SSC LAB ASSIGNMENT NO.4
MACRO PASS 2

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

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

Aim: Design of pass II of Two Pass Macroprocessor.

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

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 II

1. The copy of the input macro source deck
2. Expanded source output
3. MDT created by pass 1
4. MNT created by pass 1
5. MDTP used to indicate the next line of text to be used during macro expansion.
6. The ALA used to substitute macro call arguments for the index markers in the stored 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) :

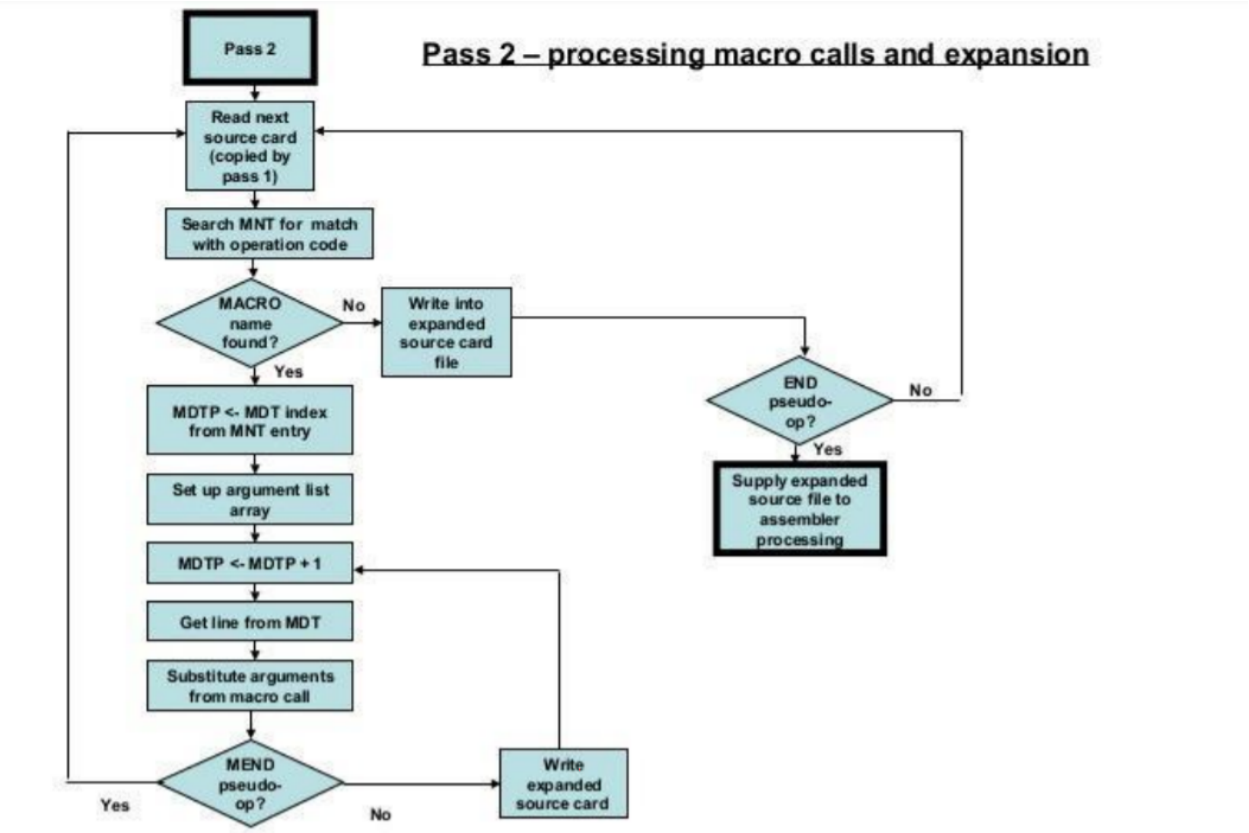
- ALA is used during both Pass1 & Pas2 but for somewhat reverse functions.
- During Pass1, in order to simplify later argument replacement during macro expansion, dummy arguments are replaced with positional indicators when defn is stored. Ex. # 1, # 2, # 3 etc. The ith dummy argument on the macro-name is represented in the body by #i.
- These symbols are used in conjunction with ALA prepared before expansion of a macro-call.
- Symbolic dummy arguments are retained on macro- name to enable the macro processor to handle argument replacement byname rather by position

PASS 2 MACRO CALLS AND EXPANSION

When a macro call is found, the call processor sets a pointer, the MDTP, to the corresponding macro definitions stored in the MDT. The initial value of the MDTP is obtained from the MDT index field of the MNT entry. The macro expander prepares the ALA consisting of a table of dummy arguments indicates and corresponding arguments to the call.

Reading proceeds from the MDT, as each successive line is read, the values from the argument list are substituted for dummy arguments indicates in the macro definition. Reading of the MEND line in the MDT terminates expansion of the macro and scanning continues from the input file. When the END op is encountered the expanded source is transferred to the assembler for further processing.

3. **Flowchart for Pass II.**



Algorithm for Pass II:

Step 1: Read Instruction from source program output by pass I divide its field as label mnemonic opcode arguments.

Step 2: Search through MNT to find match for opcode of instruction step 1 with micro name MNT.

Step 3: / * if no micro call found * / if no match then it indicates that this instruction is not a micro call instruction and then hence :

- write this instruction to expanded source program.
- Check whether opcode of this instruction is END as NOT.
- If NOT END then it indicates that this is not end of source pogram & hence go is step 1.
- If Opcode = then this indicates end of source program and hence give the output pass II e. expanded source program to assembler.

Step 4: /* if macro name found */ if OPCODE of (instruction read in step 1) = any macro name of MNT then it indicate & hence : this instruction is macro call instruction & hence :

- Obtain corresponding MDT indent and assign to MDTP.
- Setup ALA (for association of integr indicates and actual parameters.
- MDTP MDTP + 1
- Get next instruction from MDT.
- Substitute actual arguments instead of integer indicate.
- If OPCODE of this instruction is not MEND then write this instruction to expand source program.
- If OPCODE of this instruction is MEND then go to step 1 again.

Input: Batch 1

MNT

MNT Index	Macro Name	MDT Index
1	INCR	1
2	DECR	5

MDT

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 BREG #4
8	MEND

ALA

ALA Index	Dummy Arguments	Actual Arguments
#1	&ARG1	D1
#2	&ARG2	D2
#3	&ARG3	D3
#4	&ARG4	D4

ALP without Macro Definition but with Macro Call

```
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:

Expanded ALP without Macro Definition and without Macro Call

```
START

   MOVER AREG S1

   MOVER BREG S1

    ADD AREG D1

    ADD BREG D2

    SUB AREG D3

    SUB BREG D4

   S1 DC 5

   D1 DC 2

   D2 DC 3

   D3 DC 4

   D4 DC 5

END
```

Conclusion: The function of pass II of a 2 pass macroprocessor is studied along with advanced macro facility.

PROGRAM :

```
/*
 SSC LAB ASSIGNMENT NO.4
DESIGN OF MACRO PASS 2
NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1
 */

package ssc_lab;
import java.util.*;
import java.io.*;

public class M_pass2 {
   static List<String> MDT;
   static List<String> outFile;
   static Map<String, String> MNT;
   static int mntPtr, mdtPtr;
   static Map<String,String> APT;
   public static void main(String[] args) {
         // TODO Auto-generated method stub
         try {
               pass2();
         }catch(Exception ex) {
               ex.printStackTrace();
         }
   }
   static void pass2() throws Exception{
         //Initialize data structure
         MDT = new ArrayList<String>();
         MNT = new LinkedHashMap<String, String>();
         APT = new HashMap<String,String>();
         outFile = new ArrayList<String>();
         mntPtr = 0; mdtPtr = 0;
         MNT.put("INCR", mdtPtr+"");
         MDT.add(mdtPtr,"INCR    &ARG1,&ARG2");
         mdtPtr++;
         MDT.add(mdtPtr,"ADD    AREG,#0");
         mdtPtr++;
         MDT.add(mdtPtr,"ADD    BREG,#1");
         mdtPtr++;
         MDT.add(mdtPtr,"MEND");
         mdtPtr++;

         MNT.put("DECR", mdtPtr+"");
         MDT.add(mdtPtr,"DECR    &AGR3,&ARG4");
         mdtPtr++;
         MDT.add(mdtPtr,"SUB    AREG,#2");
         mdtPtr++;
```

```
        MDT.add(mdtPtr,"SUB    BREG,#3");
        mdtPtr++;
        MDT.add(mdtPtr,"MEND");
        mdtPtr++;
        mntPtr++;

        System.out.println("SSC LAB 4 :");
        System.out.println("MACRO PASS 2\n");

        System.out.println("======== MNT ========");
        Iterator<String> itMNT = MNT.keySet().iterator();
        String key, mntRow, mdtRow,aptRow;
        while(itMNT.hasNext()) {
              key = (String)itMNT.next();
              mntRow = key + " " + MNT.get(key);
              System.out.println(mntRow);
        }


        System.out.println("========== MDT ==========");
        for(int i =0; i < MDT.size(); i++) {
              mdtRow = i + " " + MDT.get(i);
              System.out.println(mdtRow);
        }
        BufferedReader input = new BufferedReader(new
InputStreamReader(new FileInputStream("input4.txt")));
        String s;
        boolean processingMacroCall = false;
        System.out.println("========== Pass 2 Output ==========");

        while((s = input.readLine())!=null) {
              Iterator<String> itMNT1= MNT.keySet().iterator();
              String mkey, mntRow1;
              String s_arr[]=tokenizeString(s," ");
              int flag=0;
              int f1=0;
              String curToken = s_arr[0];
              while(itMNT1.hasNext()) {
                    key = (String)itMNT1.next();
                    mntRow1 = key+ " " + MNT.get(key);
                    String m_arr[] = tokenizeString(mntRow1,"    ");
                    String curTokenMacro = m_arr[0];
                    if(curTokenMacro.equalsIgnoreCase(curToken))
                    {
                          flag=1;
                          APT = new HashMap<String, String>();
                          String ap[] = tokenizeString(s_arr[1],",");
                          for(int i = 0; i < ap.length; i++) {
                                    f1=1;
                                    APT.put(ap[i], "#"+i);
                               }
                          if(f1==1) {
                               APT.put("D3","#2");
                               APT.put("D4","#3");
                          }

                          int cmdtp=Integer.parseInt(m_arr[1]);
```

```
                        int cmdtpn = cmdtp+1;
                        mdtRow = MDT.get(cmdtpn);
                        while(!(mdtRow.equalsIgnoreCase("MEND")))
                        {
                              String mdt_arr[] = tokenizeString(mdtRow,"
");
                              //System.out.print("this is mdt_arr[0]
"+mdt_arr[0]+" and this is mdt_arr[1] "+mdt_arr[1]+"\n");
                              System.out.print("+"+" "+mdt_arr[0]+" ");
                              String mdt_par[]=tokenizeString(mdt_arr[1],
", ");
                              //System.out.print("this is mdt_par[0]
"+mdt_par[0]+ " and mdt_par[1] "+mdt_par[1]+"\n");
                              for(int i=0;i<mdt_par.length;i++)
                              {
                                    if(mdt_par[i].startsWith("#")) {
                                          Iterator<String> itALA =
APT.keySet().iterator();
                                          String key1, alaRow;
                                          while(itALA.hasNext()) {
                                                key1=(String)itALA.next();
                                                String ind=APT.get(key1);

if(mdt_par[i].equalsIgnoreCase(ind))
                                                {

System.out.println(key1);


                                                }
                                          }
                                    }

                                    else {
                                          System.out.print(mdt_par[i]+"
");
                                    }
                              }


                              cmdtpn=cmdtpn+1;
                              mdtRow=MDT.get(cmdtpn);
                        }
                  }
            }

                  if(flag==0) {
                        //System.out.println(" ");
                        System.out.println(s);
                        //System.out.println("FLAG = 0");
                        if(s=="END")
                              break;
                  }
            }



      }
```

```
static String[] tokenizeString(String str, String separator) {
   StringTokenizer st = new StringTokenizer(str,separator,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;
}
}
```

OUTPUT SS :