***PRACTICAL NO:03***

***Aim:*** To Study and implement 2 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. Macro processors have been used for language expansion (defining new language constructs that can be expressed in terms of existing language components), for systematic text replacements that require decision making, and for text re-formatting (e.g. conditional extraction of material from an HTML file).

The following four major tasks are done by macro processor.

1. Recognize macro definition by identifying MACRO and MEND pseudo-ops.

2. Save these definitions which are required for macro expansion process

3. Recognize macro calls by identifying macro name which appears as operation mnemonic.

4. Finally expand macro call and substitute arguments for the dummy arguments.

**Databases Used**

* Pass1 Databases:

1. The input source program

2. The output source deck to be used by pass 2

3. The Macro Definition Table (MDT) : this table stores the macro definition.

4. The Macro name Table(MNT) : This table stores macro names which are defined.

5. The Macro Definition Table Counter (MDTC), which used to indicate next available entry in MDT.

6. The Macro Name Table Counter (MNTC), which used to indicate next available entry in MNT.

7. Argument list Array (ALA), which stores the index markers for dummy arguments.

* Pass 2 Databases:

1. Copy of input source program

2. The output for assembler: the expanded source code

3. The Macro Definition Table (MDT) :created by pass1.

4. The Macro name Table(MNT) : This created by pass1

5. The Macro Definition Table Pointer (MDTP), which used to indicate next line used in macro expansion.

6. Argument list Array (ALA), which is used to substitute arguments for index markers.

**Format of databases:**

We will use following example to discuss the format of all databases.

MACRO

&LAB ADDM &ARG1, &ARG2, &ARG3 A 1,&ARG1

A 2,&ARG2

A 3,&ARG3

MEND

………………………………

LOOP ADDM D1, D2, D3

………………………………..

1. **Argument List Array (ALA)**

This is an array which stores all arguments used in macro definition. Each argument is assigned an index marker.

Consider following macro call,

LOOP ADDM D1, D2, D3

The ALA for this would be:

|  |  |
| --- | --- |
|  | 8 bytes per entry |
| Index | Argument |
| 0 | “Loopbbbb” |
| 1 | “D1bbbbbb” |
| 2 | “D2bbbbbb” |
| 3 | “D3bbbbbb” |

1. **Macro Definition Table (MDT)**

This table stores each line of macro definition in it except the line for MACRO pseudo-op.The MDT for above example is :

|  |  |
| --- | --- |
|  | 80 bytes per entry |
| Index | Card |
| …… | ……………. |
| 10 | &LABADDM &ARG1, &ARG2, &ARG3 |
| 11 | A 1,&ARG1 |
| 12 | A 2,&ARG2 |
| 13 | A 3,&ARG3 |
| 14 | MEND |
| ……. | ……… |

1. **Macro Name Table (MNT)**

Each entry in MNT has following fields: Index:

Name: It is a macro name

MDT Index: This is an index from MDT which indicates the line number from which macro definition is stored in MDT.

The MNT for above example is :

|  |  |  |
| --- | --- | --- |
|  | 8 bytes per entry |  |
| index | Name | MDT index |
| …… | ………………. | ………………. |
| 3 | “ADDMbbbb” | 10 |
| ….. | …………….. | …………. |

**Algorithm**

**Pass 1 processing:**

1. Initialize MDTC and MNTC as MDTC=MNTC=1
2. Read a line from input source card.
3. If it is MACRO pseudo-op then
   1. Read the next line which will be the macro name line. Enter the macro name in MNT with current value of MDTC.
   2. Increment the value of MNTC by 1
   3. Then the argument list array is prepared for the arguments found in the macro name line.
   4. The macro name cared is also inserted in MDT.
   5. Increment the value of MDTC by 1
   6. Read the next line from source card
   7. Substitute the index markers for arguments from ALA prepared in previous step and then enter this line into MDT
   8. Increment the value of MDTC by 1
   9. Check whether it is MEND pseudo-op then
      1. Go to step 2.

else

* + 1. Go to step 3.6

1. Else, if it is not MACRO pseudo-op then simply write the line in the copy of source deck prepared for pass2.
2. Check if it is END pseudo-op then
   1. Go to step 6

else

5.2 Go to step 2.

1. Stop

**Pass 2 processing:**

1. Read a line from copy of source deck prepared by pass1.
2. Search the MNT for match with the op-code.
3. If macro name found in MNT then
   1. Initialize MDTP with the value of MDT index from the MNT entry. Thus MDTP holds the location from which macro definition is stored in MDT.
   2. Argument list array is prepared for the arguments found in macro call.
   3. Increment value of MDTP by 1.
   4. Read the next line from MDT and substitute the arguments from macro call for the dummy arguments.
   5. If it is MEND pseudo-op then
      1. Go to step 1.

else

* 1. Write this line into expanded source code.
  2. Increment the value of MDTP by 1
  3. Go to step 3.4 in order to process remaining statements from MDT.

1. Else if match for macro name is not found in MNT then write the line into expanded source code.
2. If it is END pseudo-op then
3. Supply this expanded source code copy to assembler.
4. Go to step 6

else

5.2 Go to step 1.

1. Stop

***Conclusion:*** Thus 2 pass macro processor studied and implemented successfully.

***Program:***

import java.util.\*;

import java.io.\*;

class twopassmacro

{

static String mnt[][]=new String[5][3]; //assuming 5 macros in 1 program

static String ala[][]=new String[10][2]; //assuming 2 arguments in each macro

static String mdt[][]=new String[20][1]; //assuming 4 LOC for each macro

static int mntc=0,mdtc=0,alac=0;

public static void main(String args[])

{

pass1();

System.out.println("Macro Table(MNT)");

display(mnt,mntc,3);

System.out.println("Argument Array(ALA) for Pass1");

display(ala,alac,2);

System.out.println("Macronition Table(MDT)");

display(mdt,mdtc,1);

pass2();

System.out.println("Argument Array(ALA) for Pass2");

display(ala,alac,2);

}

static void pass1()

{

int index=0,i;

String s,prev="",substring;

try

{

BufferedReader inp = new BufferedReader(new FileReader("input.txt"));

File op = new File("pass1\_output.txt");

if (!op.exists())

op.createNewFile();

BufferedWriter output = new BufferedWriter(new FileWriter(op.getAbsoluteFile()));

while((s=inp.readLine())!=null)

{

if(s.equalsIgnoreCase("MACRO"))

{

prev=s;

for(;!(s=inp.readLine()).equalsIgnoreCase("MEND");mdtc++,prev=s)

{

if(prev.equalsIgnoreCase("MACRO"))

{

StringTokenizer st=new StringTokenizer(s);

String str[]=new String[st.countTokens()];

for(i=0;i<str.length;i++)

str[i]=st.nextToken();

mnt[mntc][0]=(mntc+1)+""; //mnt formation

mnt[mntc][1]=str[0];

mnt[mntc++][2]=(++mdtc)+"";

st=new StringTokenizer(str[1],","); //tokenizing the arguments

String string[]=new String[st.countTokens()];

for(i=0;i<string.length;i++)

{

string[i]=st.nextToken();

ala[alac][0]=alac+""; //ala table formation

index=string[i].indexOf("=");

if(index!=-1)

ala[alac++][1]=string[i].substring(0,index);

else

ala[alac++][1]=string[i];

}

}

else //automatically eliminates tagging of arguments in definition

{ //mdt formation

index=s.indexOf("&");

substring=s.substring(index);

for(i=0;i<alac;i++)

if(ala[i][1].equals(substring))

s=s.replaceAll(substring,"#"+ala[i][0]);

}

mdt[mdtc-1][0]=s;

}

mdt[mdtc-1][0]=s;

}

else

{

output.write(s);

output.newLine();

}

}

output.close();

}

catch(FileNotFoundException ex)

{

System.out.println("Unableind file ");

}

catch(IOException e)

{

e.printStackTrace();

}

}

static void pass2()

{

int alap=0,index,mdtp,flag=0,i,j;

String s,temp;

try

{

BufferedReader inp = new BufferedReader(new FileReader("pass1\_output.txt"));

File op = new File("pass2\_output.txt");

if (!op.exists())

op.createNewFile();

BufferedWriter output = new BufferedWriter(new FileWriter(op.getAbsoluteFile()));

for(;(s=inp.readLine())!=null;flag=0)

{

StringTokenizer st=new StringTokenizer(s);

String str[]=new String[st.countTokens()];

for(i=0;i<str.length;i++)

str[i]=st.nextToken();

for(j=0;j<mntc;j++)

{

if(str[0].equals(mnt[j][1]))

{

mdtp=Integer.parseInt(mnt[j][2]);

st=new StringTokenizer(str[1],",");

String arg[]=new String[st.countTokens()];

for(i=0;i<arg.length;i++)

{

arg[i]=st.nextToken();

ala[alap++][1]=arg[i];

}

for(i=mdtp;!(mdt[i][0].equalsIgnoreCase("MEND"));i++) //expand till MEND

{

index=mdt[i][0].indexOf("#");

temp=mdt[i][0].substring(0,index);

temp+=ala[Integer.parseInt(""+mdt[i][0].charAt(index+1))][1]; //converting char->string->integer & appending it

output.write(temp);

output.newLine();

}

flag=1;

}

}

if(flag==0) //when it is not a macro

{

output.write(s);

output.newLine();

}

}

output.close();

}

catch(FileNotFoundException ex)

{

System.out.println("Unableind file ");

}

catch(IOException e)

{

e.printStackTrace();

}

}

static void display(String a[][],int n,int m)

{

int i,j;

for(i=0;i<n;i++)

{

for(j=0;j<m;j++)

System.out.print(a[i][j]+" ");

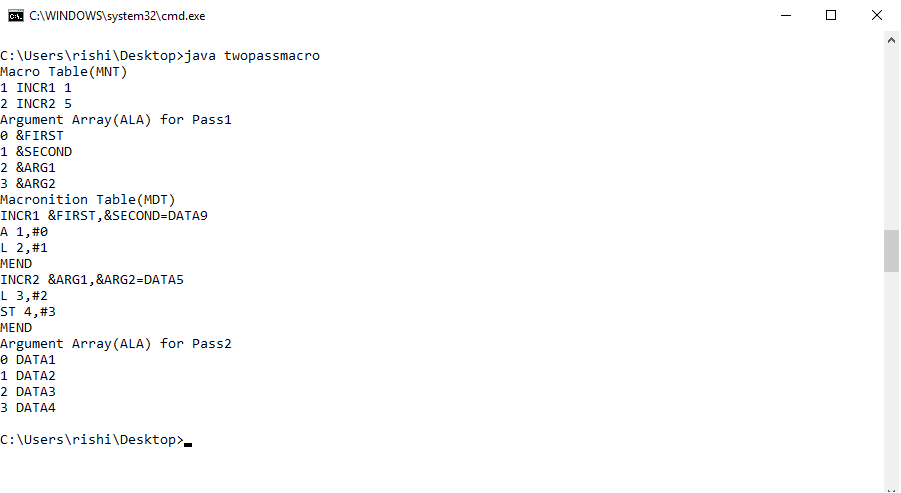
System.out.println();

}

}

}

***OUTPUT:***

******

***pass1\_output.txt***

PRG2 START

USING \*,BASE

INCR1 DATA1,DATA2

INCR2 DATA3,DATA4

FOUR DC F'4'

FIVE DC F'5'

BASE EQU 8

TEMP DS 1F

DROP 8

END

***pass2\_output.txt***

PRG2 START

USING \*,BASE

A 1,DATA1

L 2,DATA2

L 3,DATA3

ST 4,DATA4

FOUR DC F'4'

FIVE DC F'5'

BASE EQU 8

TEMP DS 1F

DROP 8

END

***Conclusion:*** Thus 2 pass macro processor studied and implemented successfully.