System Programming and Operating Systems Lab

**ASSIGNMENT 8**

**Name:Maulick Metkar**

**Roll No: 323045**

**Batch: C2**

# Aim:

To design suitable data structure and implement pass-1 of two pass macro-processor using OOP features in Java.

# Objectives:

To design pass-1 of two pass macro-processor.

# Theory:

A macro processor is a program that copies a stream of text from one place to another, making a systematic set of replacements as it does so. Macro processors are often embedded in other programs, such as assemblers and compilers. Sometimes they are standalone programs that can be used to process any kind of text.

A macro instruction (macro) is a notational convenience for the programmer. It allows the programmer to write shorthand version of a program (module programming). A Macro represents a commonly used group of statements in the source programming language.A macro instruction (macro) is a notational convenience for the programmer. It allows the programmer to write shorthand version of a program (module programming)

The macro processor replaces each macro instruction with the corresponding group of source language statements (expanding) Normally, it performs no analysis of the text it handles.It does not concern the meaning of the involved statements during macro expansion. The design of a macro processor generally is machine independent.

Two new assembler directives are used in macro definition

MACRO: identify the beginning of a macro definition MEND: identify the end of a macro definition

Usually in the Pass 1 of the two pass Macro-Processor,Macros are defined and in the second pass of the macro-processor,macros are expanded.Data Structures required in the Pass 1 of the

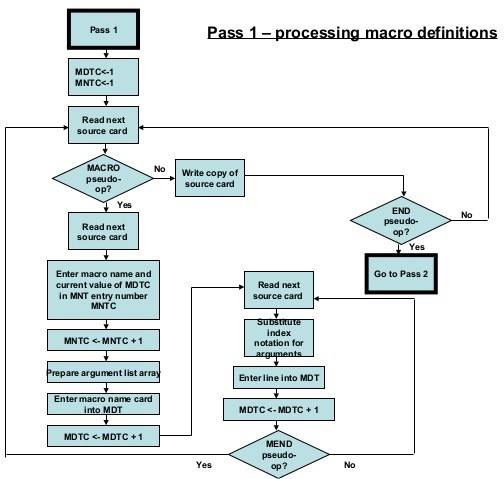
Macro-Processor are as follows:-

* 1. Input Source File
  2. Intermediate File
  3. MNT (Macro Name Table)
  4. MDT (Macro Defination Table)
  5. MNT Pointer
  6. MDT Pointer

# Algorithm:

* 1. begin macro processor
  2. EXPANDING : = FALSE
  3. while OPCODE END do
  4. begin
  5. GETLINE
  6. PROCESSLINE
  7. end while
  8. end macro processor
  9. procedure PROCESSLINE
  10. begin
  11. search MNT for OPCODE
  12. if found then
  13. EXPAND
  14. else if OPCODE = MACRO then
  15. DEFINE
  16. else write source line to expanded file
  17. end PROCESSLINE

# Flowchart:



1. **Code:**

;− − − − − − − − − − − −

import java . i o . ;

∗

import java . lang . S tri ng ;

import java . u t i l . Scanner ;

publ i c c l a s s Macro Processor s t a t i c i nt mdtp=0 ,mntp=0;

{

publ i c s t a t i c void main ( S tri ng [ ] args ) throws Exception

{

Fi l e f=new Fi l e (” program . txt ” ) ; Scanner s=new Scanner ( f ) ;

S tri ng tokens [ ] ; // Array f o r s to r i n g tokens generated S tri ng MDT Def , m ac ro de f l i ne ;

MNT[ ] mnt obj=new MNT[ 5 ] ; // Array o f o b j e c t s f o r MNT

MDT[ ] mdt obj=new MDT[ 5 ] ; // Array o f o b j e c t s f o r MDT

S tri ng [ ] ALA; // Data member o f MNT Class while ( s . has NextLine ( ) )

{

m ac ro de f l i ne=s . next Line ( ) ; // read l i n e in m ac ro de f l i ne v ari ab l e tokens=m ac ro de f l i ne . s p l i t (” ” ) ; // s p l i t the l i n e on space

f o r ( i nt i =0; i *<*tokens . l ength ; i++)

{

i f ( tokens [ i ] . equals (”MACRO” )) // i f macro

{

// MNT

− − − − − − − −

mnt obj [ mntp]=new MNT( mdtp , mntp ) ; // c re ate MNT o bje c t mnt obj [ mntp ] . name=tokens [ i + 1 ] ; // name o f macro

// mnt obj [ mntp ] . pri nt ( ) ; // pri nt MNT Contents

// MDT

− − − − − − − − − − − − − − − − − − − − − − − − − − − − − − −

MDT Def=m ac ro de f l i ne ; // s to re l i n e in MDT Def

while ( true ) // read u n t i l MEND not found

S tri ng x=s . next Line ( ) ; // read macro d e f i n t i o n l i n e by l i n e i f ( x . equals (”MEND” ))

{

{

MDT Def=MDT Def . concat ( x ) ; //MDT a l s o contains MEND statement

mdtp++; // increment mdtp f o r next macro d e f i n i t i o n break ;

}

MDT Def=MDT Def+” n ” ; // add new l i n e to MDT contents

\

MDT Def=MDT Def . concat ( x ) ; // put next statement o f macro d e f i n i t i o n

mdtp++;

}

i f (MDT Def . contai ns (”X” )) // Replace argument with #1 , # 2 . . . . . .

MDT Def=MDT Def . re p l a c e A l l (”X” ,”#0 ”);

i f (MDT Def . contai ns (”Y” ))

MDT Def=MDT Def . re p l a c e A l l (”Y” ,”#1 ”);

MDT Def=MDT Def . r e p l a c e F i r s t (”#0” ,”X” ) ; // Macro name l i n e contai ns parameters a i s

MDT Def=MDT Def . r e p l a c e F i r s t (”#1” ,”Y” ) ;

mdt obj [ mntp]=new MDT(MDT Def ) ; // c re ate mdt o bje c t and s to re contents

// mdt obj [ mntp ] . pri nt ( ) ; // pri nt mdt contents

// ALA

− − − − − − − −

ALA=t okens [ i + 2 ] . s p l i t ( ” , ” ) ; // arguments at 3 rd index o f token arra mnt obj [ mntp ] . setALA (ALA) ; // s e t ALA f o r each macro

mntp++;

}

}

pri nt ( mnt obj , mdt obj ) ; s . c l o s e ( ) ;

}

}

s t a t i c void pri nt (MNT[ ] obj ,MDT[ ] obj 1 )

{

System . out . p r i n t l n(”− − − − −MNT−−−−−−−−”);

f o r ( i nt i =0; i *<*mntp ; i++) obj [ i ] . pri nt ( ) ;

System . out . p r i n t l n ( ) ;

System . out . p r i n t l n(” MDT ”); f o r ( i nt i =0; i *<*mntp ; i++)

− − − − − −−−−−−−−

obj 1 [ i ] . pri nt ( ) ;

System . out . p r i n t l n ( ) ;

System . out . p r i n t l n(” ALA ”); f o r ( i nt i =0; i *<*mntp ; i++)

− − − − −−−−−−−−

obj [ i ] . printALA ( ) ;

System . out . p r i n t l n ( ) ;

}

}

c l a s s MNT

{

i nt no , address ; S tri ng name ;

S tri ng [ ] ALA;

MNT( i nt mdtp , i nt mntp) no=mntp ;

{

i f ( mntp==0)

address=mdtp ; // Fi r s t Macro w i l l be at 0 th p o s i t i o n in MDT e l s e

address=mdtp+1; name=””;

}

publ i c void pri nt ()

{

System . out . pri nt ( t h i s . no + ” t”+ t h i s . name + ” t ” + t h i s . address ) ; System . out . p r i n t l n ( ) ;

\ \

}

publ i c void setALA ( S tri ng [ ] obj ) t h i s .ALA=o bj ;

{

}

publ i c void printALA ()

{

System . out . p r i n t l n (” 0 ” + t h i s .ALA[ 0 ] ) ; System . out . p r i n t l n (” 1 ” + t h i s .ALA[ 1 ] ) ;

}

}

c l a s s MDT

{

S tri ng def ;

MDT( S tri ng d e f i n t i o n ) t h i s . def=d e f i n t i o n ;

{

}

publ i c void pri nt ()

{

System . out . pri nt ( t h i s . def ) ; System . out . p r i n t l n ( ) ;

}

}

# Output:

1. **Conclusion:**

Through this assignment we understood the working of Pass 1 of Macro-Processor in which the macro is defined in the form of 2 major data structures which are MNT(Macro Name Table) and MDT(Macro Defination Table).