CHAPTER - 03

Macros & Macro Processors

book Systems Programming by John J. Donovan.

Macro is an abbreviation for a sequence of operati

ex.

```
    :
ADD   1, Data
ADD   2, DATA
ADD   3, DATA
    :
ADD   1, DATA
ADD   2, DATA
ADD   3, DATA
    :
DATA  DC  '5'
```

In above program the sequence

```
ADD   1, DATA
ADD   2, DATA
ADD   3, DATA
```

occurs twice.

A macro facility permits us to attach a name to this sequence and to use this name in its place.

A macro processor effectively constitutes a seperat language processor with its own language.

Syntax to define Macro:

MACRO → start of definition.
<Name of macro> — macro name
___
___ } sequence to be abbreviated
___
MEND. — end of definition.

ex. for previous example.

```
MACRO
 INCR.
ADD  1, DATA
ADD  2, DATA
ADD  3, DATA
MEND.
```

Consider previous example with macro.

```
    .
    .
    .
  INCR
    .
    .
    .
  INCR
    .
    .
    .
DATA DC '5'
```

In this case the macro processor replaces each macro call with the lines.

```
ADD   1, DATA
ADD   2, DATA
ADD   3, DATA
```

This process of replacement is called expanding the macro.

macro is useful in reusability of code in convient way.

Macro instructions are an extension of the basic assembler language.

Most assemblers use preprocessors known as macro processors.

The outputs of the macro processors are assembly programs that become inputs to the assembler.

Features of a macro facility:-

1] Macro Instruction Arguments:-

Macro facility is capable of inserting blocks of instruction in place of macro calls.

All of the calls to any given macro will be replaced by identical blocks.

ex suppose

```
        :
ADD     1, DATA1
ADD     2, DATA2
ADD     3, DATA1
        :
ADD     1, DATA2
ADD     2, DATA2
ADD     3, DATA2
        :
        :
DATA1   DC    '5'
DATA2   DC    '10'
```

In this case the istruction sequences are very similar but not identical. The 1st sequence performs

an operation using DATA 1 as operand, the 2nd, using DATA2. They can be considered to perform the same operation with a variable parameter or dummy argument.

It is specified on the macro name line & distinguished by the ampersand (&), which is always its first character.

```
:  MACRO
   INCR &ARG
   ADD  1, &ARG
   ADD  2, &ARG
   ADD  3, &ARG
   MEND.
    :
   INCR  DATA 1
    :
   INCR  DATA2
    :
DATA1  DC  "5"
DATA2  DC  "10"
```

It is possible to supply more than one argument in a macro call.

~~MACRO~~ ex.

```
LOOP1   A    1, DATA1
        A    2, DATA2
        A    3, DATA3
             :

LOOP2   A    1, DATA3
        A    2, DATA2
        A    3, DATA1
             :

        DATA1   DC   F'5'
        DATA2   DC   F'10'
        DATA3   DC   F'15'
```

ex.

```
        MACRO
&LAB    INCR  &ARG1, &ARG2, &ARG3
&LAB    A     1, &ARG1
        A     2, &ARG2
        A     3, &ARG3
        MEND
              :
LOOP1   INCR  DATA1, DATA2, DATA3
              :
LOOP2   INCR  DATA3, DATA2, DATA1
              :
DATA1   DC    F'5'
```

OR

```
        MACRO
        INCR  &ARG1, &ARG2, &ARG3, &LAB
&LAB    A     1, &ARG1
        A     2, &ARG2
        A     3, &ARG3
        MEND
```

```
DATA  DC F'10'
DATA  DC  F'15'
        :
```

2] **Conditional Macro Expansion :-**

Here expansion of macro depends on codition.

ex.

```
        :
LOOP1   A     1, DATA1
        A     2, DATA2
        A     3, DATA3
        :
LOOP2   A     1, DATA3
        A     2, DATA2
        :
LOOP3   A     1, DATA1.
        :
DATA1   DC    F'5'
DATA2   DC    F'10'
DATA3   DC    F'15'
```

In this example, the operands, labels & the number of instructions generated change in each sequence. This program could be written as follows.

```
        :
        MACRO
&ARGO   VARY   &count, &arg1, &arg2, &arg3.
&ARGO    A     1, &ARG1
         AIF   (&count EQ 1).FINI (Test if &count=1)
         A     2, &arg2
         AIF   (&count EQ 2). FINI (Test if &count=2)
         A     3, &arg3
.FINI    MEND.
        :
        :
LOOP1   VARY   3, DATA1, DATA2, DATA3
        :
LOOP2   VARY   2, DATA3, DATA2
        :
LOOP3   VARY   1, DATA1.
        :
DATA1   DC      F'5'
DATA2   DC      F'10'
DATA3    DC     F'15'
```

macro label

AIF is a conditional branch pseudo-op. It performs an arithmetic test & branches only if the tested condition is true.

AGO - is an unconditional branch pseudo-op or

go-to statement-

3] Macro Calls within Macros

ex.

```
MACRO
ADD1   &ARG.
Load   1, &ARG
ADD    1, =F'1'
STORE  1, &ARG
MEND.

MACRO
ADDS   &ARG1, &ARG2, &ARG3
ADD1   &ARG1
ADD1   &ARG2
ADD1   &ARG3
MEND.
  :
  .

ADDS  DATA1, DATA2,
      DATA3
  :
  :

DATA1  DC  F'5'
DATA2  DC  F'10'
DATA3  DC  F'15'
```

Expanded Source (Level 1)

```
ADD1  DATA1
ADD1  DATA2
ADD1  DATA3
```

Level 2

```
Load  1, DATA1
ADD   1, =F'1'
STORE 1, DATA1
```

4] Macro Instructions defining Macros :-

If we want to define many macro we can write a single macro which when called define other macros.

ex

```
MACRO
DEFINE &SUB.
MACRO
&SUB  &Y.
  —
  —
  —
MEND.
MEND.
```

First We call.

```
DEFINE COS.
```

It define a new macro named COS: The statement expands into a new macro definition then we can called.

```
COS  DATA1.
```

Four basic tasks of any macro instruction:-

1] Recognize macro definitions:- identified by the MACRO and MEND pseudo-ops.

2] Save the definitions:-

3] Recognize calls:-

4] Expand calls & substitute arguments:-

Design of Two-Pass macro processor

Remember that a macro call substitutes text not val for parameters.

Why macro processor required two passes ?

→ Macro processor cannot expand a macro call before having found & saved the corresponding macro definition.

That's Why, In first pass it search for macro definitions, & In second pass for macro calls.

Specification of Databases :-

1] Databases for Pass-I

a) The input macro source deck.

b) The output macro source deck copy for use by pass-2

c) Macro Definition Table (MDT) - used to store the body of the macro definitions.

d) Macro Name Table (MNT) - used to store the names of defined macros.

e) Macro Definition Table Counter (MDTC) - used to indicate the next available entry in MDT.

f) Macro Name Table Counter (MNTC) - used to indicate the next available entry in the MNT

g) Argument List Array (ALA) - used to substitute index markers for dummy arguments before storing a macro definition.

2] Database for Pass-II

1] source program

2] MDT

3] MNT

4] MDTP (Macro Definition Table Pointer) - used to indicate the next line of text to be used during macro expansion.

5] Argument List Array (ALA) - used to substitute macro call arguments for the index markers in the stored macro definition.

ex.

```
      MACRO
&LAB  INCR  &ARG1,&ARG2,&ARG3
&LAB  A     1,&ARG1
      A     2,&ARG2
      A     3,&ARG3
      MEND.
```

MDT.

i) Every line of each macro definition, except the MACRO line, is stored in the MDT

ii) MEND is kept to indicate the end of the definition. and the macro name line is retained to facilitate keyword argument replacement.

MDT.

index			
:			
15	&LAB	INCR	&ARG1,&ARG2,&ARG3
16	#0	A	1,#1
17		A	2,#2
18		A	3,#3
19		MEND.	
:			

2] ALA

i) In pass-I, ALA maintain index markers for dummy arguments before storing a macro definition.

ii) Using ALA macro processor enter the macro definition instructions with index marker symbol in MDT.

index	ALA arguments
0	$LAB
1	$ARG1
2	$ARG2
3	$ARG3

iii) During pass 2 it is necessary to substitute macro call arguments for the index markers stored in the macro definition.

Thus upon encountering the call.

```
LOOP   INCR   DATA1, DATA2, DATA3
```

the macro call expander would prepare an argument list array.

ALA

Index	argument
0	LOOP1
1	DATA1
2	DATA2
3	DATA3

3] MNT

index	Name	MDT index
⋮	⋮	⋮
3	INCR	15
⋮	⋮	⋮

Macro Assembler :-

is an assembler that can perform macro substitution & expansion.