**Nested Macro Calls**

A macro body may also contain further macro definitions. However, these nested macro definitions aren't valid until the enclosing macro has been expanded! That means, the enclosing macro must have been called, before the nested macros can be called.

**Example 1:**

A macro, which can be used to define macros with arbitrary names, may look as follows:

DEFINE MACRO MACNAME

MACNAME MACRO

DB 'I am the macro &MACNAME.'

ENDM

ENDM

In order not to overload the example with "knowhow", the nested macro only introduces itself kindly with a suitable character string in ROM. The call

DEFINE Obiwan

would define the macro

Obiwan MACRO

DB 'I am the macro Obiwan.'

ENDM

and the call

DEFINE Skywalker

would define the following macro:

Skywalker MACRO

DB 'I am the macro Skywalker.'

ENDM

 nested macro instruction definition is a macro instruction definition you can specify as a set of model statements in the body of an enclosing macro definition. This lets you create a macro definition by expanding the outer macro that contains the nested definition.

All nested inner macro definitions are effectively "black boxes": there is no visibility to the outermost macro definition of any variable symbol or sequence symbol within any of the nested macro definitions. This means that you cannot use an enclosing macro definition to tailor or parameterize the contents of a nested inner macro definition.

High Level Assembler allows both inner macro instructions and inner macro definitions. The inner macro definition is not edited until the outer macro is generated as the result of a macro instruction calling it, and then only if the inner macro definition is encountered during the generation of the outer macro. If the outer macro is not called, or if the inner macro is not encountered in the generation of the outer macro, the inner macro definition is never edited. [Figure 1](https://www.ibm.com/docs/en/zos/2.2.0?topic=SSLTBW_2.2.0/com.ibm.zos.v2r2.asma400/nestmac.htm#nestmac__fgmac) shows the editing of inner macro definitions.

Figure 1. Editing inner macro definitions

┌─────────────┐

│ MACRO │

│ MAC1 ├─────────────────────────────────────┐

│ • │ │

└─────────────┘ │

┌─────────────┐ │

│ MACRO │ │

│ MAC2 ├──────────────────┐ │

│ • │ │ │

└─────────────┘ │ │

┌─────────────┐ │ │

│ MACRO │ │ │

│ MAC3 │ │ │

│ • │ Edited when │ Edited when │ Edited

when

│ • ├── MAC2 is called ├── MAC1 is called ├── definition first

│ • │ and generated │ and generated │ encountered

│ • │ │ │

│ MEND │ │ │

└─────────────┘ │ │

┌─────────────┐ │ │

│ • ├──────────────────┘ │

│ MEND │ │

└─────────────┘ │

┌─────────────┐ │

│ • ├─────────────────────────────────────┘

│ MEND │

└─────────────┘

First MAC1 is edited, and MAC2 and MAC3 are not. When MAC1 is called, MAC2 is edited (unless its definition is bypassed by an AIF or AGO branch); when MAC2 is called, MAC3 is edited. No macro can be called until it has been edited.

There is no limit to the number of nesting allowed for inner macro definitions.

* + Nested macros are macros in which definition of one macro contains definition of other macros.Consider the macro definition example given below, which is used to swap two numbers. The macro named SWAP defines another macro named STORE inside it. These type of macro are called nested macros.

SWAP MACRO &X,&Y

LDA &X

LDX &Y

STORE MACRO &X,&Y

STA &Y

STX &X

MEND MEND

Inner macro

outer macro

**Advanced Macro Facilities**

Advance macro facilities are aimed at supporting semantic expansion. 

1. Facilities for alteration of flow of control during expansion.
2. Expansion time variables 
3. Attributes of parameters.

**A)Alteration of flow of control during expansion: **

**i)Expansion time sequencing symbols (SS). **

**ii)Expansion time statements AIF, AGO and ANOP.**

1. A SS is defined by putting it in the label field of statement in the macro body.
2. It is used as operand in an AIF, AGO statement for expansion control transfer.

ii)An AIF statement has syntax

AIF <Expr><Seq symbol> 

Where Expr is relational expression involving ordinary strings, formal parameters and their attributes, and expansion time variables.  If the relational expression evaluates to true, expansion time control is transferred to the statement containing <SS>in its label field.

An AGO statement the syntax AGO

AGO<SS>

Unconditionally transfer expansion time control to the statement containing in its label field. 

An ANOP statement is written as <SS>ANOP 

Simply has the effect of defining the sequencing symbol.

**Expansion Time variables**

Expansion time variable are variables which can only be used during the expansion of macro calls.

 Local EV is created for use only during a particular macro call. 

* Global EV exists across all macro calls situated in program and can be used in any macro which has a declaration for it.
* **LCL<EV Specification>[,<EV specification>…]**
* **GBL <EV specification>[,<EV specification>…]**

<EV specification>has syntax

**<EV Name>** where EV name is ordinary string.

 Initialize EV by preprocessor statement SET.

**<EV Specification >SET<SET-Expression>**

**Attributes of parameters**

An attribute is written using the syntax:-

<attribute name> ’ <formal parameter specification>

T–Type

L-Length

S-Size

**EXAMPLE**

  MACRO

  ME &A

AIF   (L’& A EQ 1).NEXT

Other facilities for expansion time loops:

REPT statement 

Syntax: REPT <expr>

<Expr> should evaluate to a numerical value during macro expansion.  The statements between REPT and an ENDM statement would be processed for expansion <expr> number of times.

REPEAT: ): This is another facility with assembler to duplicate and

assemble the sequence a number of times during Macro Expansion.

Syntax: REPT <Expression>

MACRO

DEF\_R

LCL &P

&P SET 5

REPT 5

DC ‘&P’

& P SET &P+1

ENDM

MEND

**Output** 5,6,7,8,9,10

**Design of Macro Provessor**

# Two pass macro processor

* It is easy to design a two-pass macro processor in which all macro definitions are processed during the first pass and all macro invocation statements are expanded during second pass.
* Such a two pass macro processor cannot handle **nested macro definitions.** Nested macros are macros in which definition of one macro contains definition of other macros.
* Consider the macro definition example given below, which is used to swap two numbers. The macro named SWAP defines another macro named STORE inside it. These type of macro are called nested macros.

SWAP MACRO &X,&Y

LDA &X

LDX &Y STORE MACRO &X,&Y

STA &Y

STX &X MEND MEND

Inner macro

outer macro

# One pass macro processor

* A one-pass macro processor uses only one pass for processing macro definitions and macro expansions.
* It can handle nested macro definitions.
* To implement one pass macro processor, the definition of a macro must appear in the source program before any statements that invoke that macro.

# Data Structures involved in the design of one pass macro processor

* There are 3 main data structures involved in the design of one pass macro processor:

## DEFTAB NAMTAB ARGTAB

**Definition table (DEFTAB)**

* All the macro definitions in the program are stored in DEFTAB, which includes macro prototype and macro body statements.
* Comment lines from macro definition are not entered into DEFTAB because they will not be a part of macro expansion.
* References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.

## Name table (NAMTAB)

* The macro names are entered into NAMTAB
* NAMTAB contains pointers to beginning and end of definition in DEFTAB.

## Argument table (ARGTAB)

* The third data structure is an argument table (ARGTAB), which is used during expansion of macro invocations.
* When macro invocation statements are recognized, the arguments are stored in ARGTAB according to their position in argument list.
* As the macro is expanded, arguments from ARGTAB are substituted for the corresponding parameters in the macro body.
* Example: Consider the following source code

SUM MACRO &X,&Y

LDA &X

MOV B

LDA &Y

ADD B MEND START

LDA 4500

ADD B

SUM P,Q

LDA 3000

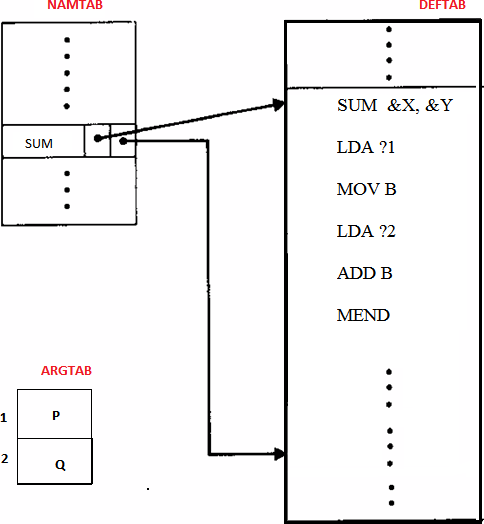
…………. END

* + When the macro definition for SUM is encountered, the macro name SUM along with its parameters X and Y are entered into DEFTAB. Then the statements in the body of macro is also entered into DEFTAB. The positional notation is used for the parameters. The parameter &X has been converted to ?1, &Y has been converted to

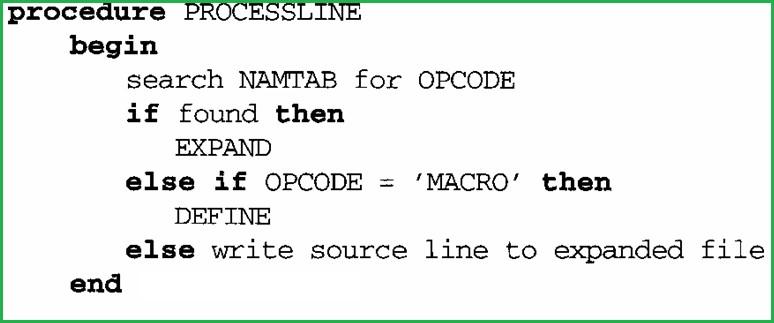
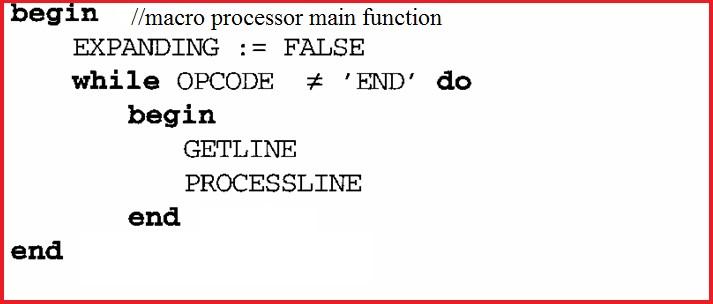
?2.

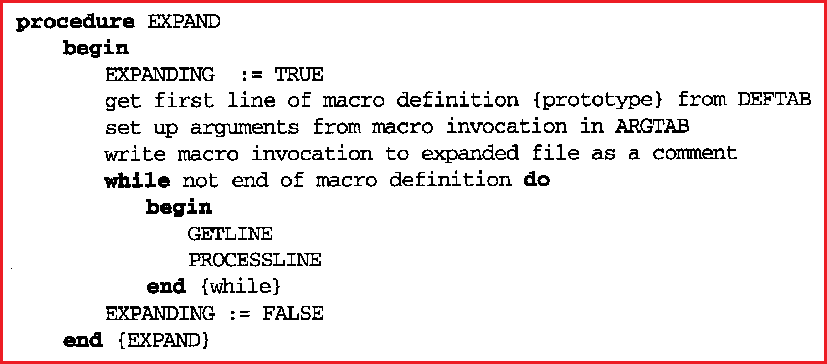
* + The macro name SUM is entered into NAMTAB and the beginning and end pointers are also marked.
  + On processing the input code, opcode in each statement is compared with the NAMTAB, to check whether it is a macro call. When the macro call SUM P,Q is recognized, the arguments P and Q will entered into ARGTAB. The macro is expanded by taking the statements from DEFTAB using the beginning and end pointers of NAMTAB.
  + When the ?n notation is recognized in a line from DEFTAB, the corresponding argument is taken from ARGTAB.

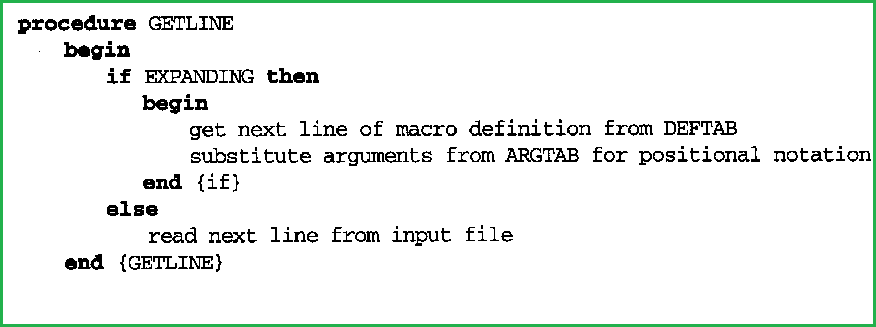
**Figure shows the different data structures used**



# Algorithm for one pass macro processor







## Explanation of algorithm

* + The algorithm uses 5 procedures
    - MACROPROCESSOR (main function)
    - DEFINE
    - EXPAND
    - PROCESSLINE
    - GETLINE

MACROPROCESSOR (MAIN function)

* + This function initialize the variable named EXPANDING to false.
  + It then calls GETLINE procedure to get next line from the source program and PROCESSLINE procedure to process that line.
  + This process will continue until the END of program.

PROCESSLINE

* + This procedure checks
    - If the opcode of current statement is present in NAMTAB. If so it is a macro invocation statement and calls the procedure EXPAND
    - Else if opcode =MACRO, then it indicates the beginning of a macro definition and calls the procedure DEFINE
    - Else it is identified as a normal statement(not a macro definition or macro call) and write it to the output file.

DEFINE

* + The control will reach in this procedure if and only if it is identified as a macro definition statement.Then:
    - Macro name is entered into NAMTAB
    - Then the macro name along with its parameters are entered into DEFTAB.
    - The statements in body of macro is also enterd into DEFTAB. References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.
    - Comment lines from macro definition are not entered into DEFTAB because they will not be a part of macro expansion.
    - Store in NAMTAB the pointers to beginning and end of definition in DEFTAB.
  + To deal with Nested macro definitions DEFINE procedure maintains a counter named LEVEL.
    - When the assembler directive MACRO is read, the value of LEVEL is incremented by 1
    - When MEND directive is read, the value of LEVEL is decremented by 1
    - That is, whenever a new macro definition is encountered within the current definition, the value of LEVEL will be incremented and the while loop which is used to process the macro definition will terminate only after the value of LEVEL =0. With this we can ensure the nested macro definitions are properly handled.

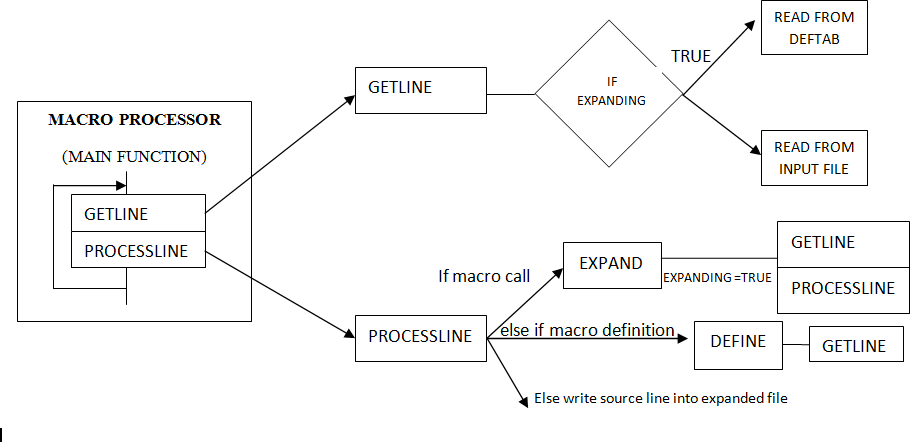
EXPAND

* + The control will reach in this procedure if and only if it is identified as a macro call.
  + In this procedure, the variable EXPANDING is set to true. It actually indicates the GETLINE procedure that it is going to expand the macro call. So that GETLINE procedure will read the next line from DEFTAB instead of reading from input file.
  + The arguments of macro call are entered into ARGTAB.
  + The macro call is expanded with the lines from the DEFTAB. When the ?n notation is recognized in a line from DEFTAB, the corresponding argument is taken from ARGTAB.

GETLINE

* + This procedure is used to get the next line.
  + If EXPANDING = TRUE, the next line is fetched from DEFTAB. (It means we are expanding the macro call)
  + If EXPANDING = False, the next line is read from input file.

## Flow Diagram of a one pass macroprocessor



PROCESSLINE

* + This procedure checks
    - If the opcode of current statement is present in NAMTAB. If so it is a macro invocation statement and calls the procedure EXPAND
    - Else if opcode =MACRO, then it indicates the beginning of a macro definition and calls the procedure DEFINE
    - Else it is identified as a normal statement(not a macro definition or macro call) and write it to the output file.

DEFINE

* + The control will reach in this procedure if and only if it is identified as a macro definition statement.Then:
    - Macro name is entered into NAMTAB
    - Then the macro name along with its parameters are entered into DEFTAB.
    - The statements in body of macro is also enterd into DEFTAB. References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.
    - Comment lines from macro definition are not entered into DEFTAB because they will not be a part of macro expansion.
    - Store in NAMTAB the pointers to beginning and end of definition in DEFTAB.
  + To deal with Nested macro definitions DEFINE procedure maintains a counter named LEVEL.
    - When the assembler directive MACRO is read, the value of LEVEL is incremented by 1
    - When MEND directive is read, the value of LEVEL is decremented by 1
    - That is, whenever a new macro definition is encountered within the current definition, the value of LEVEL will be incremented and the while loop which is used to process the macro definition will terminate only after the value of LEVEL =0. With this we can ensure the nested macro definitions are properly handled.

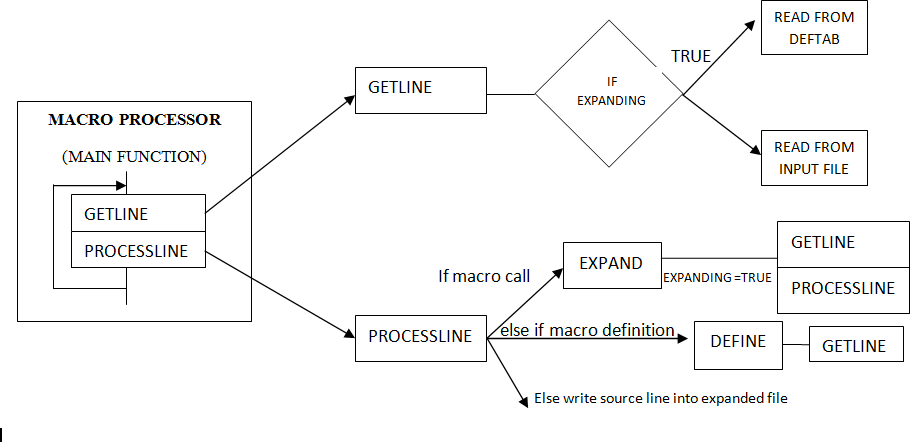
EXPAND

* + The control will reach in this procedure if and only if it is identified as a macro call.
  + In this procedure, the variable EXPANDING is set to true. It actually indicates the GETLINE procedure that it is going to expand the macro call. So that GETLINE procedure will read the next line from DEFTAB instead of reading from input file.
  + The arguments of macro call are entered into ARGTAB.
  + The macro call is expanded with the lines from the DEFTAB. When the ?n notation is recognized in a line from DEFTAB, the corresponding argument is taken from ARGTAB.

GETLINE

* + This procedure is used to get the next line.
  + If EXPANDING = TRUE, the next line is fetched from DEFTAB. (It means we are expanding the macro call)
  + If EXPANDING = False, the next line is read from input file.

## Flow Diagram of a one pass macroprocessor



The Remove Directive: Remove Macro from MDT(Macro definition table).

• The IRP Directive: (Indefinite repeat)

It is used by the assembler to repeatedly duplicate and assemble the sequence a number of times determined by a compound parameter.

**IRP Example**

MACRO

MAX\_X &P,&Q

IRP &P

ADD REG1,&P

IRP

MEND

**MACRO CALL** :-  MAX\_X(A,B,#3),H

on Execution

ADD REG1,A

ADD REG1,B

ADD REG1,#3