SSC LAB ASSIGNMENT NO.3
MACRO PASS 1

NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1

Assignment Title: Design of Pass 1 of Two Pass Macroprocessor.

Aim: Design suitable data structure & implement pass 1 of Two Pass Macroprocessor.

Objective: Design suitable data structure & implement pass 1 of Two Pass Macroprocessor. Input should consist of a one macro definition and one macro call and few assembly language instructions.

Theory:

1. Description about the macro processor.

Macro instruction is the notational convenience for the programmer. For every occurrence of macro the whole macro body or macro block of statements gets expanded in the main source code. Thus Macro instructions make writing code more convenient.

2. Data structures required for 2 pass macro processors.

Pass I

1. Input macro source deck(ALP)

2.Output macro source deck copy for use by pass 2

3.Macro definition table (MDT) ,used to store body of macro def.

4.Macro name table (MNT) ,used to store names of defined macros

5.MDTC-MDT counter ,used to indicate next available entry in MDT

6.MNTC-MNT counter, used to indicate next available entry in MNT

7.Argument List Array(ALA) used to substitute index markers for dummy

arguments before storing macro definition.

MDT (Macro Definition Table) :

- MDT is a table of text lines.
- Every line of each macro definition except the MACRO line, is stored in the MDT (MACRO line is useless during macro-expansion)
- MEND is kept to indicate the end of the depns.
- The macro-name line is retained to facilitate keyword argument replacement.

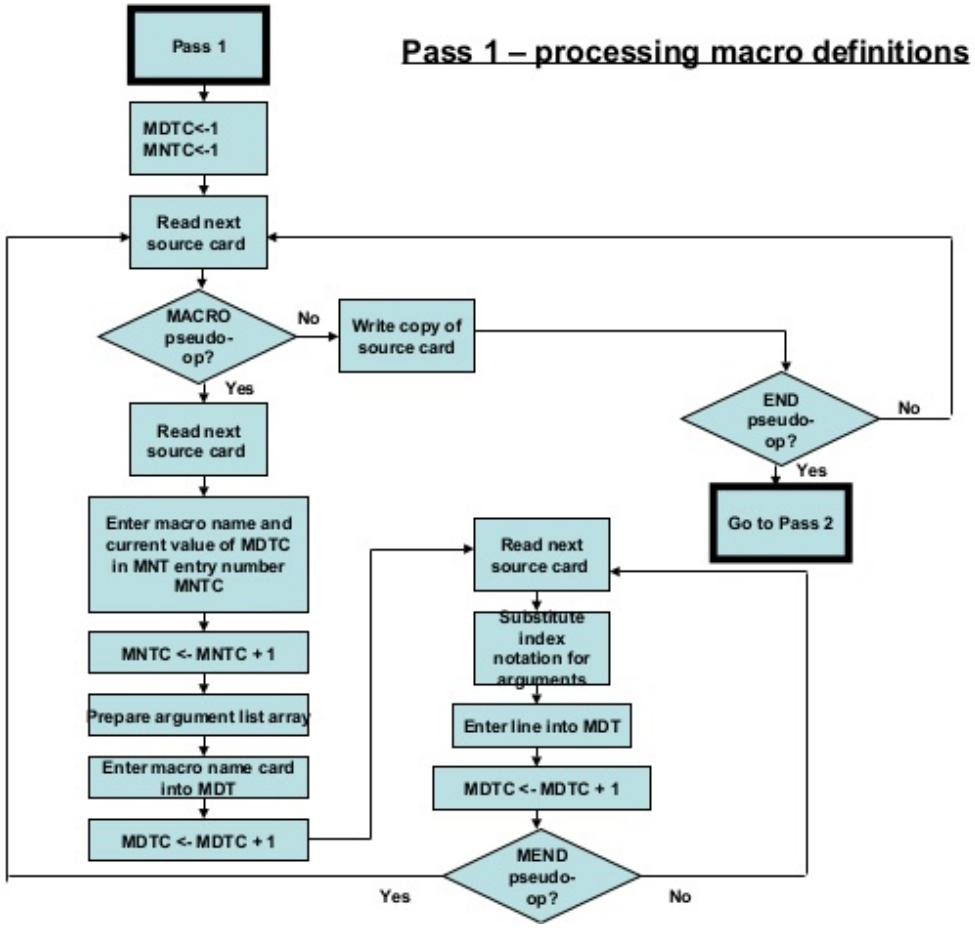
MNT (Macro Name Table) :

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

ALA (Argument List Array) :

- ALA is used during both Pass1 & Pas2 but for some what reverse functions.
- During Pass1, 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. The ith dummy argument on the macro-name is represented in the body by #i.
- These symbols are used in conjunction with ALA prepared before expansion of a macro-call.
- Symbolic dummy argument are retained on macro- name to enable the macro processor to handle argument replacement byname rather by position.

3. Flowchart for Pass I.



Algorithm for Pass 1:

Step 1: /* Initialization of counters for MDT and MNT * /

Step 2: Read Next Instruction (and divide it into it's various field as label,

mnemonic (opcode arguments).

Step 3: / * Check for macro definition start * /

if opcode = MACRO goto Step 5

else /* this is not macro definition * /

go to step 4.

Step 4: (a) Write copy of instruction to output of Pass-I

(b) Check whether opcode = END or not

(c) if OPCODE "* END goto Step 2

(d) if OPCODE = END goto Pass-2 i.e. End of this algorithm for Pass- I.

Step 5: /* Start of macro definition is identified. Now Pass-I will process

contents of macro definition after pseudo op MACRO to MEND * /

(a) Read Next Instruction.

(*definitely this is macro name instruction therefore as a processing of this instruction an entry will be made in MNT, ALA will be prepared for this macro, this macro name instruction will be entered in MDT * /

(b) Enter <macro-name, MDTC> into MNT at MNTC

/* current available rows in MDT and MNT are MDTC and MNTC, so macro name and it's starting MDT index i.e. current value of

MDTC is entered in MNT at available row i.e. MNTC * /

(c) MNTC ~ MNTC + 1 /* To point next available row in MNT * /

(d) Prepare Argument List Array

/* ALA is partially constructed by Pass-I to assign universal

integer index to dummy arguments * /

(e) Enter macroname instruction in MDT at MDTC.

(1) MDTC ~ MDTC + 1.

/* In step 5, macro name instruction (instruction just after MACRO pseudo op in macro definition and on this instruction name of the macro and corresponding dummy arguments are specified) is processed * /

Step 6: /* Process other instructions in macro definition inducing MEND

Instruction * /

(a) Read next card

(b) Substitute Index notations for dummy-arguments.

(c) Enter this instruction (where dummy arguments are replaced by integer indices) into MDT.

(d) MDTC ~ MDTC + 1

(e) if OPCODE of this instruction is MEND then goto Step 2.

else goto Step 6 a.

The data structures associated with Macro Processor:

Input: Assembly Language Program.

BATCH 1 INPUT :

```
MACRO
INCR &ARG1,&ARG2
ADD AREG,&ARG1
ADD BREG,&ARG2
MEND
MACRO
DECR &ARG3,&ARG4
SUB  AREG,&ARG3
SUB  BREG,&ARG4
```

```
MEND
START
MOVER  AREG,S1
MOVER  BREG,S1
INCR  D1  D2
DECR D3 D4
S1  DC  5
D1  DC  2
D2  DC  3
D3 DC 4
D4 DC 5
END
```

Output:

1. Program without Macro Definition (Pass-I)

```
START
MOVER  AREG,S1
MOVER  BREG,S1
INCR  D1  D2
DECR D3 D4
S1  DC  5
D1  DC  2
D2  DC  3
D3 DC 4
D4 DC 5
END
```

2. Macro Definition Table (MDT)

MDT TABLE :	
MDT INDEX	INSTRUCTION
1	INCR &ARG1,&ARG2
2	ADD AREG,#1
3	ADD BREG,#2
4	MEND
5	DECR &ARG3,&ARG4
6	SUB AREG,#3
7	SUB AREG,#4
8	MEND

3. Macro Name Table (MNT)

MNT TABLE :		
ID	NAME	MDT INDEX
1	INCR	0
2	DECR	4

4. Argument List Array (ALA)

ALA TABLE :	
ALA INDEX	DUMMY ARGS
#1	&ARG1
#2	&ARG2
#3	&ARG3
#4	&ARG4

Conclusion: The function of Pass 1 in a Macro Processor studied.

Platform: Linux (Java)

Conclusion: The function of Pass 1 in assembler is studied along with errors coming in each pass.

PROGRAM :

```
/*
 SSC LAB ASSIGNMENT NO.3
DESIGN OF MACRO PASS 1
NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1
 */

package ssc_lab;

import java.util.*;
import java.io.*;
import java.io.BufferedReader;
```

```
public class lab3_macro {
   static List<String> MDT;
   static Map<String, String> MNT;
   static Map<String, String> ALA;
   static List<String> outFile;
   static int mntPtr, mdtPtr, filePtr;
   static int argIndex = 1;

   public static void main(String[] args) {
         try {
               pass1();
         }
         catch(Exception ex) {
               ex.printStackTrace();
         }
   }

   static void pass1() throws Exception{
         //Initialize data structures
         MDT = new ArrayList<String>();
         MNT = new LinkedHashMap<String, String>();
         ALA = new LinkedHashMap<String, String>();
         outFile = new ArrayList<String>();
         mntPtr = 0;
         mdtPtr = 0;

         BufferedReader input = new BufferedReader(new
InputStreamReader(new FileInputStream("lab3_input.txt")));
         String s;
         boolean procMacDef = false;
         boolean procMacName = false;

         System.out.println("SSC LAB 3 :");
         System.out.println("MACRO PASS 1\n");

         while((s=input.readLine())!=null) {
               String s_arr[] = tokenizeString(s, " ");

               String curToken = s_arr[0];

               if(curToken.equalsIgnoreCase("MACRO")) {
                     procMacDef = true;
                     procMacName = true;
                     continue;
               }
               else {
                     if(procMacDef == false && procMacName == false) {
                           outFile.add(filePtr,s);
                           filePtr++;
                     }
               }
               if(procMacName == true) {

                     MNT.put(curToken, mdtPtr+"");
                     mntPtr++;
                     procMacName = false;
                     processArgumentList(s_arr[1]);
```

```
                    MDT.add(mdtPtr,s);
                    mdtPtr++;
                    continue;
               }
               if(procMacDef == true) {
                    String strIndexArg;
                    strIndexArg = processArguments(s);
                    MDT.add(mdtPtr, strIndexArg);
                    mdtPtr++;
               }
               if(curToken.equalsIgnoreCase("MEND")) {
                    procMacDef = false;
                    continue;
               }
          }
               input.close();
               System.out.println("--------------MNT--------------");
               Iterator<String> itMNT = MNT.keySet().iterator();
               String key, mntRow, mdtRow;
               while(itMNT.hasNext()) {
                    key = (String)itMNT.next();
                    mntRow = key + " " + MNT.get(key);
                    System.out.println(mntRow);
               }

               System.out.println("\n---------------ALA--------------");
               Iterator<String> itALA = ALA.keySet().iterator();
               String key1, alaRow;
               while(itALA.hasNext()) {
                    key1 = (String)itALA.next();
                    alaRow = key1 + " " + ALA.get(key1);
                    System.out.println(alaRow);
               }

               System.out.println("\n---------------MDT--------------");
               for(int i = 0; i< MDT.size();i++) {
                    mdtRow = i + " " + MDT.get(i);
                    System.out.println(mdtRow);
               }
               System.out.println("\n-----------OUTPUT FILE------------");
               String outFileRow;
               for(int i = 0;i<outFile.size();i++) {
                    outFileRow = i + " " + outFile.get(i);
                    System.out.println(outFileRow);
               }

}

static String[] tokenizeString(String str, String seperator) {
     StringTokenizer st = new StringTokenizer(str, seperator, false);

     String s_arr[] = new String[st.countTokens()];
     for(int i=0; i<s_arr.length;i++) {
          s_arr[i] = st.nextToken();
     }
     return s_arr;
}
```

```
    static void processArgumentList(String argList) {
          StringTokenizer st = new StringTokenizer(argList,",",false);
          int argCount = st.countTokens();
          String curArg;
          for(int i=1;i<=argCount;i++) {
                curArg = st.nextToken();
                ALA.put(curArg, "#"+argIndex);
                argIndex++;
          }
    }
    static String processArguments(String argList) {
          StringTokenizer st = new StringTokenizer(argList,",",false);
          int argCount = st.countTokens();
          String curArg, argIndexed;
          for(int i = 1; i<=argCount;i++) {
                curArg = st.nextToken();
                argIndexed = ALA.get(curArg);
                if(argIndexed == null) {
                      continue;
                }
                argList = argList.replaceAll(curArg, argIndexed);
          }
          return argList;
    }
}
```

OUTPUT SS :

```
<terminated> lab3_macro [Java Application] C:\Program Files\AdoptOpenJDK\jdk-14.0.2.12-hotspot\bin\ja
SSC LAB 3 :
MACRO PASS 1

--------------MNT--------------
INCR 0
DECR 4

--------------ALA--------------
&ARG1 #1
&ARG2 #2
&ARG3 #3
&ARG4 #4

--------------MDT--------------
0 INCR &ARG1,&ARG2
1 ADD AREG,#1
2 ADD BREG,#2
3 MEND
4 DECR &ARG3,&ARG4
5 SUB  AREG,#3
6 SUB  BREG,#4
7 MEND

-----------OUTPUT FILE------------
0 START
1 MOVER  AREG,S1
2 MOVER  BREG,S1
3 INCR  D1  D2
4 DECR D3 D4
5 S1  DC  5
6 D1  DC  2
7 D2  DC  3
8 D3 DC 4
9 D4 DC 5
10 END
```