Pass 1 Assembler INPUT :

```
import java.io.*;
class SymTab
{
	public static void main(String args[])throws Exception
	{
		FileReader FP=new FileReader("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[2])-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"))          //if
Assembler Directive
                    {
                        OpTab[opTabLine][1]="AD";
                        OpTab[opTabLine][2]="(01)";

                    }
                    else if(tokens[1].equalsIgnoreCase("END"))       //if
Assembler Directive
                    {
                        OpTab[opTabLine][1]="AD";
                        OpTab[opTabLine][2]="(02)";

                    }
                    else if(tokens[1].equalsIgnoreCase("ORIGIN"))
        //if Assembler Directive
                    {
                        OpTab[opTabLine][1]="AD";
                        OpTab[opTabLine][2]="(03)";

                    }
                    else if(tokens[1].equalsIgnoreCase("EQU"))       //if
Assembler Directive
                    {
                        OpTab[opTabLine][1]="AD";
```

```
                                        OpTab[opTabLine][2]="(04)";

                                }
                                else if(tokens[1].equalsIgnoreCase("LTORG"))
//if Assembler Directive
                                {
                                        OpTab[opTabLine][1]="AD";
                                        OpTab[opTabLine][2]="(05)";

                                }
                                else if(tokens[1].equalsIgnoreCase("DS"))
                                {
                                        OpTab[opTabLine][1]="DL";
                                        OpTab[opTabLine][2]="(01)";

                                }
                                else if(tokens[1].equalsIgnoreCase("DC"))
                                {
                                        OpTab[opTabLine][1]="DL";
                                        OpTab[opTabLine][2]="(02)";

                                }
                                else if(tokens[1].equalsIgnoreCase("MOVER"))
                                {
                                        OpTab[opTabLine][1]="IS";
                                        OpTab[opTabLine][2]="(04,1)";
                                }
                                else if(tokens[1].equalsIgnoreCase("ADD"))
                                {
                                        OpTab[opTabLine][1]="IS";
                                        OpTab[opTabLine][2]="(01,1)";
                                }
                                else if(tokens[1].equalsIgnoreCase("MOVEM"))
                                {
                                        OpTab[opTabLine][1]="IS";
                                        OpTab[opTabLine][2]="(05,1)";
                                }
                                else if(tokens[1].equalsIgnoreCase("PRINT"))
                                {
                                        OpTab[opTabLine][1]="IS";
                                        OpTab[opTabLine][2]="(10,1)";
                                }

                        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();
        }
}
```

Pass 1 Assembler INPUT FILE :

```
        START           100
A       DS      3
L1      MOVER           AREG,B
        ADD     AREG,C
        MOVEM           AREG,='2'
        MOVEM           AREG,='3'
D       EQU     A+1
        LTORG
                ='2'
                ='3'
L2      PRINT D
        MOVEM           AREG,='4'
        MOVEM           AREG,='5'
        ORIGIN          L2+1
        LTORG
                ='4'
                ='5'
B       DC      19
C       DC      17
        END
```

Pass 1 Assembler OUTPUT:

SYMBOL TABLE

SYMBOL	ADDRESS	LENGTH
A	100	1
L1	101	1
D	105	1
L2	109	1
B	116	1
C	117	1

OPCODE TABLE

MNEMONIC	CLASS	INFO
DS	DL	(01)
MOVER	IS	(04,1)
ADD	IS	(01,1)
MOVEM	IS	(05,1)
MOVEM	IS	(05,1)
EQU	AD	(04)
LTORG	AD	(05)
PRINT	IS	(10,1)
MOVEM	IS	(05,1)
MOVEM	IS	(05,1)
ORIGIN	AD	(03)
LTORG	AD	(05)
DC	DL	(02)
DC	DL	(02)
END	AD	(02)

LITERAL TABLE

LITERAL	ADDRESS
='2'	107
='3'	108
='4'	114
='5'	115

POOL TABLE

LITERAL NUMBER
#1
#3

Pass 2 Assembler INPUT :

```
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("intermediate.txt"));
		BufferedReader b2 = new BufferedReader(new FileReader("symtab.txt"));
		BufferedReader b3 = new BufferedReader(new FileReader("littab.txt"));
		FileWriter f1 = new FileWriter("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();
    }
```

Pass 2 Assembler INPUT FILE :

Intermediate Code :
(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)

Symbol Table :

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

Literal Table :

5	206
1	207
1	213

Pass 2 OUTPUT :

machine code :
+ 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