Pass – 1 Assembler Program

```
import java.io.*;
class SymbTab
{
public static void main(String args[])throws Exception
{
FileReader FP=new FileReader("/Desktop/Java/input.txt");
BufferedReader bufferedReader = new BufferedReader(FP);
String line=null;
int line_count=0,LC=0,symTabLine=0,opTabLine=0,litTabLine=0,poolTabLine=0;
//Data Structures
final int MAX=100;
String SymbolTab[][]=new String[MAX][3];
String OpTab[][]=new String[MAX][3];
String LitTab[][]=new String[MAX][2];
int PoolTab[]=new int[MAX];
// int litTabAddress=0;
/*---------------------------------------------------------------------------------------------------*/
System.out.println("___________________________________________________________");
while((line = bufferedReader.readLine()) != null)
{
String[] tokens = line.split("\t");
if(line_count==0)
{
LC=Integer.parseInt(tokens[1]);
//set LC to operand of START
for(int i=0;i<tokens.length;i++) //for printing the input program
System.out.print(tokens[i]+"\t");
System.out.println("");
}
else
{
for(int i=0;i<tokens.length;i++) //for printing the input program
System.out.print(tokens[i]+"\t");
System.out.println("");
if(!tokens[0].equals(""))
{
//Inserting into Symbol Table
SymbolTab[symTabLine][0]=tokens[0];
SymbolTab[symTabLine][1]=Integer.toString(LC);
SymbolTab[symTabLine][2]=Integer.toString(1);
symTabLine++;
}
else if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
{
//Entry into symbol table for declarative statements
SymbolTab[symTabLine][0]=tokens[0];
SymbolTab[symTabLine][1]=Integer.toString(LC);
SymbolTab[symTabLine][2]=Integer.toString(1);
symTabLine++;
}

if(tokens.length==3 && tokens[2].charAt(0)=='=')
{
//Entry of literals into literal table
LitTab[litTabLine][0]=tokens[2];
LitTab[litTabLine][1]=Integer.toString(LC);
litTabLine++;
}
else if(tokens[1]!=null)
{
//Entry of Mnemonic in opcode table
OpTab[opTabLine][0]=tokens[1];
if(tokens[1].equalsIgnoreCase("START")||tokens[1].equalsIgnoreCase("END")||tokens[1].equalsIgnoreCase("
ORIGIN")||tokens[1].equalsIgnoreCase("EQU")||tokens[1].equalsIgnoreCase("LTORG")) //if Assembler
Directive
{
OpTab[opTabLine][1]="AD";
OpTab[opTabLine][2]="R11";
}
else if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
{
OpTab[opTabLine][1]="DL";
OpTab[opTabLine][2]="R7";
}
else
{
OpTab[opTabLine][1]="IS";
OpTab[opTabLine][2]="(04,1)";
}
opTabLine++;
}
}
line_count++;
LC++;
}

System.out.println("______________________________________________________");
```

```
//print symbol table
System.out.println("\n\n SYMBOL TABLE ");
System.out.println("--------------------------");
System.out.println("SYMBOL\tADDRESS\tLENGTH");
System.out.println("--------------------------");
for(int i=0;i<symTabLine;i++)
System.out.println(SymbolTab[i][0]+"\t"+SymbolTab[i][1]+"\t"+SymbolTab[i][2]);
System.out.println("--------------------------");


//print opcode table
System.out.println("\n\n OPCODE TABLE ");
System.out.println("----------------------------");
System.out.println("MNEMONIC\tCLASS\tINFO");
System.out.println("----------------------------");
for(int i=0;i<opTabLine;i++)
System.out.println(OpTab[i][0]+"\t\t"+OpTab[i][1]+"\t"+OpTab[i][2]);
System.out.println("----------------------------");

//print literal table
System.out.println("\n\n LITERAL TABLE ");
System.out.println("-----------------");
System.out.println("LITERAL\tADDRESS");
System.out.println("-----------------");
for(int i=0;i<litTabLine;i++)
System.out.println(LitTab[i][0]+"\t"+LitTab[i][1]);
System.out.println("------------------");

//intialization of POOLTAB
for(int i=0;i<litTabLine;i++)
{
if(LitTab[i][0]!=null && LitTab[i+1][0]!=null ) //if literals are present
{
if(i==0)
{
PoolTab[poolTabLine]=i+1;
poolTabLine++;
}
else if(Integer.parseInt(LitTab[i][1])<(Integer.parseInt(LitTab[i+1][1]))-1)
{
PoolTab[poolTabLine]=i+2;
poolTabLine++;
}
}
}
//print pool table
System.out.println("\n\n POOL TABLE ");
System.out.println("-----------------");
System.out.println("LITERAL NUMBER");
System.out.println("-----------------");
for(int i=0;i<poolTabLine;i++)
System.out.println(PoolTab[i]);
System.out.println("------------------");
// Always close files.
bufferedReader.close();
}
}
```

OUTPUT:

```
____________________________________________________

START   100

        READ    A

LABLE   MOVER           A,B

        LTORG

                ='5'

                ='1'

                ='6'

                ='7'

        MOVEM           A,B

        LTORG

                ='2'

LOOP    READ    B

A       DS      1

B       DC      '1'

                ='1'

        END
        SYMBOL TABLE
--------------------------
SYMBOL      ADDRESS     LENGTH
--------------------------
LABLE 102   1
LOOP  111   1
A     112   1
B     113   1
--------------------------


        OPCODE TABLE
----------------------------
MNEMONIC    CLASS INFO
----------------------------
READ        IS    (04,1)
MOVER             IS    (04,1)
LTORG       AD    R11
MOVEM             IS    (04,1)
LTORG       AD    R11
READ        IS    (04,1)
DS          DL    R7
DC          DL    R7
END         AD    R11
----------------------------
```

```
  LITERAL TABLE
-----------------
LITERAL       ADDRESS
-----------------
='5'     104
='1'     105
='6'     106
='7'     107
='2'     110
='1'     114
------------------



  POOL TABLE
-----------------
LITERAL NUMBER
-----------------
1
5
6
------------------
```

Pass-2 Assembler Program

Progarm:

```
package spos;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;

public class Pass2 {
	public static void main(String[] Args) throws IOException{
		BufferedReader b1 = new BufferedReader(new FileReader("/home/omkar/eclipse-
workspace/spos/src/intermediate.txt"));
		BufferedReader b2 = new BufferedReader(new FileReader("/home/omkar/eclipse-
workspace/spos/src/symtab.txt"));
		BufferedReader b3 = new BufferedReader(new FileReader("/home/omkar/eclipse-
workspace/spos/src/littab.txt"));
		FileWriter f1 = new FileWriter("/home/omkar/eclipse-workspace/spos/src/Pass2.txt");
		HashMap<Integer, String> symSymbol = new HashMap<Integer, String>();
		HashMap<Integer, String> litSymbol = new HashMap<Integer, String>();
		HashMap<Integer, String> litAddr = new HashMap<Integer, String>();
		String s;
		int symtabPointer=1,littabPointer=1,offset;
		while((s=b2.readLine())!=null){
			String word[]=s.split("\t\t\t");
			symSymbol.put(symtabPointer++,word[1]);
		}
		while((s=b3.readLine())!=null){
			String word[]=s.split("\t\t");
			litSymbol.put(littabPointer,word[0]);
			litAddr.put(littabPointer++,word[1]);
		}
		while((s=b1.readLine())!=null){
			if(s.substring(1,6).compareToIgnoreCase("IS,00")==0){
				f1.write("+ 00 0 000\n");
			}
			else if(s.substring(1,3).compareToIgnoreCase("IS")==0){
				f1.write("+ "+s.substring(4,6)+" ");
				if(s.charAt(9)==')'){
					f1.write(s.charAt(8)+" ");
					offset=3;
				}
				else{
					f1.write("0 ");
					offset=0;
				}
				if(s.charAt(8+offset)=='S')
					f1.write(symSymbol.get(Integer.parseInt(s.substring(10+offset,s.length()-
1)))+"\n");
				else
					f1.write(litAddr.get(Integer.parseInt(s.substring(10+offset,s.length()-1)))+"\n");
			}
			else if(s.substring(1,6).compareToIgnoreCase("DL,01")==0){
				String s1=s.substring(10,s.length()-1),s2="";
				for(int i=0;i<3-s1.length();i++)
					s2+="0";
				s2+=s1;
```

```
                        f1.write("+ 00 0 "+s2+"\n");
                }
                else{
                        f1.write("\n");
                }
        }
        f1.close();
        b1.close();
        b2.close();
        b3.close();
    }
}
```

Intermediate.txt

```
(AD,01)(C,200)
(IS,04)(1)(L,1)
(IS,05)(1)(S,1)
(IS,04)(1)(S,1)
(IS,04)(3)(S,3)
(IS,01)(3)(L,2)
(IS,07)(6)(S,4)
(DL,01)(C,5)
(DL,01)(C,1)
(IS,02)(1)(L,3)
(IS,07)(1)(S,5)
(IS,00)
(AD,03)(S,2)+2
(IS,03)(3)(S,3)
(AD,03)(S,6)+1
(DL,02)(C,1)
(DL,02)(C,1)
(AD,02)
(DL,01)(C,1)
```

littab.txt

```
5               206
1               207
1               213
```

symtab.txt

```
A                       211                     1
LOOP                    202                     1
B                       212                     1
NEXT                    208                     1
BACK                    202                     1
LAST                    210                     1
```

Output
Pass2.txt

```
+ 04 1 206
+ 05 1 211
+ 04 1 211
+ 04 3 212
+ 01 3 207
+ 07 6 208
+ 00 0 005
+ 00 0 001
+ 02 1 213
+ 07 1 202
+ 00 0 000

+ 03 3 212




+ 00 0 001
```

Input.txt

```
MACRO
INCR1      &FIRST,&SECOND=DATA9
A          1,&FIRST
L          2,&SECOND
MEND
MACRO
INCR2      &ARG1,&ARG2=DATA5
L          3,&ARG1
ST         4,&ARG2
MEND
PRG2       START
           USING          *,BASE
           INCR1          DATA1
           INCR2          DATA3,DATA4
FOUR       DC               F'4'
FIVE       DC             F'5'
BASE       EQU              8
TEMP       DS               1F
           DROP           8
           END
```

MACRO.java

```
import java.util.*;
import java.io.*;
class MACRO
{
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("\n*********PASS-1 MACROPROCESSOR***********\n");
System.out.println("MACRO NAME TABLE (MNT)\n");
System.out.println("i macro loc\n");
display(mnt,mntc,3);
System.out.println("\n");
System.out.println("ARGUMENT LIST ARRAY(ALA) for Pass1\n");
display(ala,alac,2);
System.out.println("\n");
System.out.println("MACRO DEFINITION TABLE (MDT)\n");
display(mdt,mdtc,1);
System.out.println("\n");
}
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("UNABLE TO END 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:

```
Microsoft Windows [Version 10.0.19044.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Prerana>cd  Desktop

C:\Users\Prerana\Desktop>javac MACRO.java

C:\Users\Prerana\Desktop>java  MACRO

*********PASS-1 MACROPROCESSOR***********

MACRO NAME TABLE (MNT)

i macro loc

1 INCR1 1
2 INCR2 5


ARGUMENT LIST ARRAY(ALA) for Pass1

0 &FIRST
1 &SECOND
2 &ARG1
3 &ARG2


MACRO DEFINITION TABLE (MDT)

INCR1      &FIRST,&SECOND=DATA9
A          1,#0
L          2,#1
MEND
INCR2      &ARG1,&ARG2=DATA5
L          3,#2
ST         4,#3
MEND
```