Program:

```
import java.util.Scanner;
public class BestFit {
                static void bestFit(int blockSize[], int m, int processSize[],int n)
                {
                        int allocation[] = new int[n];
                        for (int i = 0; i < allocation.length; i++) {
                                allocation[i] = -1;
                        }
                        for (int i=0; i<n; i++)
                        {
                                int bestIdx = -1;
                                for (int j=0; j<m; j++)
                                {
                                        if (blockSize[j] >= processSize[i])
                                        {
                                                if (bestIdx == -1)
                                                        bestIdx = j;
                                                else if (blockSize[bestIdx] > blockSize[j])
                                                        bestIdx = j;
                                        }
                                }
                                if (bestIdx != -1)
                                {
                                        allocation[i] = bestIdx;
                                        blockSize[bestIdx] -= processSize[i];
                                }
                        }

                        System.out.println("\nProcess No.\tProcess Size\tBlock no.");
                        for (int i = 0; i < n; i++)
                        {
                                System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t");
                                if (allocation[i] != -1) {
                                        System.out.print(allocation[i] + 1);}
                                else {
                                        System.out.print("Not Allocated");}
                                System.out.println();
                        }
                }


        public static void main(String[] args) {
                int m,n,num;
                Scanner in=new Scanner(System.in);
                System.out.print("Enter how many number of blocks you want to enter:");
                m=in.nextInt();
                int blockSize[]=new int[m];
                for(int i=0;i<m;i++) {
                        System.out.print("Enter Data:");
                        num=in.nextInt();
                        blockSize[i]=num;
                }
                System.out.print("Enter how many number of process you want to enter:");
```

```
		n=in.nextInt();
		int processSize[]=new int[n];
		for(int i=0;i<n;i++) {
			System.out.print("Enter Data:");
			num=in.nextInt();
			processSize[i]=num;
		}
		bestFit(blockSize, m, processSize, n);

	}

}
```

Output:

```
Enter how many number of blocks you want to enter:4
Enter Data:10
Enter Data:15
Enter Data:15
Enter Data:15
Enter how many number of process you want to enter:4
Enter Data:10
Enter Data:15
Enter Data:14
Enter Data:16

Process No.   Process Size   Block no.
 1            10              1
 2            15              2
 3            14              3
 4            16             Not Allocated
```

Program:

```
import java.util.Scanner;
public class FirstFit {
		static void firstFit(int blockSize[], int m,int processSize[], int n)
		{
			int allocation[] = new int[n];
			for (int i = 0; i < allocation.length; i++) {
				allocation[i] = -1;
			}
			for (int i = 0; i < n; i++)
			{
				for (int j = 0; j < m; j++)
				{
					if (blockSize[j] >= processSize[i])
					{
						allocation[i] = j;
						blockSize[j] -= processSize[i];
						break;
					}
				}
			}
			System.out.println("\nProcess No.\tProcess Size\tBlock no.");
			for (int i = 0; i < n; i++)
			{
				System.out.print(" " + (i+1) + "\t\t" +processSize[i] + "\t\t");
				if (allocation[i] != -1) {
					System.out.print(allocation[i] + 1);
				}
				else {
					System.out.print("Not Allocated");
					}
				System.out.println();
			}
		}

	public static void main(String[] args) {
		int m,n,num;
		Scanner in=new Scanner(System.in);
		System.out.print("Enter how many number of blocks you want to enter:");
		m=in.nextInt();
		int blockSize[]=new int[m];
		for(int i=0;i<m;i++) {
			System.out.print("Enter Data:");
			num=in.nextInt();
			blockSize[i]=num;
		}
		System.out.print("Enter how many number of process you want to enter:");
		n=in.nextInt();
		int processSize[]=new int[n];
		for(int i=0;i<n;i++) {
			System.out.print("Enter Data:");
			num=in.nextInt();
```

```
                              processSize[i]=num;
                    }
                    firstFit(blockSize, m, processSize, n);

          }

}
```

Output:

```
Enter how many number of blocks you want to enter:4
Enter Data:10
Enter Data:15
Enter Data:15
Enter Data:15
Enter how many number of process you want to enter:4
Enter Data:10
Enter Data:15
Enter Data:14
Enter Data:16

Process No.    Process Size    Block no.
 1             10              1
 2             15              2
 3             14              3
 4             16              Not Allocated
```

Program:

```
import java.util.Arrays;
import java.util.Scanner;

public class NextFit {

        static void NextFit(int blockSize[], int m, int processSize[], int n) {

                int allocation[] = new int[n], j = 0;
                Arrays.fill(allocation, -1);
                for (int i = 0; i < n; i++) {
                        int count=0;
                        while (count<m){
                                count++;
                                if (blockSize[j] >= processSize[i]) {
                                        allocation[i] = j;

                                        blockSize[j] -= processSize[i];
                                        break;
                                }
                                j=(j + 1) % m;
                                count+=1;
                        }
                }

                System.out.print("\nProcess No.\tProcess Size\tBlock no.\n");
                for (int i = 0; i < n; i++) {
                        System.out.print( i + 1 + "\t\t" + processSize[i]+ "\t\t");
                        if (allocation[i] != -1) {
                                System.out.print(allocation[i] + 1);
                        } else {
                                System.out.print("Not Allocated");
                        }
                        System.out.println("");
                }
        }

    public static void main(String[] args) {
        int m,n,num;
        Scanner in=new Scanner(System.in);
        System.out.print("Enter how many number of blocks you want to enter:");
        m=in.nextInt();
        int blockSize[]=new int[m];
        for(int i=0;i<m;i++) {
                System.out.print("Enter Data:");
                num=in.nextInt();
                blockSize[i]=num;
        }
        System.out.print("Enter how many number of process you want to enter:");
        n=in.nextInt();
        int processSize[]=new int[n];
```

```
                for(int i=0;i<n;i++) {
                        System.out.print("Enter Data:");
                        num=in.nextInt();
                        processSize[i]=num;
                }
                NextFit(blockSize, m, processSize, n);
        }

}
```

Output:

```
Enter how many number of blocks you want to enter:4
Enter Data:10
Enter Data:15
Enter Data:15
Enter Data:15
Enter how many number of process you want to enter:4
Enter Data:10
Enter Data:14
Enter Data:15
Enter Data:16

Process No.   Process Size   Block no.
1             10             1
2             14             2
3             15             3
4             16             Not Allocated
```

Program:

```
import java.util.Scanner;
public class WorstFit {

                static void worstFit(int blockSize[], int m, int processSize[],int n)
                {
                        int allocation[] = new int[n];
                        for (int i = 0; i < allocation.length; i++) {
                                allocation[i] = -1;}
                        for (int i=0; i<n; i++)
                        {
                                int wstIdx = -1;
                                for (int j=0; j<m; j++)
                                {
                                        if (blockSize[j] >= processSize[i])
                                        {
                                                if (wstIdx == -1)
                                                        wstIdx = j;
                                                else if (blockSize[wstIdx] < blockSize[j])
                                                        wstIdx = j;
                                        }
                                }
                                if (wstIdx != -1)
                                {
                                        allocation[i] = wstIdx;
                                        blockSize[wstIdx] -= processSize[i];
                                }
                        }

                        System.out.println("\nProcess No.\tProcess Size\tBlock no.");
                        for (int i = 0; i < n; i++)
                        {
                                System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t");
                                if (allocation[i] != -1)
                                        System.out.print(allocation[i] + 1);
                                else
                                        System.out.print("Not Allocated");
                                System.out.println();
                        }
                }

        public static void main(String[] args) {
                int m,n,num;
                Scanner in=new Scanner(System.in);
                System.out.print("Enter how many number of blocks you want to enter:");
                m=in.nextInt();
                int blockSize[]=new int[m];
                for(int i=0;i<m;i++) {
                        System.out.print("Enter Data:");
                        num=in.nextInt();
                        blockSize[i]=num;
                }
                System.out.print("Enter how many number of process you want to enter:");
                n=in.nextInt();
                int processSize[]=new int[n];
                for(int i=0;i<n;i++) {
                        System.out.print("Enter Data:");
                        num=in.nextInt();
                        processSize[i]=num;
                }
                worstFit(blockSize, m, processSize, n);
        }

}
```

Output:

```
Enter how many number of blocks you want to enter:4
Enter Data:10
Enter Data:15
Enter Data:15
Enter Data:15
Enter how many number of process you want to enter:4
Enter Data:10
Enter Data:14
Enter Data:15
Enter Data:16

Process No.   Process Size   Block no.
 1            10             2
 2            14             3
 3            15             4
 4            16             Not Allocated
```

Program:

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

class MnemonicTable {
        public String mnemonic;
        public String opcode;
        public int num;
        public MnemonicTable(String mnemonic,String opcode,int num ){
                this.mnemonic=mnemonic;
                this.opcode=opcode;
                this.num=num;
        }

}

public class Pass_1 {

   Map<String,MnemonicTable> is=new Hashtable<String,MnemonicTable>();
   ArrayList<String>symtab=new ArrayList<>();
   ArrayList<Integer> symaddr=new ArrayList<>();
   ArrayList<String>littab=new ArrayList<>();
   ArrayList<Integer> litaddr=new ArrayList<>();
   ArrayList<Integer>pooltab=new ArrayList<>();
   int LC=0;
   public void createIS() throws Exception  {
        Scanner input=new Scanner(System.in);
      MnemonicTable m1=new MnemonicTable("STOP","00", 0);
      is.put("STOP",m1);
      MnemonicTable m2=new MnemonicTable("ADD","01", 0);
```

```
    is.put("ADD",m2);
    MnemonicTable m3=new MnemonicTable("SUB","02", 0);
    is.put("SUB",m3);
    MnemonicTable m4=new MnemonicTable("MULT","03", 0);
    is.put("MULT",m4);
    MnemonicTable m5=new MnemonicTable("MOVER","04", 0);
    is.put("MOVER",m5);
    MnemonicTable m6=new MnemonicTable("MOVEM","05", 0);
    is.put("MOVEM",m6);
    MnemonicTable m7=new MnemonicTable("COMP","06", 0);
    is.put("COMP",m7);
    MnemonicTable m8=new MnemonicTable("BC","07", 0);
    is.put("BC",m8);
    MnemonicTable m9=new MnemonicTable("DIV","08", 0);
    is.put("DIV",m9);
    MnemonicTable m10=new MnemonicTable("READ","09", 0);
    is.put("READ",m10);
    MnemonicTable m11=new MnemonicTable("PRINT","10", 0);
        is.put("PRINT",m11);
        /*BufferedWriter wr=new BufferedWriter(new FileWriter("ic.txt"));
        String string=input.next();
        wr.write(string);
        wr.flush();
        wr.close();      */
}
public void generateIC() throws Exception {
    BufferedWriter wr=new BufferedWriter(new FileWriter("ic.txt"));
    BufferedReader br=new BufferedReader(new FileReader("input.asm"));
    String line=" ";
    pooltab.add(0, 0);
    wr.write("---------------------\n  Intermediate Code\n---------------------\n");
    while((line=br.readLine())!=null) {
```

```
String[] split=line.split("\\s+");
if(split[0].length()>0) {
    //it is a label
    if(!symtab.contains(split[0])) {
        symtab.add(split[0]);
        symaddr.add(LC);
    }
    else {
        int index=symtab.indexOf(split[0]);
        symaddr.remove(index);
        symaddr.add(index,LC);
    }
}

if(split[1].equals("START")) {
    LC=Integer.parseInt(split[2]);
    wr.write("(AD,01)(C,"+split[2]+") \n");
}
else if(split[1].equals("ORIGIN")) {
    if(split[2].contains("+") || split[2].contains("-")) {
        LC=getAddress(split[2]);
    }
    else {
        LC=symaddr.get(symtab.indexOf(split[2]));
    }
}
else if(split[1].equals("EQU")) {
    int addr=0;
    if(split[2].contains("+") || split[2].contains("-")) {
        addr=getAddress(split[2]);
    }
```

```
        else {
            addr=symaddr.get(symtab.indexOf(split[2]));
        }
        if(!symtab.contains(split[0])) {
            symtab.add(split[0]);
            symaddr.add(addr);
        }
        else {
            int index=symtab.indexOf(split[0]);
            symaddr.remove(index);
            symaddr.add(index,addr);
        }
    }
    else if(split[1].equals("LTORG") || split[1].equals("END")) {
        if(litaddr.contains(0)) {
            for(int i=pooltab.get(pooltab.size()-1);i<littab.size();i++) {
                if(litaddr.get(i)==0) {
                    litaddr.remove(i);
                    litaddr.add(i, LC);
                    LC++;
                }
            }
            if(!split[1].equals("END")) {
                pooltab.add(littab.size());
                wr.write("\n(AD,05)\n");
            }
            else
                wr.write("(AD,04) \n");
        }
    }
    else if(split[1].contains("DS")) {
        LC+=Integer.parseInt(split[2]);
    wr.write("(DL,01) (C,"+split[2]+") \n");
}
else if(split[1].equals("DC")) {
    LC++;
    wr.write("\n(DL,02) (C,"+split[2].replace("'", "").replace("'", "")+") \n");
}
else if(is.containsKey(split[1])) {
    wr.write("(IS,"+is.get(split[1]).opcode+") ");
    if(split.length>2 && split[2]!=null) {
        String reg=split[2].replace(",","");
        if(reg.equals("AREG")) {
            wr.write("(1) ");
        }
        else if(reg.equals("BREG")) {
            wr.write("(2) ");
        }
        else if(reg.equals("CREG")) {
            wr.write("(3) ");
        }
        else if(reg.equals("DREG")) {
            wr.write("(4) ");
        }
        else {
            if(symtab.contains(reg)) {
                wr.write("(S,"+symtab.indexOf(reg)+")\n");
            }
            else {
                symtab.add(reg);
                symaddr.add(0);
                wr.write("(S,"+symtab.indexOf(reg)+") \n");


            }
            }
        }

        if(split.length>3 && split[3]!=null) {
            if(split[3].contains("=")) {
                String norm=split[3].replace("=","").replace("'", "").replace("'", "");
                if(!littab.contains(norm)) {
                    littab.add(norm);
                    litaddr.add(0);
                    wr.write("(L,"+littab.indexOf(norm)+")");
                }
                else {
                    wr.write("L,"+littab.indexOf(norm)+")");
                }

            }
            else if(symtab.contains(split[3])) {
                wr.write("(S,"+symtab.indexOf(split[3])+") \n");

            }
            else {
                symtab.add(split[3]);
                symaddr.add(0);
                wr.write("(S,"+symtab.indexOf(split[3])+") \n");

            }
        }
        LC++;
    }
}
wr.flush();
BufferedReader icr=new BufferedReader(new FileReader("ic.txt"));
```

```
while(icr.ready()){
System.out.print((char)icr.read());
}
icr.close();
wr.close();
BufferedWriter br1=new BufferedWriter(new FileWriter("sym.txt"));
br1.write("-------------------\n    Symbol Table\n-------------------\nSymbol    Address\n");
for(int i=0;i<symtab.size();i++) {
   br1.write("  "+symtab.get(i)+"       "+symaddr.get(i)+"\n");
}
br1.flush();
BufferedReader br1r=new BufferedReader(new FileReader("sym.txt"));
while(br1r.ready()){
System.out.print((char)br1r.read());
}
br1r.close();
br1.close();
BufferedWriter br2=new BufferedWriter(new FileWriter("lit.txt"));
br2.write("-----------------------\n    Literal Table\n-----------------------\nLiteral
Address\n");
for(int i=0;i<littab.size();i++) {
   br2.write("='"+littab.get(i)+"'           "+litaddr.get(i)+"\n");
}
br2.flush();
BufferedReader br2r=new BufferedReader(new FileReader("lit.txt"));
while(br2r.ready()){
System.out.print((char)br2r.read());
}
br2r.close();
br2.close();
BufferedWriter br3=new BufferedWriter(new FileWriter("pool.txt"));
BufferedReader br3r=new BufferedReader(new FileReader("pool.txt"));
```

```
        br3.write("-----------------------------\n        Pool Table\n-----------------------------\nPool Index
    Literal Index\n");
        for(int i=0;i<pooltab.size();i++){
            br3.write("     "+i+"              "+pooltab.get(i)+"\n");
        }
        br3.flush();
        while(br3r.ready()){
            System.out.print((char)br3r.read());
        }
        br3r.close();

    }
    private int getAddress(String string) {
        int temp=0;
        if(string.contains("+")) {
            String sp[]=string.split("\\+");
            int ad=symaddr.get(symtab.indexOf(sp[0]));
            temp=ad+Integer.parseInt(sp[1]);
        }
        else if(string.contains("-")) {
            String sp[]=string.split("\\-");
            int ad=symaddr.get(symtab.indexOf(sp[0]));
            temp=ad-Integer.parseInt(sp[1]);
        }
        return temp;
    }
    public static void main(String[] args) throws Exception {
        Pass_1 p=new Pass_1();
        p.createIS();
        p.generateIC();
    }
}
```

Input:

```
START    100
A    DS    3
L1    MOVEM    AREG,    B
    ADD    AREG,    C
    MOVER    AREG,    ='12'
D    EQU    A+1
    LTORG
L2    PRINT    D
    ORIGIN    A-1
    MOVER    AREG,    ='5'
C    DC    '5'
    ORIGIN    L2+1
    STOP
B    DC    '19'
    END
```

Output:

```
---------------------
  Intermediate Code
---------------------
(AD,01)(C,100)
(DL,01) (C,3)
(IS,05) (1) (S,2)
(IS,01) (1) (S,3)
(IS,04) (1) (L,0)
(AD,05)
(IS,10) (S,4)
(IS,04) (1) (L,1)
(DL,02) (C,5)
(IS,00)
(DL,02) (C,19)
(AD,04)
-------------------
    Symbol Table
-------------------
Symbol    Address
  A       100
  L1       103
  B       109
  C       100
  D       101
  L2       107
-----------------------
    Literal Table
-----------------------
Literal       Address
='12'          106
='5'          110
-----------------------------
        Pool Table
-----------------------------
Pool Index    Literal Index
     0             0
     1             1
```

PASS 2:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.ArrayList;

class TableRow {
	String symbol;
	int address;
	int index;

	public TableRow(String symbol, int address) {
		super();
		this.symbol = symbol;
		this.address = address;

	}
	public TableRow(String symbol, int address,int index) {
		super();
		this.symbol = symbol;
		this.address = address;
		this.index=index;

	}
	public int getIndex() {
		return index;
	}
	public void setIndex(int index) {
		this.index = index;
	}
	public String getSymbol() {
		return symbol;
	}
	public void setSymbol(String symbol) {
		this.symbol = symbol;
	}
	public int getAddress() {
		return address;
	}
	public void setAddress(int address) {
		this.address = address;
	}

}
public class Pass_2 {
	ArrayList<TableRow> SYMTAB,LITTAB;

	public Pass_2()
	{
		SYMTAB=new ArrayList<>();
		LITTAB=new ArrayList<>();
	}
	public static void main(String[] args) {
				Pass_2 pass2=new Pass_2();

				try {
					pass2.generateCode("IC.txt");
				} catch (Exception e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
			}
			public void readtables()
			{
				BufferedReader br;
				String line;
				try
				{
					br=new BufferedReader(new FileReader("SYMTAB.txt"));
					while((line=br.readLine())!=null)
					{
						String parts[]=line.split("\\s+");
						SYMTAB.add(new TableRow(parts[1],
		Integer.parseInt(parts[2]),Integer.parseInt(parts[0]) ));
					}
					br.close();
					br=new BufferedReader(new FileReader("LITTAB.txt"));
					while((line=br.readLine())!=null)
					{
						String parts[]=line.split("\\s+");
						LITTAB.add(new TableRow(parts[1],
		Integer.parseInt(parts[2]),Integer.parseInt(parts[0])));
					}
					br.close();
				}
				catch (Exception e) {
					System.out.println(e.getMessage());
				}
			}

			public void generateCode(String filename) throws Exception
			{
				readtables();
				BufferedReader br=new BufferedReader(new FileReader(filename));

				BufferedWriter bw=new BufferedWriter(new FileWriter("PASS2.txt"));
				String line,code;
				while((line=br.readLine())!=null)
				{
					String parts[]=line.split("\\s+");
					if(parts[0].contains("AD")||parts[0].contains("DL,02"))
					{
						bw.write("\n");
						continue;
					}
					else if(parts.length==2)
					{
						if(parts[0].contains("DL")) //DC INSTR
					{
						parts[0]=parts[0].replaceAll("[^0-9]", "");
						if(Integer.parseInt(parts[0])==1)
						{
							int 
constant=Integer.parseInt(parts[1].replaceAll("[^0-9]", ""));
							code="00\t0\t"+String.format("%03d", 
constant)+"\n";
							bw.write(code);


						}
					}
					else if(parts[0].contains("IS"))
					{
						int opcode=Integer.parseInt(parts[0].replaceAll("[^0-9]", 
""));
						if(opcode==10)
						{
							if(parts[1].contains("S"))
							{
								int 
symIndex=Integer.parseInt(parts[1].replaceAll("[^0-9]", ""));
								code=String.format("%02d", 
opcode)+"\t0\t"+String.format("%03d", SYMTAB.get(symIndex-1).getAddress())+"\n";
								bw.write(code);
							}
						else if(parts[1].contains("L"))
							{
								int 
symIndex=Integer.parseInt(parts[1].replaceAll("[^0-9]", ""));
								code=String.format("%02d", 
opcode)+"\t0\t"+String.format("%03d", LITTAB.get(symIndex-1).getAddress())+"\n";
								bw.write(code);
							}

						}
					}
				}
				else if(parts.length==1 && parts[0].contains("IS"))
				{
					int opcode=Integer.parseInt(parts[0].replaceAll("[^0-9]", ""));
					code=String.format("%02d", 
opcode)+"\t0\t"+String.format("%03d", 0)+"\n";
					bw.write(code);
				}
				else if(parts[0].contains("IS") && parts.length==3) //All OTHER IS 
INSTR
				{
				int opcode=	Integer.parseInt(parts[0].replaceAll("[^0-9]", ""));

				int regcode=Integer.parseInt(parts[1]);

				if(parts[2].contains("S"))
				{
```

```
                                        int symIndex=Integer.parseInt(parts[2].replaceAll("[^0-9]", ""));
                                        code=String.format("%02d",
opcode)+"\t"+regcode+"\t"+String.format("%03d", SYMTAB.get(symIndex-
1).getAddress())+"\n";
                                        bw.write(code);
                              }
                              else if(parts[2].contains("L"))
                              {
                                        int symIndex=Integer.parseInt(parts[2].replaceAll("[^0-9]", ""));
                                        code=String.format("%02d",
opcode)+"\t"+regcode+"\t"+String.format("%03d", LITTAB.get(symIndex-
1).getAddress())+"\n";
                                        bw.write(code);
                              }
                              }
                    }
                    bw.close();
                    br.close();
                    System.out.println("Pass2 Processing done............ :)");
          }
}
```

Input:

```
---------------------
  Intermediate Code
---------------------
(AD,01)         (C,100)
(IS,04) 1       (L,1)
(IS,05) 2       (S,02)
(IS,01) 1       (L,2)
(DL,01)         (C,5)
(DL,01)         (C,2)
(IS,04) 1       (S,03)
(DL,01)         (C,5)
(DL,02)         (C,2)
(AD,02)
-------------------
   Symbol Table
-------------------
Index   Symbol Address
1       L1      100
2       X       106
3       Y       107

-----------------------
   Literal Table
-----------------------
Literal     Address
5       104
2       105
```

Output:

```
Pass2 Processing done............ :)
PS C:\Users\Bhushan Kadam\Desktop\Practical\LP1\A1-Assembler\Assembler\Pass_2
Assembler>
```

```
Pass_2 Output-
04      1       104
05      2       106
01      1       105
00      0       005
00      0       002
04      1       107
00      0       005
```

Program:

```
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 MacroPass1 {

        public static void main(String[] args) throws IOException{
                BufferedReader br=new BufferedReader(new
FileReader("macro_input.asm"));

                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?kpdtp:(kpdtp+1))+"\n");
                                        continue;
                                }
                                for(int i=1;i<parts.length;i++) //processing of parameters
                                {
                                        parts[i]=parts[i].replaceAll("[&,]", "");
                                        //System.out.println(parts[i]);
                                        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;
                              //System.out.println("KP="+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("Macro Pass1 Processing done....... :)");
        }

}
```

Output:

```
Macro Pass1 Processing done....... :)
PS C:\Users\Bhushan Kadam\Desktop\Practical\LP1\A2-Macro\Macro\Pass_1>
```

Macro Pass1 file

```
---------------------
  Macro Input
---------------------
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


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

---------------------
  MDT
---------------------
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

---------------------
  MNT
---------------------
M1      2       2       1       1
M2      2       2       6       3

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

Pass 2:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.HashMap;
import java.util.Vector;


class MNTEntry {
	String name;
	int pp,kp,mdtp,kpdtp;
	public MNTEntry(String name, int pp, int kp, int mdtp, int kpdtp) {
		super();
		this.name = name;
		this.pp = pp;
		this.kp = kp;
		this.mdtp = mdtp;
		this.kpdtp = kpdtp;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getPp() {
		return pp;
	}
	public void setPp(int pp) {
		this.pp = pp;
	}
	public int getKp() {
		return kp;
	}
	public void setKp(int kp) {
		this.kp = kp;
	}
	public int getMdtp() {
		return mdtp;
	}
	public void setMdtp(int mdtp) {
		this.mdtp = mdtp;
	}
	public int getKpdtp() {
		return kpdtp;
	}
	public void setKpdtp(int kpdtp) {
		this.kpdtp = kpdtp;
	}
```

```
}



public class MacroPass2 {

        public static void main(String[] args) throws Exception {
                BufferedReader irb=new BufferedReader(new FileReader("intermediate.txt"));
                BufferedReader mdtb=new BufferedReader(new FileReader("mdt.txt"));
                BufferedReader kpdtb=new BufferedReader(new FileReader("kpdt.txt"));
                BufferedReader mntb=new BufferedReader(new FileReader("mnt.txt"));

                FileWriter fr=new FileWriter("pass2.txt");

                HashMap<String, MNTEntry> mnt=new HashMap<>();
                HashMap<Integer, String> aptab=new HashMap<>();
                HashMap<String,Integer> aptabInverse=new HashMap<>();

                Vector<String>mdt=new Vector<String>();
                Vector<String>kpdt=new Vector<String>();

                int pp,kp,mdtp,kpdtp,paramNo;
                String line;
                while((line=mdtb.readLine())!=null)
                {
                        mdt.addElement(line);
                }
                while((line=kpdtb.readLine())!=null)
                {
                        kpdt.addElement(line);
                }
                while((line=mntb.readLine())!=null)
                {
                        String parts[]=line.split("\\s+");
                        mnt.put(parts[0], new MNTEntry(parts[0], Integer.parseInt(parts[1]),
Integer.parseInt(parts[2]), Integer.parseInt(parts[3]), Integer.parseInt(parts[4])));

                }

                while((line=irb.readLine())!=null)
                {
                        String []parts=line.split("\\s+");
                        if(mnt.containsKey(parts[0]))
                        {
                                pp=mnt.get(parts[0]).getPp();
                                kp=mnt.get(parts[0]).getKp();
                                kpdtp=mnt.get(parts[0]).getKpdtp();
                                mdtp=mnt.get(parts[0]).getMdtp();
                                paramNo=1;
                                for(int i=0;i<pp;i++)
                                {
                                        parts[paramNo]=parts[paramNo].replace(",", "");
                                        aptab.put(paramNo, parts[paramNo]);
                                        aptabInverse.put(parts[paramNo], paramNo);
                                        paramNo++;
                                }
                                int j=kpdtp-1;
                                for(int i=0;i<kp;i++)
                                {
                                        String temp[]=kpdt.get(j).split("\t");
                                        aptab.put(paramNo,temp[1]);
                                        aptabInverse.put(temp[0],paramNo);
                                        j++;
                                        paramNo++;
                                }

                                for(int i=pp+1;i<parts.length;i++)
                                {
                                        parts[i]=parts[i].replace(",", "");
                                        String splits[]=parts[i].split("=");
                                        String name=splits[0].replaceAll("&", "");
                                        aptab.put(aptabInverse.get(name),splits[1]);
                                }
                                int i=mdtp-1;
                                while(!mdt.get(i).equalsIgnoreCase("MEND"))
                                {
                                        String splits[]=mdt.get(i).split("\\s+");
                                        fr.write("+");
                                        for(int k=0;k<splits.length;k++)
                                        {
                                                if(splits[k].contains("(P,"))
                                                {
                                                        splits[k]=splits[k].replaceAll("[^0-9]",
"");//not containing number
                                                        String
value=aptab.get(Integer.parseInt(splits[k]));
                                                        fr.write(value+"\t");
                                                }
                                                else
                                                {
                                                        fr.write(splits[k]+"\t");
                                                }
                                        }
                                        fr.write("\n");
                                        i++;
                                }

                                aptab.clear();
                                aptabInverse.clear();
                        }
                        else
                        {
                                fr.write(line+"\n");
                        }

        }

        fr.close();
        mntb.close();
        mdtb.close();
```

```
            kpdtb.close();
            irb.close();
            System.out.println("Macro Pass2 Processing done............ :)");
            }  }
```

Output:

```
Macro Pass2 Processing done............ :)
PS C:\Users\Bhushan Kadam\Desktop\Practical\LP1\A2-Macro\Macro\Pass_2>
```

Macro Pass2 file-

```
---------------------
  Macro Input
---------------------
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

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

---------------------
  Pass2
---------------------
START           100
+MOVER          AREG  10
+ADD    AREG  ='1'
+MOVER          CREG  20
+ADD    CREG  ='5'
+MOVER          BREG  100
+MOVER          AREG  200
+ADD    BREG  ='15'
+ADD    AREG  ='10'
END
```

Program:

JAVA File

```
import java.util.Scanner;
public class JNI {
	public native void JniAdd(int no1,int no2);
	public native void JniSub(int no1,int no2);
	public native void JniMult(int no1,int no2);
	public native void JniDiv(double no1,double no2);
	public native void JniPow(int no1,int no2);
	public native void JniSqrt(int no1);
	public native void JniMod(int no1,int no2);
	static { System.load("C:\\Users\\Bhushan 
Kadam\\Desktop\\Practical\\LP1\\DLL\\libJNI.dll");}
	public static void main(String[] args)throws Exception {
		int no1,no2;
		 Scanner in =new Scanner(System.in);
		 JNI MJ=new JNI();
		 System.out.println("JNI using C");
		 System.out.print("Enter first number: ");
		 no1=in.nextInt();
		 double no1f=no1;
		 System.out.print("Enter second number: ");
		 no2=in.nextInt();
		 MJ.JniAdd(no1,no2);
		 MJ.JniSub(no1,no2);
		 MJ.JniMult(no1,no2);
		 MJ.JniDiv((double)no1,(double)no2);
		 MJ.JniPow(no1,no2);
		 MJ.JniSqrt(no2);
		 MJ.JniMod(no1,no2);

	}

}
```

C File_JNI.h

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class JNI_JNI */

#ifndef _Included_JNI_JNI
#define _Included_JNI_JNI
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     JNI_JNI
 * Method:    JniAdd
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_JNI_JNI_JniAdd
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     JNI_JNI
 * Method:    JniSub
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_JNI_JNI_JniSub
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     JNI_JNI
 * Method:    JniMult
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_JNI_JNI_JniMult
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     JNI_JNI
 * Method:    JniDiv
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_JNI_JNI_JniDiv
  (JNIEnv *, jobject, jdouble, jdouble);

/*
 * Class:     JNI_JNI
 * Method:    JniPow
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_JNI_JNI_JniPow
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     JNI_JNI
 * Method:    JniSqrt
 * Signature: (I)V
 */
JNIEXPORT void JNICALL Java_JNI_JNI_JniSqrt
  (JNIEnv *, jobject, jint);

/*
 * Class:     JNI_JNI
 * Method:    JniMod
 * Signature: (II)V
 */
JNIEXPORT void JNICALL Java_JNI_JNI_JniMod
  (JNIEnv *, jobject, jint, jint);

#ifdef __cplusplus
}
#endif
```

```
#endif
```

C file_JNI.c

```
#include <JNI.h>
#include<math.h>

#define PI 3.14159265

JNIEXPORT void JNICALL Java_JNI_JNI_JniAdd
  (JNIEnv *e, jobject obj, jint no1, jint no2)
{
int add=no1+no2;
printf("Addition of nos.= %d",add);
}

JNIEXPORT void JNICALL Java_JNI_JNI_JniSub
  (JNIEnv *e, jobject obj, jint no1, jint no2)
{
  int sub=no1-no2;
  printf("\nSubtraction of nos. is= %d",sub);
}

JNIEXPORT void JNICALL Java_JNI_JNI_JniMult
  (JNIEnv *e, jobject obj, jint no1, jint no2)
{
   int mult=no1*no2;
  printf("\nMultiplication of nos. is= %d",mult);
}


JNIEXPORT void JNICALL Java_JNI_JNI_JniDiv
  (JNIEnv *e, jobject obj, jdouble no1, jdouble no2)
{
  double div=no1/no2;
  printf("\nDivision of nos. is= %.3f",div);
}

JNIEXPORT void JNICALL Java_JNI_JNI_JniMod
  (JNIEnv *e, jobject obj, jint no1, jint no2)
{
   printf("\nRemainder is= %.3f",fmod(no1,no2));
}

JNIEXPORT void JNICALL Java_JNI_JNI_JniPow
  (JNIEnv *e, jobject obj, jint no1, jint no2)
{
   printf("\nPower is= %.3f",pow(no1,no2));
}


JNIEXPORT void JNICALL Java_JNI_JNI_JniSqrt
  (JNIEnv *e, jobject obj, jint no1)
```

```
{
  printf("\nSquare root %d is= %.3f",no1,sqrt(no1));

}
```

Output:

```
Microsoft Windows [Version 10.0.22621.674]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Desktop\Practical\LP1\DLL>javac -h . JNI.java
C:\Users\Desktop\Practical\LP1\DLL> gcc -o libJNI.dll -shared -fPIC -I"C:\Program
Files\Java\jdk-18.0.1.1\include" -I"C:\Program Files\Java\jdk-18.0.1.1\include\win32"

JNI using C
 Enter first number: 21
Enter second number: 15
Addition of nos.= 36
Subtraction of nos. is= 6
Multiplication of nos. is= 315
Division of nos. is= 1.400
Power is= 68122318582951682000.000
Square root 15 is= 3.873
Remainder is= 6.000
```

Program:

Infix.l

```
%{
#include "y.tab.h"
%}
%%
[0-9]+ { yylval.dval=atoi(yytext); return NUMBER;}
[0-9]*"."[0-9]+ { yylval.dval=atof(yytext); return NUMBER;}
[a-zA-Z] { return LETTER; }
"+" { return PLUS;}
"-" { return MINUS;}
"*" { return MULTIPLY;}
"/" { return DIVIDE;}
"(" { return OPEN;}
")" { return CLOSE;}
"\n" { return ENTER;}
"$" { return 0;}
%%
```

Infix.y

```
%{
#include<stdio.h>
#include<math.h>
%}
%union {
double dval;
char symbol;
}
%token<dval>NUMBER
%token<symbol>LETTER
%token PLUS MINUS MULTIPLY DIVIDE OPEN CLOSE ENTER
%left PLUS MINUS
%left DIVIDE MULTIPLY
%nonassoc UMINUS
%type<dval>E
%%
print: E ENTER { printf("\n\v VALID INFIX EXP.......\n"); exit (0); }
;
E:E PLUS E
|
E MINUS E
|
E MULTIPLY E
|
E DIVIDE E
```

```
|
MINUS E %prec UMINUS {$$=-$2;}
|
OPEN E CLOSE { $$=$2;}
|
NUMBER { $$=$1; }
|
LETTER {$$=$1;}
;
%%
int main()
{
printf("\n Enter infix expression: ");
yyparse();
return 0;
}
void yyerror( char *msg)
{
printf("\n INVALID INFIX EXPRESSION.....: ");
}
int yywrap(){return(1);}
```

Output:

```
Ubuntu@Ubuntu-ThinkCentre-M72e:~$ lex infix.l
Ubuntu@Ubuntu-ThinkCentre-M72e:~$ yacc -d infix.y
Ubuntu@Ubuntu-ThinkCentre-M72e:~$ gcc lex.yy.c y.tab.c -w
Ubuntu@Ubuntu-ThinkCentre-M72e:~$ ./a.out

 Enter infix expression: a+b

 VALID INFIX EXP.......

Ubuntu@Ubuntu-ThinkCentre-M72e:~$ ./a.out

 Enter infix expression: (+a+b)

 INVALID INFIX EXPRESSION.....:
```