Name: Omkar Thange.
Roll No.: 19121014
Subject: System Programming and Operating System

Assignment No.: 02

Title:

Design suitable data structures and implement Pass-I of a two pass macro processor using OOP features in Java/C++. The output of Pass-I (MNT, MDT, ALA & Intermediate code file without any macro definitions) should be input for Pass-II.

Objectives:

1. To identify and design different data structure used in macro-processor implementation
2. To apply knowledge in implementation of two pass microprocessor.

Hardware Requirement:

PC/Laptop

Software Requirement:

1. Notepad **2.** JDK **3**. CMD

Theory:

1) What is macro processor?

- Macro represents a group of commonly used statements in the source programming language. Macro Processor replaces each macro instruction with the corresponding group of source language statements. This is known as the expansion of macros.
- Using Macro instructions programmer can leave the mechanical details to be handled by the macro processor. Macro Processor designs are not directly related to the computer architecture on which it runs.
- Macro Processor involves definition, invocation, and expansion.

2) Differentiate Macro and Function?

Macro	Function
1. Macros are Preprocessed **2. No Type Checking is done in Macro**	**1. Functions are Compiled** **2. Type Checking is Done in Function**

3. Using Macro increases the code length	**3. Using Function keeps the code length unaffected**
4. Use of macro can lead to side effect at later stages	**4. Functions do not lead to any side effect in any case.**
5. Speed of Execution using Macro is Faster	**5. Speed of Execution using Function is Slower**
6. Before Compilation, macro name is replaced by macro value	**6. During function call, transfer of control takes place**
7. Macros are useful when small code is repeated many times	**7. Functions are useful when large code is to be written.**
8. Macro does not check any Compile-Time Errors	**8. Function checks Compile-Time Errors**

3) Explain the design of Pass- I of macro-processor with the help of flowchart?

In Pass-I the macro definitions are searched and stored in the Macro Definition Table (MDT) and the entry is made in Macro Name Table (MNT).

Pass I:

1. The input- macro source program.
2. The output -macro source program to be used by Pass2.
3. Macro-Definition Table (MDT): store the body of macro definitions.
4. Macro-Definition Table Counter (MDTC): mark next available entry MDT.
5. Macro- Name Table (MNT): store names of macros.
6. Macro Name Table counter (MNTC): indicate the next available entry in MNT.
7. Argument List Array (ALA): substitute index markers for dummy arguments before storing a macro- defns.

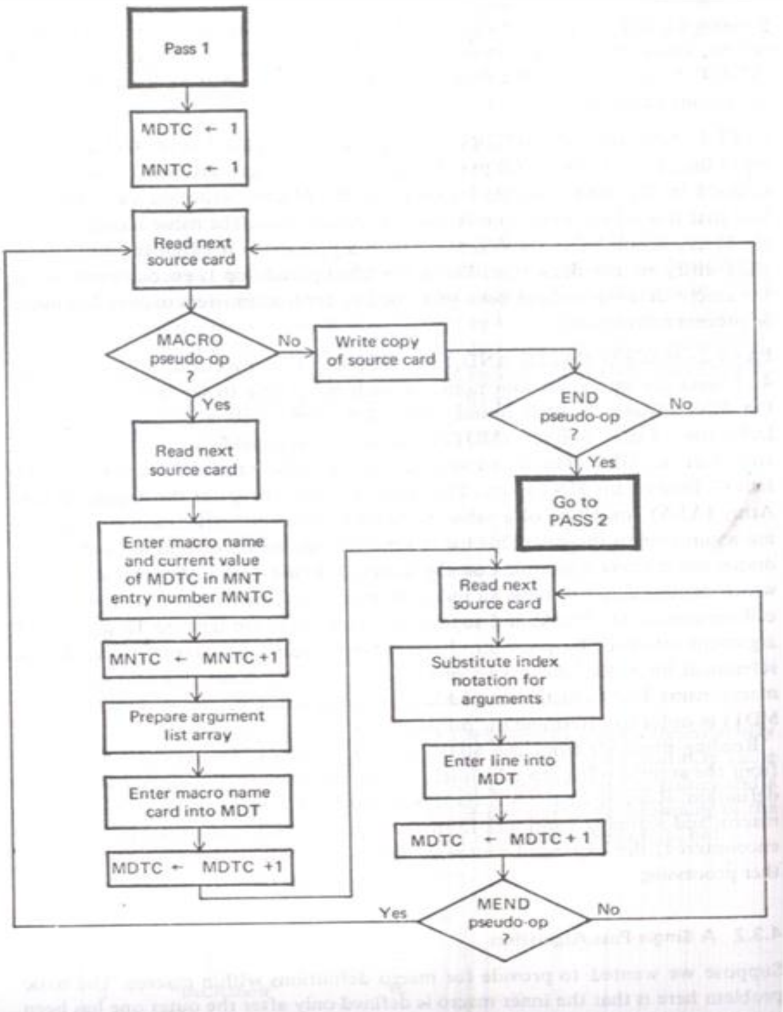
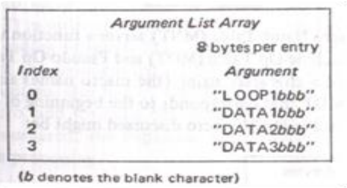


FIGURE 4.1 Pass 1–processing macro definitions

4) Explain the design of Data structure used in Pass-I?

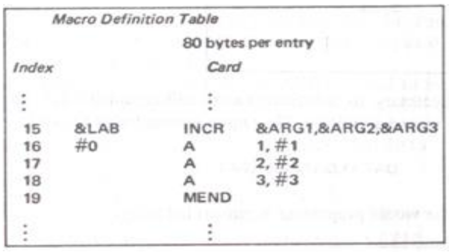


Argument List Array

8 bytes per entry

Index	*Argument*
0	"LOOP1*bbb*"
1	"DATA1*bbb*"
2	"DATA2*bbb*"
3	"DATA3*bbb*"

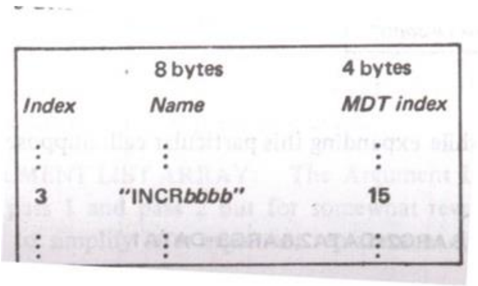
(*b* denotes the blank character)



Macro Definition Table

80 bytes per entry

Index	*Card*		
⋮		⋮	
15	&LAB	INCR	&ARG1,&ARG2,&ARG3
16	#0	A	1, #1
17		A	2, #2
18		A	3, #3
19		MEND	
⋮		⋮	



	8 bytes	4 bytes
Index	*Name*	*MDT index*
⋮	⋮	⋮
3	"INCR*bbbb*"	15
⋮	⋮	⋮

5) Explain the data structures used in Pass-I?

- **MDT**

  a. MDT is a table of text lines.
  b. Every line of each macro definition except the MACRO line, is stored in the MDT (MACRO line is useless during macro-expansion)
  c. Index keeps track of line numbers of the macro definition
  d. Card is 80 bytes of size entry stores the macro definition
  e. MEND is pseudo code &indicates the end of the definition.

- **MNT**

  Each MNT entry consists of:
  1. A character string (the macro name) &
  2. A pointer (index) to the entry in MDT that corresponds to the beginning of the macro- definition.(MDT index)

- **Argument List Array(ALA):**
  1. ALA is used during both Pass I & PassII but for some what reverse functions.
  2. During Pass I, in order to simplify later argument replacement during macro expansion, dummy arguments are replaced with positional indicators when defn is stored. Ex. # 1, # 2, # 3 etc.
  3. The ith dummy argument on the macro-name is represented in the body by #i.
  4. During Pass II, when there is macro expansion the ALA fills the arguments of the corresponding index with its appropriate argument in the call.

Input:

Macro_input.txt

```
MACRO
M1      &X, &Y, &A=AREG, &B=
MOVER       &A, &X
ADD     &A, ='1'
MOVER       &B, &Y
ADD     &B, ='5'
MEND
MACRO
M2      &P, &Q, &U=CREG, &V=DREG
MOVER       &U, &P
MOVER       &V, &Q
ADD     &U, ='15'
ADD     &V, ='10'
MEND
START       100
M1      10, 20, &B=CREG
M2      100, 200, &V=AREG, &U=BREG
END
```

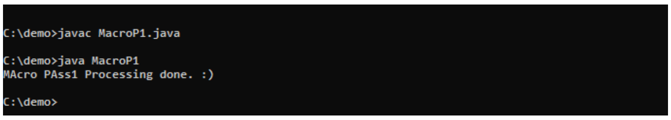
Program Code (Java):

MacroP1.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedHashMap;

public class MacroP1 {

    public static void main (String [] args) throws IOException {
        BufferedReader    br=new    BufferedReader    (new    FileReader
("C:\\demo\\macro_input.txt"));

        FileWriter mnt=new FileWriter ("mnt.txt");
        FileWriter mdt=new FileWriter ("mdt.txt");
        FileWriter kpdt=new FileWriter ("kpdt.txt");
        FileWriter pnt=new FileWriter ("pntab.txt");
        FileWriter ir=new FileWriter ("intermediate.txt");
        LinkedHashMap<String, Integer> pntab=new LinkedHashMap<> ();
        String line;
        String Macroname = null;
        int mdtp=1, kpdtp=0, paramNo=1, pp=0, KP=0, flag=0;
        while ((line=br.readLine ())! =null)
        {

            String parts[]=line.split("\\s+");
            if(parts[0].equalsIgnoreCase("MACRO"))
            {
                flag=1;
                line=br.readLine();
                parts=line.split("\\s+");
                Macroname=parts[0];
                if(parts.length<=1)
                {

    mnt.write(parts[0]+"\t"+pp+"\t"+kp+"\t"+mdtp+"\t"+(kp==0?kpdtp:(kpdtp+1))+"\n");
                    continue;
                }
                for(int i=1;i<parts.length;i++) //processing of parameters
                {
                    parts[i]=parts[i].replaceAll("[&,]", "");
                    //System.out.println(parts[i]);
                if(parts[i].contains("="))
                {
                    ++kp;
                    String keywordParam[]=parts[i].split("=");
                    pntab.put(keywordParam[0], paramNo++);
                    if(keywordParam.length==2)
                    {

kpdt.write(keywordParam[0]+"\t"+keywordParam[1]+"\n");
                    }
                    else
                    {
                        kpdt.write(keywordParam[0]+"\t-\n");
                    }
                }
                else
                {
                    pntab.put(parts[i], paramNo++);
                    pp++;
                }
            }

mnt.write(parts[0]+"\t"+pp+"\t"+kp+"\t"+mdtp+"\t"+(kp==0?kpdtp:(kpdtp+1))+"\n");
            kpdtp=kpdtp+kp;
            //System.out.println("KP="+kp);


        }
        else if(parts[0].equalsIgnoreCase("MEND"))
        {
            mdt.write(line+"\n");
            flag=kp=pp=0;
            mdtp++;
            paramNo=1;
            pnt.write(Macroname+":\t");
            Iterator<String> itr=pntab.keySet().iterator();
            while(itr.hasNext())
            {
                pnt.write(itr.next()+"\t");
            }
            pnt.write("\n");
            pntab.clear();
        }
        else if(flag==1)
        {
            for(int i=0;i<parts.length;i++)
```

```
                {
                    if(parts[i].contains("&"))
                    {
                        parts[i]=parts[i].replaceAll("[&,]", "");
                        mdt.write("(P,"+pntab.get(parts[i])+")\t");
                    }
                    else
                    {
                        mdt.write(parts[i]+"\t");
                    }
                }
                mdt.write("\n");
                mdtp++;
            }
            else
            {
                ir.write(line+"\n");
            }
        }
        br.close();
        mdt.close();
        mnt.close();
        ir.close();
        pnt.close();
        kpdt.close();
        System.out.println("MAcro PAss1 Processing done. :)");
    }

}
```

Output:



```
C:\demo>javac MacroP1.java

C:\demo>java MacroP1
MAcro PAss1 Processing done. :)

C:\demo>
```

Intermediate.txt

```
START       100
M1     10, 20, &B=CREG
M2     100, 200, &V=AREG, &U=BREG
END
```

Kdpt.txt

```
A      AREG
B      -
U      CREG
V      DREG
```

Mdt.txt

```
MOVER       (P,3)   (P,1)
ADD    (P,3)   ='1'
MOVER       (P,4)   (P,2)
ADD    (P,4)   ='5'
MEND
MOVER       (P,3)   (P,1)
MOVER       (P,4)   (P,2)
ADD    (P,3)   ='15'
ADD    (P,4)   ='10'
MEND
```

Mnt.txt

```
M1     2      2      1      1
M2     2      2      6      3
```

Pntab.txt

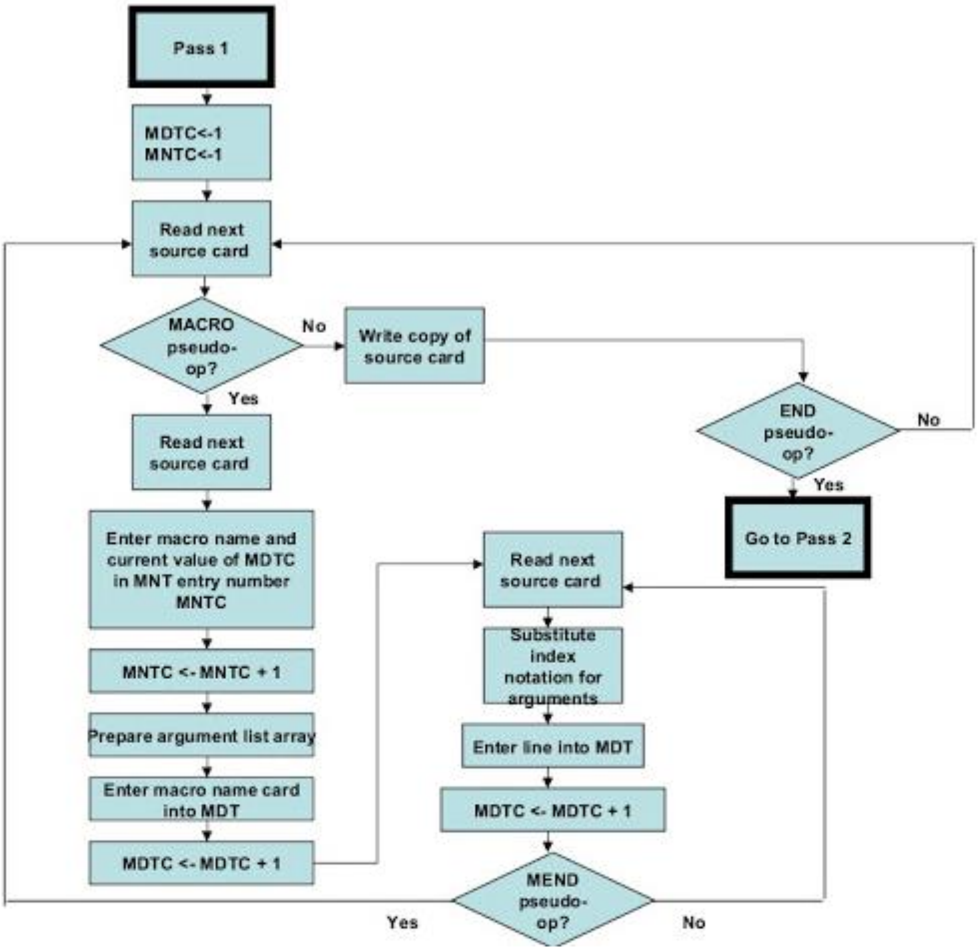
```
M1:    X      Y      A      B
M2:    P      Q      U      V
```

Algorithm/Flowchart(Pass 1 Macro processor):

Algorithm:-

1. Initializes MDTC and MNTC to 1
2. Reads next source card.
3. If MACRO pseudo code then
    a. Read from next source card
    b. Enter macro name and current value of MDTC in MNT entry number MNTC.
    c. Increment MNTC.
    d. Prepare ALA
    e. Enter macro name card into MDT.
    f. Increment MDTC.
    g. Read next source card.
    h. Substitute index notation for the arguments.
    i. Enter line into MDT
    j. Increment MDTC
    k. If MEND GOTO 2. else GOTO3.g.
4. Else write copy of source card.
5. If END then GOTO pass2 else GOTO 2.

Flowchart:


Pass 1
MDTC<-1
MNTC<-1
Read next source card
MACRO pseudo-op?
No
Write copy of source card
Yes
Read next source card
END pseudo-op?
No
Yes
Go to Pass 2
Enter macro name and current value of MDTC in MNT entry number MNTC
Read next source card
Substitute index notation for arguments
MNTC <- MNTC + 1
Prepare argument list array
Enter line into MDT
Enter macro name card into MDT
MDTC <- MDTC + 1
MDTC <- MDTC + 1
MEND pseudo-op?
Yes
No

Frequently Asked Questions:

1) **Define macro?**
   - Macro is a Single line abbreviation for set of instructions.
   - The macro processor replaces each macro invocation with the corresponding sequence of statements i.e. called as Macro.
   - **Syntax:**

```
MACRO          -------- Start of definition
INCR           -------- Macro name
A 1, DATA
A 2, DATA       ------Sequence of instructions to be
A 3, DATA                abbreviated
MEND           -------- End of definition
```

2) **Define purpose of pass-1 of two pass macro processor.**

   In Pass-I the macro definitions are searched and stored in the Macro Definition Table (MDT) and the entry is made in Macro Name Table (MNT).

3) **List out types of macro arguments.**

   Two ways of specifying arguments to a macro call
   - **Positional argument:**

     Argument are matched with dummy arguments according to order in which they appear.

     E.g. INCR A, B, C
     - “A” replaces first dummy argument.
     - “B” replaces second dummy argument
     - “C‟ replaces third dummy argument
   - **keyword arguments**

     This allows reference to dummy arguments by name as well as by position.

     E.g. INCR &arg1 = A, &arg3 = C, &arg2 =‟B‟

     e.g.□ INCR &arg1 = &arg2 = A, &arg2 =‟C‟

4) **What is the use of MDT-index field in MNT?**

   Each MNT entry consist of a pointer (index) to the entry in MDT that corresponds to the beginning of the macro- definition (MDT index).

5) **What we store in ALA?**

   In pass1 ALA is used in order to simplify later argument replacement during macro expansion. Dummy arguments are replaced with positional indicators when den is stored. Ex. # 1, # 2, # 3 etc.

Conclusion:

We have successfully completed implementation of pass-1 of macro processor.