Name :- Rushikesh Karbhari Palve
Roll No. :- 31258

Date : / /20 | Page No.

Assignment No. 5

DOP :- 20-09-2021 DOS :- 24-09-2021

Problem Definition :-

Design suitable data structures and implement pass-I of a two-pass macro-processor in Java.

Learning Objectives :-

(i) To study basic functioning of Macro processor.
(ii) To study working of nested macro.
(iii) To understand data structure of Pass-I macroprocessor.
(iv) To understand macro farfility.

Outcomes :-

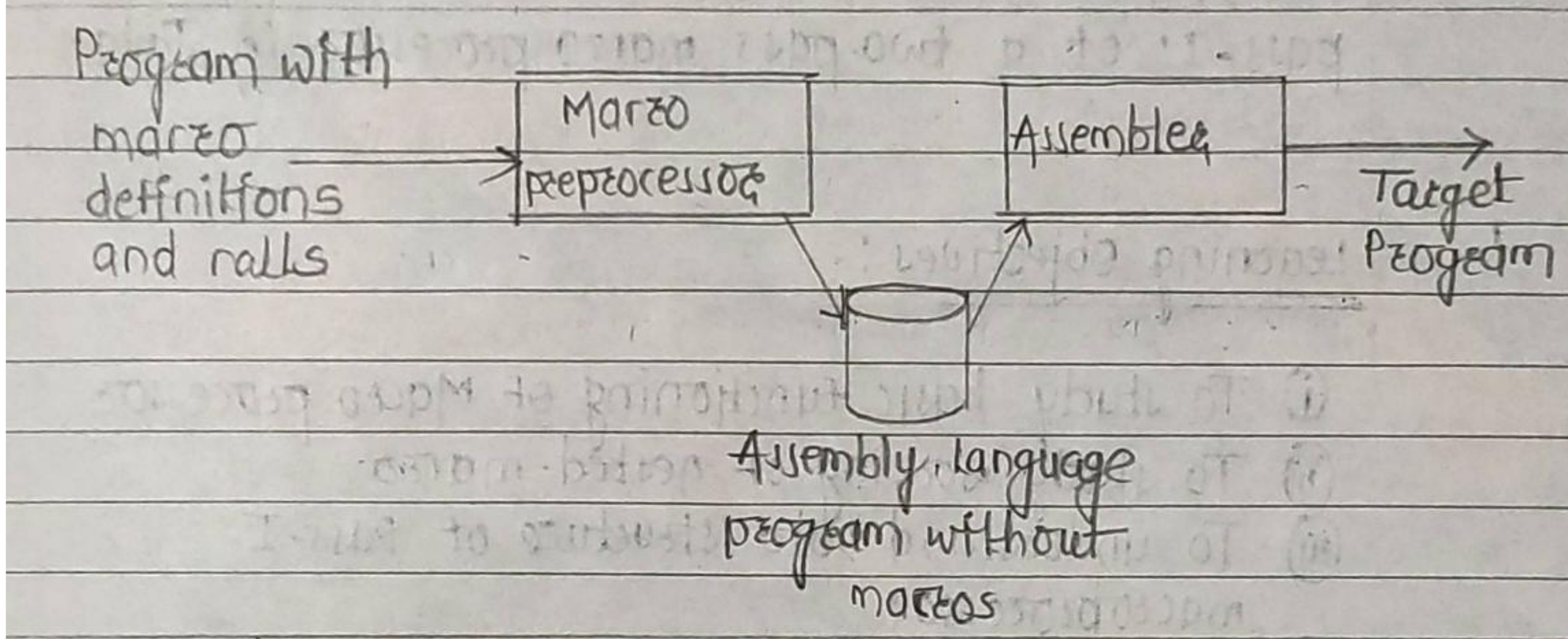
After completion of this assignment students will be able to -
(i) Implement pass-I macro processor.
(ii) Implement MNT, MDT table -
(iii) Understand concept pass-I macroprocessor.

Theory concepts :-

Macroprocessor :-

Macro pre-processor takes a source program containing macro definitions and macro calls and translates into an assembly language

program without any macro deffnitions or calls. This program ran now be handed over to a conventional assembler to obtain the target language. (as shown in Fig.)



Macro :-

Macro allows a sequence of source language code to be deffned once and then referred to by name each time it is referred. Each time this name occurs in a program, the sequence of codes is substituted at that point.

Macro has following parts :-

1. Name of macro
2. Parameters in macro
3. Macro Deffnition

Parameters are optional.

How To Define a Macro :-

Macro can be formatted in following order :-

Start of definition	MACRO
macro name	mymacro
macro body	ADD AREG, X
	ADD BREG, X
End of macro definition	MEND

'MACRO' pseudo-op is the first line of definition and identifies the following line as macro instruction name.

Following the name line is sequence of instructions being abbreviated the instructions comprising the 'MACRO' instruction.

the definition is terminated by a line with MEND pseudo-op.

Example of Macro -

① Macro without parameters

```
MACRO
    mymacro
        ADD AREG, X
        ADD BREG, X
MEND
```

② Macro With parameters

```
MACRO
    addmacro &A
        ADD AREG, &A
        ADD BREG, &A
MEND
```

The macro parameters (Formal parameters) are initialized with '&', used as it is in operation.

Formal parameters are those which are in deffinition of macro.
Whereas while calling macros we use Actual parameters.

How to call a macro?

A macro is called by writing macro name with actual parameters in an Assembly Program.
Macro calls leads to Macro Expansion.

Syntax:-

<macro-name> [<list of parameters>]

Example:- for above deffinitions of macro...

① mymacro
② addmacro X

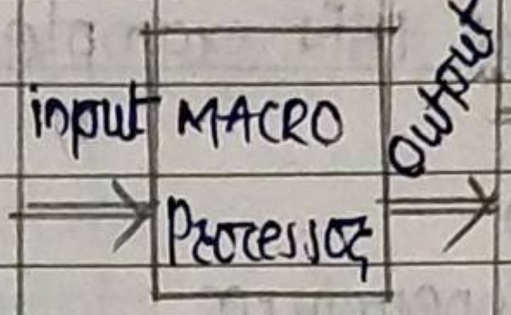
Macro Expansion:-

Each call to macro is replaced by its body.
During Replacement, actual pardmeter is used in place of formal parameter.

① During macro expansion, each statement forming the body of macro is picked up one bye one sequentially.
② Each statement inside the macro may have:
   1) An ordinary string, which is copied as it is during expansion.
   2) The name of a formal parameter which is proceeded by character '&'.
③ During macro expansion an ordinary string is retained without any modification. Formal Parameters (strings starting with &) is replaced by

the actual parameter value.

Source Program	Target Code
MACRO	ADD AREG, X
mymacro	ADD AREG, X
ADD AREG, X	
ADD BREG, X	ADD AREG, X
MEND	ADD AREG, X
mymacro	
mymacro	ADD AREG, X
mymacro	ADD AREG, X



Macro Expansion

Macro with keyword parameters :-

These are the methods to call macros with formal parameters. These formal parameters are of two types.

1) Positional Parameters :-

Initiated with '&'.

Ex:- mymacro &X.

2) Keyword Parameters :-

Initiated with '&' but has some default value. During a call to macro, a keyword parameter is specified by its name.

Ex:- mymacro &X=A.

Nested Macro Calls :-

Nested macro calls are just like nested function calls in our normal calls. Only the transfer of control from one macro to other is done

Consider this example :-

```
MACRO
    Innermacro
        ADD AREG, X
MEND
MACRO
    outermacro
        Innermacro
        ADD AREG, Y
MEND
outermacro
```

In this example, firstly the MACRO outermacro gets executed & then Innermacro. So Output will be Adding X & Y values in AREG register.

Algorithm :-

Scan all MACRO definitions one by one.

a) Enter its name in macro name table (MNT).
b) Store the entire macro definition in the macro definition table (MDT).
c) Add the information in the MNT indicates where definition of macro can be found in MDT.
d) Prepare argument list array (ALA).

Data structures of TWO pass macros :-

1) Macro Name Table pointer (MNTP)
2) Macro Deffnition Table pointer (MDTP)
3) Macro Name Table :
   - macro number (i.e. pointer referenced from MNTP)
   - Name of macros
   - MDTP (i.e. points to start position to MDT)
4) Macro Definition Table :
   - Location Counter (where MDTP points to start position of macro)
   - Opcode
   - Rest (i.e. it will contain the other part than opcodes used in macro).
5) Argument List Array :
   - Index given to parameter
   - Name of parameter.

Example for pass I of macro processor :-

Assembly Code Segment :-

```
MACRO
INCR &X, &Y, &REG = AREG
MOVER &REG, &X
ADD &REG, &Y
MOVEM &REG, &X
MEND
MACRO
DECR &A, &B, &REG = BREG
MOVER &REG, &A
SUB &REG, &B
```

```
MOVEM &REG, &A
MEND
START
READ N1
READ N2
INCR N1, N2, REG=CREG
DECR N1, N2
STOP
N1 DS 1
N2 DS 2
END
```

Output of pass-I of Macro processor :-

#MNT

<#name>	<#pp>	<#kp>	<#MDTP>	<#KPDTP>
INCR	2	1	1	101
DECR	2	1	65	102

#PNTAB(INCR)

1.	X
2.	Y
3.	REG

#PNTAB(DECR)

1.	A
2.	B
3.	REG

#KPDTAB

	<name>	<value>
101.	REG	AREG
102.	REG	BREG

#MDT

1.	MOVER (P,3) (P,1)
2.	ADD (P,3) (P,2)
3.	MOVEM (P,3) (P,1)
4.	MEND
5.	MOVER (P,3) (P,1)
6.	SUB (P,3) (P,2)
7.	MOVEM (P,3) (P,1)
8.	MEND

CONCLUSION :-

Thus, I have implemented Pass-I macroprocessor by producing MNT and MDT table. Also, I understood working of nested macro and macro farfility.

CODE :-

```
package assignmentNo_5;

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 AssignmentNo_5
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new FileReader("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?kp
dtp:(kpdtp+1))+"\n");
                    continue;
                }
                for(int i=1; i<parts.length; i++) //processing of parameters
                {
                    parts[i]=parts[i].replaceAll("[&,]", "");
                    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;
            }
            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("\n\t Executed Successfully ...!!");
    }
}
```

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
```

OUTPUT :-

mnt.txt

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

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
```

kpdt.txt

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

pntab.txt

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

intermediate.txt

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