card” is the %%, which must be the last card in the “sandwhich”. It tells the macro-processor where the ***Mortran data*** ends.
- The %In directive defines spacing in the Mortran listing
  - e.g., to indent 2 places per nest level in the Mortran listing, use %I2
- The %F and %M allows the user to switch back and forth between Mortran and Fortran (which we will show next)

…Control Cards (cont.)

```
%I2   "Indent TWO spaces in the Mortran listing"
"MAIN code (including HOWFAR and AUSGAB) follows"
STOP;  END;

%I2    "An extra one is needed (explained later)"
%F     "This is the Mortran-to-Fortran switch"
      SUBROUTINE X           ! Writing in FORTRAN now
      RETURN
      END

      FUNCTION Y             ! Still writing in FORTRAN
      RETURN
      END
%M                           "This is the Fortran-to-Mortran switch"
```

…Control Cards (cont.)

- Problem with **%F**
  - A bug in the Mortran3 processor causes statements "preceding" the **%F** to be "eaten up"
  - To avoid this, simply add a line with a **%I2** immediately before each **%F** line
  - Or, a line with a semicolon will works just as well

A Few General Items

- The ***null*** macro:

```
REPLACE {$MXREG} WITH {;}
```

Does just what it says – nothing! …well, not exactly

- Buffer overflow:
  - Happens when the working (string) buffer gets full
  - For example, when you have created too many comments
  - Remedy: <u>Insert a semicolon</u> to clear the buffer

The Disappearing Semicolon Problem

- This usually only occurs at the beginning of a User Code (e.g., with the very first COMIN statement), as we shall explain
- Assume that COMIN/BOUNDS/; is the first statement and carefully note that there is the usual (required) trailing semicolon, but not a leading one
- The macro

```
REPLACE {;COMIN/BOUNDS/;} WITH
{;COMMON/BOUNDS/ECUT($MXREG),PCUT($MXREG),
          VACDST;}
```

will simply not be able to match the pattern in this case.

- Remedy is quite simple ➔ ;COMIN/BOUNDS/;

Parameters in Macros

- The pattern part of a macro may contain up to nine **formal** parameters, denoted by the # symbol
- **Formal** parameters are also called **“dummy”** parameters
- For example, the pattern

```
{EXAMPLE#PATTERN#DEFINITION}
```

contains two **formal** parameters, and they are ***positional***

(the first # is the first **formal** parameter, etc.)

…Parameters in Macros (cont.)

- The corresponding **actual** parameters are detected and saved during the matching process
- For example, in the string

```
EXAMPLE OF A PATTERN IN A MACRO DEFINITION
        -----        ---------------
        {P1}              {P2}
```

the first **actual** parameter is the string OF A and the second **actual** parameter is the string IN A MACRO

…Parameters in Macros (cont.)

- The parameters are saved in a ***holding buffer*** until
  - All of the matching is done
  - The expansion process is completed
- The replacement part of a macro may contain an arbitrary number of occurrences of formal parameters of the form `{Pi}`, where `i=1, 2, 3,…9`
- During expansion, each **formal** parameter of the replacement part gets replaced by the i-th **actual** parameter

Example 11 – Simple Use of *Parameters*

- Consider the macro

  ```
  REPLACE {PLUS #;} WITH {{P1}={P1}+1;}
  ```

  where there is only one **formal** parameter—i.e., the single occurrence of #

- This macro would match a string in the code text, such as

  ```
  PLUS NCOUNT;
  ```

  and, after expansion, would produce

  ```
  NCOUNT=NCOUNT+1;
  ```

Example 12 – The PARAMETER Macro

- The following macro is defined in egsnrc.macros:

```
REPLACE {PARAMETER #=#;} WITH
          {REPLACE {{P1}} WITH {{P2}}}
```

- Also in egsnrc.macros are the strings:

```
PARAMETER $MXMED=10;
PARAMETER $MXREG=2000;
```

- After expansion we get the following:

```
REPLACE {$MXMED} WITH {10}
REPLACE {$MXREG} WITH {2000}
```

which, of course, are used with other macros in EGSnrc

The COMIN Macro – Revisited

- Consider the following macro in egsnrc.macros:

```
REPLACE {;COMIN/#,#/;} WITH {;COMIN/{P1}/;COMIN/{P2}/;}
```

- Upon finding the string

```
;COMIN/BOUNDS,EPCONT,STACK/;
```

the following expansion takes place

```
;COMIN/BOUNDS/; COMIN/EPCONT,STACK/;
```

which gets further expanded to

```
;COMIN/BOUNDS/; COMIN/EPCONT/; COMIN/STACK/;
```

which are then expand into their Fortran COMMONs

The $COMIN-*string* Pattern

- $COMIN-***string*** is a convenient way of defining which COMMONs to include in the various subprograms of EGSnrc
- For example, the macro

```
        REPLACE {$COMIN-ANNIH;}  WITH
{;COMIN/DEBUG,STACK,UPHIOT,USEFUL,RANDOM/;}
```

defines the COMMONs for SUBROUTINE ANNIH

and it is implemented by placing the pattern $COMIN-ANNIH at the beginning of SUBROUTINE ANNIH

Example: $COMIN-ANNIH

To be specific, the pattern $COMIN-ANNIH is located as shown:

```
SUBROUTINE ANNIH;
$COMIN-ANNIH;
(many lines of code)
RETURN;  END;
```

and it gets expanded to

```
SUBROUTINE ANNIH;
;COMIN/DEBUG,STACK,UPHIOT,USEFUL,RANDOM/;
(many lines of code)
RETURN;  END;
```

and then further expanded into…

...$COMIN-ANNIH (cont.)

```
SUBROUTINE ANNIH;
;COMIN/DEBUG/;
;COMIN /STACK/;
;COMIN /UPHIOT/;
;COMIN/USEFUL/;
;COMIN/RANDOM/;
(many lines of code)
RETURN;  END;
```

User Addition to $COMIN-*string* Macro

- Many macros of the type $COMIN-***string*** can be found in the subprograms (and BLOCK DATA) of EGSnrc
- Simply search for $COMIN throughout egsnrc.macros
- One way of adding new COMMONs to a subprogram is to add ***override code*** at the beginning of your User Code
- One can use REPLACE, but it is ***much better*** to use APPEND
- The reason why can be found in the EGSnrc manual (see APPEND vs REPLACE in the index)

…$COMIN-*string* Macros (cont.)

- Here is the recommended way of adding ***<u>your</u>*** new COMMON to an EGSnrc subprogram:

```
    APPEND {;COMIN/YOUR/;} TO {$COMIN-ANNIH;}
```

plus, of course, the necessary definition

```
    REPLACE {;COMIN/YOUR/;} WITH
        {;COMMON/YOUR/MyArray($MXMED),MyInteger;}
```

Summary to this point

- Macro changes are ***global*** changes
- They allow one to get into EGSnrc during ***run time***
- No ***permanent*** changes need to be made to EGSnrc itself
- Maintain the ***same*** EGSnrc code for everyone…only the User Codes need to be different (i.e., customized)
- User Code changes are actually in the form of ***overrides***
- **Benefit:** Changes become more ***obvious*** to all EGSnrc users

List-Generator Macros

- There are a number of what we call ***list-generator*** macros
  - Defined in `egsnrc.macros`
  - Important for user to understand how they work
- The list-generator macro

```
        $LGN(A,B,C(123))
```

produces the string

```
        A(123),B(123),C(123)
```

…List-Generator Macros (cont.)

- $LGN is often used in Block Commons
- For example

```
;COMIN/STACK/$LGN(E,X,Y,Z,U,V,W,DNEAR,WT,
      IQ,IR,LATCH($MXSTACK)),NP,NPold,LATCHI;
```

ends up becoming the following Fortran:

```
COMMON/STACK/E(40),X(40),Y(40),Z(40),U(40),
     * V(40),W(40),DNEAR(40),WT(40),IQ(40),IR(40),
     * LATCH(40),NP,NPold,LATCHI
```

Conditional REPLACEment

- Consider the two macros:

```
REPLACE {$COMPUTER} WITH {1}  “Insert 1 for RS6000, 2 for Sparc”
REPLACE {$SpecialCode} WITH {
        {SETR F=$COMPUTER}
        [IF]        {COPY F}=1        [...some lines of code]
        [ELSE]                        [...different lines of code]
}
```

- The macro works as follows:
  - $COMPUTER is defined by the user in the first macro
  - F is one of 35 user-accessable counters, 1..9, and A...Z
  - The F register is set equal to $COMPUTER
  - A “copy” of F is used in the decision-making process

The (original) $RANDOMSET Macro

- Purpose of $RANDOMSET
  - *In-line* code for the pseudo-random number generator
  - *Speed !*
- $RANDOMSET used in the following example

```
$RANDOMSET RN;          "Sample RN uniformly on (0,1)"
PHI=TwoPI*RN;           "Obtain azimuthal angle"
```

which (originally) lead to the following ***in-line*** Fortran code:

```
IXX=IXX*663608941
IX(2)=IXX
RN=DRN+0.D0
PHI=TwoPI*RN
```

...$RANDOMSET (cont.)

- Although the algorithm(s) used have changed over the years, the concept has not
- One still needs make sure COMIN/RANDOM/ is still available in any subprogram where $RANDOMSET is used
- Care should be taken to ***initialize*** the random number seed(s)
- There will more about random numbers in a subsequent lecture—it is introduced here primarily to illustrate one of several possible macro forms that have been used

Control Cards – Revisited

- Earlier we mentioned that there is a second type of control card more properly known as the ***free-form directive*** (reference: Section 7.6.2 of the EGSnrc manual)
- Examples include:
  - `!LIST;` Turn on Mortran listing (same as `%L`)
  - `!COMMENTS;` Print Mortran comments as Fortran comments (but `C` remains in column one)
  - `!LABELS n;` Reset Fortran statement-label generator to n

…Control Cards (cont.)

- `!INDENT Mn;` Set automatic indentation of Mortran source listing to `n` columns (same as `%In`)
- `!INDENT Fn;` Set automatic indentation of Fortran source listing to `n` columns
- `!INDENT Cn;` Set automatic indentation of Fortran comments to n columns (but `C` remains in column one)

“Bracketing Out” Code

- There is a nice (but undocumented) way to bracket out Mortran code—i.e, to actually leave code ***in place*** but have it ignored during the Mortran-to-Fortran process
- The “brackets” are:

  `GENERATE;` `NOGENERATE;` and `ENDGENERATE;`

- To properly implement this feature, you should first add the free-form directive

  ```
  !NEWCONDITIONAL;
  ```

  somewhere prior to performing the “bracketing”

…”Bracketing Out” Code (cont.)

!NEWCONDITIONAL;	“Place near top of User Code”
(lots of code)	
NOGENERATE;	“Don’t process the following Mortran code”
(lines of code)	
ENDGENERATE;	
GENERATE;	“Process the following Mortran code”
(lines of code)	
ENDGENERATE;	