EXPERIMENT NO 2

Date of performance :
Date of submission :

Aim: to implement a two pass macro preprocessor.

Software Used: c, c++, java language

Theory:

Macro is a set of instructions, it contains: Macro definition, macro call. Macro definition contains 3 parts:

1. Macro body
2. header of the macro.
3. footer

Macro preprocessor, like an assembler, scans and processes lines of a text. In assembly language, lines are interrelated by addressing: a line can refer to another by its address or name, which must be available to the assembler. Moreover, the address assigned to each line depends upon proceeding lines, upon their address, and possibly upon their contents as well. If we consider macro definitions to constitute integral entities, we may say that the lines of our macro language are not so closley interrelated. Macro definitions refer to nothing outside themselves, and macro calls refer only to macro definitions.

Tasks performed by a macro preprocessor:
1. Recognize macro definitions
2. Save the definitions
3. Recognize macro calls
4. Expand calls and substitute arguments

Specification of database

Pass 1 database
1) the i/p macro source desk
2) the o/p macro source desk copy for use by pass 2
3) The macro definition table(MDT) use to store the body of the macro definition
4) The macro name table(MNT) use to store the names of defined macro
5) The macro definition table counter(MDTC) used to indicate the next available entry in the MDT
6) The macro name table counter (MNTC) used to indicate the next available entry in the MNT
7) The argument list array (ALA) used to substitute index markers for dummy arguments before storing a macro definition

Pass 2 database
1) the copy of the i/p macro source deck
2) the o/p expanded source deck to be used as i/p to the assembler
3) the macro definition table(MDT) create by pass 1
4) the macro name table (MNT) create by pass 1
5) the macro defination table pointer(MDTP) used to indicate macro call arguments for the index markers in the stored macro defination

problem defination
/* input code for macroprocess */
ADD A

```
MACRO ADD1 &ARG
LOAD ARG
MEND
MACRO PQR  &A,&B,&C
ADD B
READ C
READ A
MEND
MACRO LMN
LOAD C
MEND
LOAD  B
PQR  5,3,2
ADD1     1
LMN
SUB     C
ENDP
```

MACRO NAME TABLE

Macro name	No. of parameter
ADD1	1
PQR	3
LMN	0

MAVRO DEFINATION TABLE(MDT)

INDEX	INSTRUCTION
1	LOAD #1
2	MEND
3	ADD #2
4	READ #3
5	READ #1
6	MEND
7	LOAD C
8	MEND

FORMAL VS POSITIONAL PARAMETER LIST

MACRO NAME	FOMRAL PARAMETER	POSTIONAL PARAMETER
ADD1	&ARG	#1
PQR	&A	#1
PQR	&B	#2
PQR	&C	#3

ACTURAL VS POSITIONAL PARAMETER LIST

MACRO NAME	ACTUAL PARAMETER	POSTIONAL PARAMETER
PQR	5	#1
PQR	3	#2
PQR	2	#3
ADD1	1	#1

EXPANDED CODE

INSTUCTION CODE
ADD A
LOAD B
ADD 2
READ 3
READ 5
LOAD 1
LOAD 2
SUB C
ENDP

Flowchart:
Pass 1

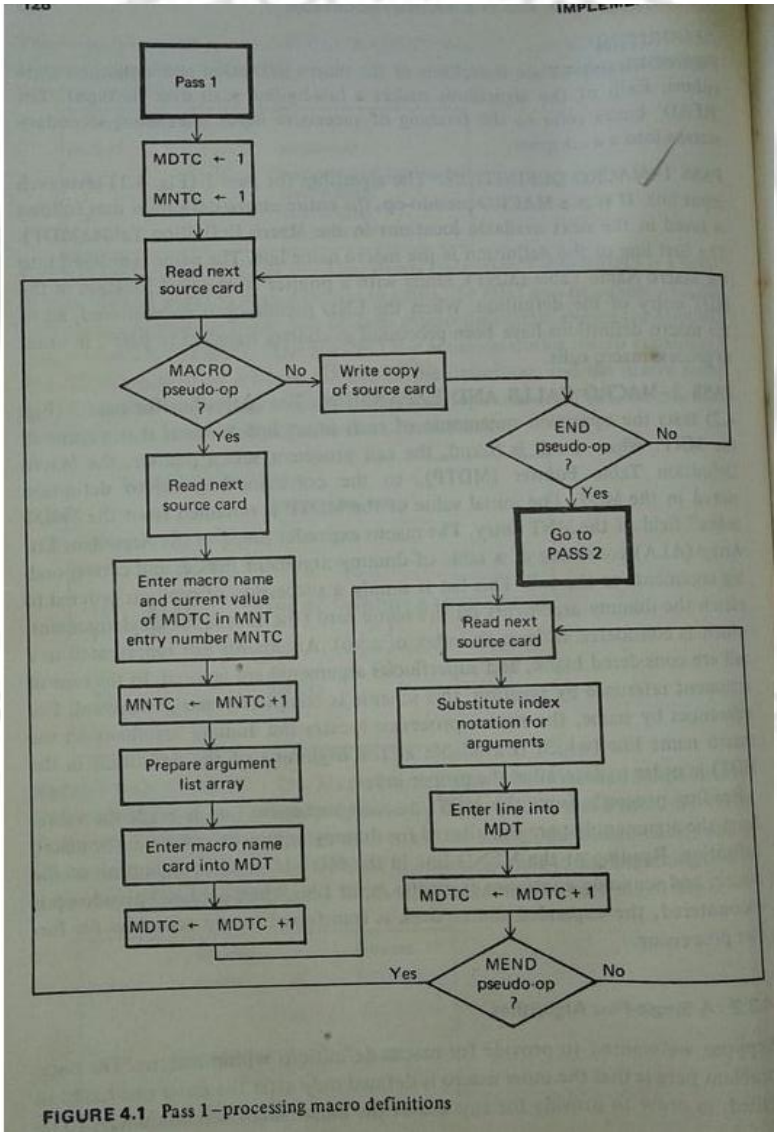

FIGURE 4.1 Pass 1–processing macro definitions

Pass 2:

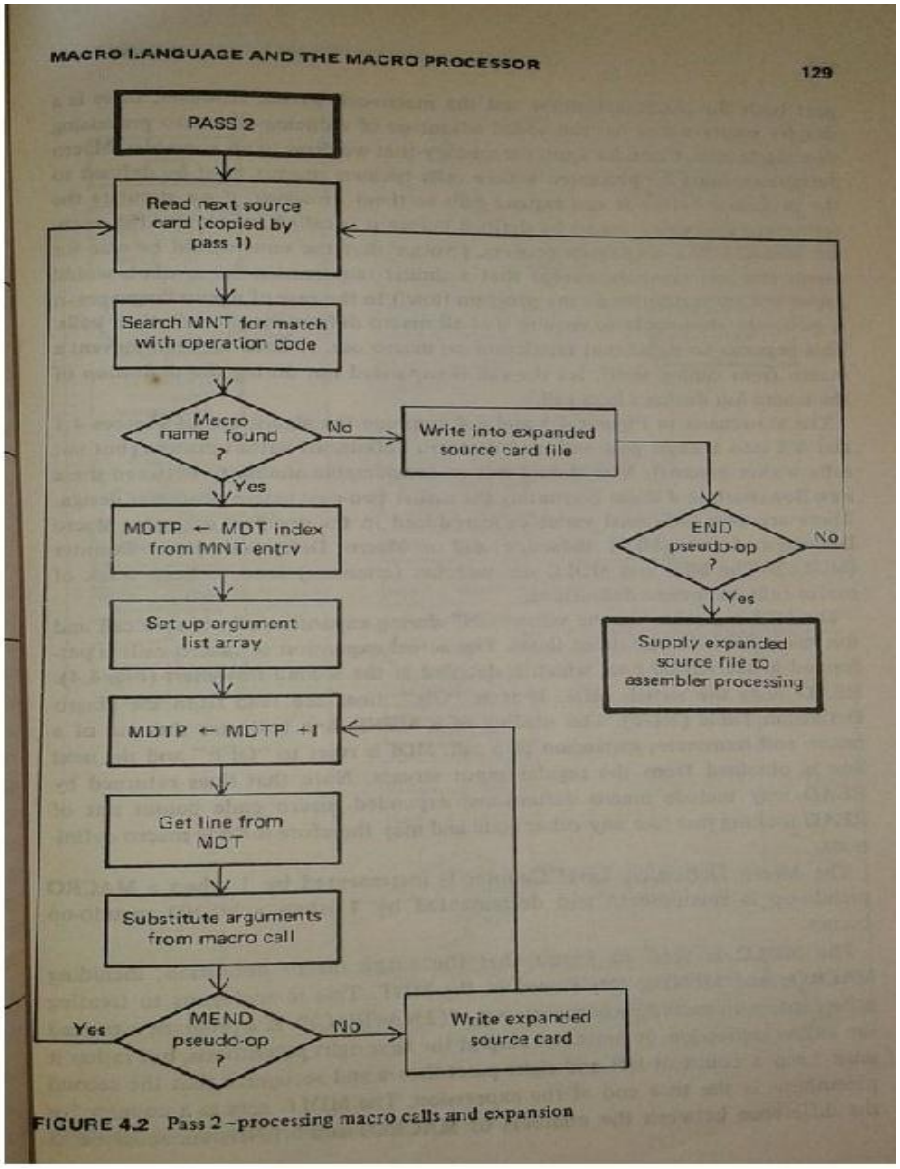

FIGURE 4.2 Pass 2 –processing macro calls and expansion

Program: TO IMPLEMENT **macro processor**

```
import java.io.*;
class Macroprocessor
{
	public static void main(String args[]){
		String code[][]={{"ADD","A","","",""},
		{"MACRO","ADD1","&ARG","",""},
		{"LOAD","ARG","","",""},
		{"MEND","","","",""},
		{"MACRO","PQR","&A","&B","&C"},
		{"ADD","B","","",""},
		{"READ","C","","",""},
		{"READ","A","","",""},
		{"MEND","","","",""},
		{"MACRO","LMN","","",""},
		{"LOAD","C","","",""},
		{"MEND","","","",""},
		{"LOAD","B","","",""},
		{"PQR","5","3","2",""},
		{"ADD1","1","","",""},
		{"LMN","","","",""},
		{"SUB","C","","",""},
```

```
        {"ENDP","","","","",}
        };
    String mn[]=new String[3],fpmn[]=new String[4],fp[]=new String[4],pp[]=new String[4];
    int parameter[]=new int[3],c=0,d=0,e=0,l=0;
    for(int i=0;i<18;i++) {
        if(code[i][0].equals("MACRO")){
            mn[c]=code[i][1];
            for(int j=2;j<5;j++){
                if(code[i][j]!=""){
                    fpmn[e]=code[i][1];
                    fp[e]=code[i][j];
                    pp[e++]="#"+(++d);}
            } parameter[c++]=d;
            d=0;
    }
}
    String apmn[]=new String[4],ap[]=new String[4],app[]=new String[4];
    c=1;
    d=0;
    for(int i=0;i<18;i++)
    {
        for(int j=0;j<mn.length;j++)
        {
            if(code[i][0].equals(mn[j])&&code[i][1]!="")
            {
                while(code[i][c]!="")
                {
                    apmn[d]=code[i][0];
                    ap[d]=code[i][c];
                    app[d]="#"+c;
                    c++;
                    d++;
                }
                c=1;
            }
        }
    }
    System.out.println("macro name table");
    System.out.println("_____");
    System.out.println("macro name no. of parameter");
    System.out.println("_____");
    for(int i=0;i<mn.length;i++)
    {System.out.println(mn[i]+"\t\t" +parameter[i]); }
    System.out.println("----------\n \n");
    System.out.println("macro definition table");
    System.out.println("----------");
    System.out.println("index \t instruction");
    System.out.println("----------");
    int index=1, i=0;
    while(i<18) {
        if(code[i][0].equals("MACRO")){
            i++;
        while(code[i][0]!="MEND"){
            for(int j=0; j<fp.length; j++){
                if(("&"+code[i][1]).equals(fp[j])){
```

```
                                        System.out.println((index++)+"\t"+code[i][0]+" "+pp[j]);
                                        break;                                          }
                        }
                        i++;}
                System.out.println((index++)+"\t MEND");
                }else{
                        i++;}
         }
        System.out.println("-------------------\n \n");
        System.out.println("Formal Vs Positional Parameter list");
        System.out.println("---------------------------");
        System.out.println("Macro Name \t Formal parameter \t Positional Parameter");
        System.out.println("---------------------------------");
        for(i=0; i<fpmn.length;i++)
                System.out.println(fpmn[i]+"\t\t"+fp[i]+"\t\t\t"+pp[i]);
        System.out.println("-------------------------------------");
        System.out.println("actual Vs positional parameter");
        System.out.println("-------------------------------------");
        System.out.println("macro name\t actual parameter\tpositional parameter");
        System.out.println("---------------------------------");
        for(i=0;i<apmn.length;i++)
{System.out.println(apmn[i]+"\t\t"+ap[i]+"\t\t\t"+app[i]);}
System.out.println("--------------------------------\n\n");
  String pvalue[][]=new String[4][2];
for(i=0;i<4;i++)
{  for(int j=0;j<4;j++) {
if (fpmn[i].equals(apmn[j])&pp[i].equals( app[j]))
{   pvalue[i][0]=fp[i];pvalue[i][1]=ap[j];break;}}}
System.out.println("expanded code");
System.out.println("-----------------");System.out.println("instruction code");
System.out.println("----------------");
i=0;
while(i<18){
if(code[i][0].equals("ADD")||code[i][0].equals("SUB")||code[i][0].equals("ENDP")||code[i][0].equals("LOAD")
{System.out.println(code[i][0]+""+code[i][1]);
i++; }
else if(code[i][0].equals("MACRO"))
{i++;
while(code[i][0]!="MEND"){i++;}
i++; }
else{
int k=0;
while(k<18)
{  if (code[k][1].equals(code[i][0]))
{  k++;
while(code[k][0]!="MEND"){
for(l=0;l<4;l++)
  if(("&"+code[k][l]).equals(pvalue[l][0]))
   System.out.println(code[k][0]+""+pvalue[l][l]);}
k++;  }k++; }k++; i++; }
} } }
```

OUTPUT-

```
macro name table
-----------------------------------------
macro name no. of parameter
-----------------------------------------
```

```
ADD1        1
PQR         3
LMN         0
-----------------------------------------------
macro definition table
-------------------------------
index    instruction
-----------------------------------
1   LOAD #1
2    MEND
3   ADD #2
4   READ #3
5   READ #1
6    MEND
7   LOAD #3
8    MEND
-------------------------
Formal Vs Positional Parameter list
-------------------------------------------------------------------------------------
Macro Name   Formal parameter    Positional Parameter
-------------------------------------------------------------------------------
ADD1              &ARG                  #1
PQR               &A                   #1
PQR               &B                   #2
PQR               &C                   #3
-------------------------------------------------------------------------------------
actual Vs positional parameter
-------------------------------------------------------------------------------
macro name   actual parameter   positional parameter
-------------------------------------------------------------------------------
PQR               5                   #1
PQR               3                   #2
PQR               2                   #3
ADD1               1                    #1
-------------------------------------------------------------------------
expanded code
-------------------------
instruction code
-------------------------
ADDA
LOADB
```

Conclusion: Thus we have implemented two pass macro preprocessor.

SIGN AND REMARK

DATE

R1	R2	R3	R4	R5	Total (15 Marks)	Signature