Input Text

```
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
```

```
 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
```

```
/*
Problem Statement: Implement Pass-II of two pass assembler for pseudo-
machine in Java using object oriented
features. The output of assignment-1 (intermediate file and symbol table)
should be
input for this assignment.
*/
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();
      }
}


/*
OUTPUT:
neha@neha-1011PX:~/Desktop/neha_SPOS/Turn1/A2$ javac Pass2.java
neha@neha-1011PX:~/Desktop/neha_SPOS/Turn1/A2$ java Pass2
neha@neha-1011PX:~/Desktop/neha_SPOS/Turn1/A2$ cat Pass2.txt

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


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     */
```

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:

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

MACRO NAME TABLE (MNT)

i macro loc

1 INCR 1
2 PVG 5


ARGUMENT LIST ARRAY(ALA) for Pass1

0 &ARG3
1 &ARG2


MACRO DEFINITION TABLE (MDT)

INCR &ARG3 &ARG2
ADD AREG &ARG1
MOVER BREG &ARG1
MEND
PVG &ARG2 &ARG1
SUB AREG #1
MOVER CREG & ARG1
MEND
```

```
/*
Problem Statement : Write a Java program for pass-II of a two-pass macro-
processor. The output of assignment-3
(MNT, MDT and file without any macro definitions) should be input for
this assignment.
*/
import java.io.*;
import java.util.HashMap;
import java.util.Vector;

public class macroPass2 {
      public static void main(String[] Args) throws IOException{
            BufferedReader b1 = new BufferedReader(new
FileReader("intermediate.txt"));
            BufferedReader b2 = new BufferedReader(new
FileReader("mnt.txt"));
            BufferedReader b3 = new BufferedReader(new
FileReader("mdt.txt"));
            BufferedReader b4 = new BufferedReader(new
FileReader("kpdt.txt"));
            FileWriter f1 = new FileWriter("Pass2.txt");
            HashMap<Integer,String> aptab=new HashMap<Integer,String>();
            HashMap<String,Integer> aptabInverse=new
HashMap<String,Integer>();
            HashMap<String,Integer> mdtpHash=new
HashMap<String,Integer>();
            HashMap<String,Integer> kpdtpHash=new
HashMap<String,Integer>();
            HashMap<String,Integer> kpHash=new HashMap<String,Integer>();
            HashMap<String,Integer> macroNameHash=new
HashMap<String,Integer>();
            Vector<String>mdt=new Vector<String>();
            Vector<String>kpdt=new Vector<String>();
            String s,s1;
            int i,pp,kp,kpdtp,mdtp,paramNo;
            while((s=b3.readLine())!=null)
                  mdt.addElement(s);
            while((s=b4.readLine())!=null)
                  kpdt.addElement(s);
            while((s=b2.readLine())!=null){
                  String word[]=s.split("\t");
                  s1=word[0]+word[1];
                  macroNameHash.put(word[0],1);
                  kpHash.put(s1,Integer.parseInt(word[2]));
                  mdtpHash.put(s1,Integer.parseInt(word[3]));
                  kpdtpHash.put(s1,Integer.parseInt(word[4]));
            }
            while((s=b1.readLine())!=null){
                  String b1Split[]=s.split("\\s");
                  if(macroNameHash.containsKey(b1Split[0])){
                        pp= b1Split[1].split(",").length-
b1Split[1].split("=").length+1;
                        kp=kpHash.get(b1Split[0]+Integer.toString(pp));

      mdtp=mdtpHash.get(b1Split[0]+Integer.toString(pp));

      kpdtp=kpdtpHash.get(b1Split[0]+Integer.toString(pp));
                        String actualParams[]=b1Split[1].split(",");
                        paramNo=1;
                        for(int j=0;j<pp;j++){
                              aptab.put(paramNo, actualParams[paramNo-1]);
                              aptabInverse.put(actualParams[paramNo-
1],paramNo);

                              paramNo++;
                        }
                        i=kpdtp-1;
                        for(int j=0;j<kp;j++){
                              String temp[]=kpdt.get(i).split("\t");
                              aptab.put(paramNo,temp[1]);
                              aptabInverse.put(temp[0],paramNo);
                              i++;
                              paramNo++;
                        }
                        i=pp+1;
                        while(i<=actualParams.length){
                              String initializedParams[]=actualParams[i-
1].split("=");

      aptab.put(aptabInverse.get(initializedParams[0].substring(1,initial
izedParams[0].length())),initializedParams[1].substring(0,initializedPara
ms[1].length()));
                              i++;
                        }
                        i=mdtp-1;
                        while(mdt.get(i).compareToIgnoreCase("MEND")!=0){
                              f1.write("+ ");
                              for(int j=0;j<mdt.get(i).length();j++){
                                    if(mdt.get(i).charAt(j)=='#')

      f1.write(aptab.get(Integer.parseInt("" + mdt.get(i).charAt(++j))));
                                    else
                                          f1.write(mdt.get(i).charAt(j));
                              }
                              f1.write("\n");
                              i++;
                        }
                        aptab.clear();
                        aptabInverse.clear();
                  }
                  else
                        f1.write("+ "+s+"\n");
            }
            b1.close();
            b2.close();
            b3.close();
            b4.close();
            f1.close();
      }
}
```

```
/*
OUTPUT:

Intermediate - -
M1 10,20,&b=CREG
M2 100,200,&u=&AREG,&v=&BREG

Kpdt—

a     AREG
b     -
u     CREG
v     DREG

pass2—
+ MOVE AREG,10
+ ADD AREG,='1'
+ MOVER AREG,20
+ ADD AREG,='5'
+ MOVER &AREG,100
+ MOVER &BREG,200
+ ADD &AREG,='15'
+ ADD &BREG,='10'


MNT—
M1    2     2     1     1
M2    2     2     6     3


MDT --
MOVE #3,#1
ADD #3,='1'
MOVER #3,#2
ADD #3,='5'
MEND
MOVER #3,#1
MOVER #4,#2
ADD #3,='15'
ADD #4,='10'
MEND
*/
```

1.FCFS

```
*/
import java.io.*;
import java.util.Scanner;
public class FCFS
{
      public static void main(String args[])
      {
            int i,no_p,burst_time[],TT[],WT[];
            float avg_wait=0,avg_TT=0;
            burst_time=new int[50];
            TT=new int[50];
            WT=new int[50];
            WT[0]=0;
            Scanner s=new Scanner(System.in);
            System.out.println("Enter the number of process: ");
            no_p=s.nextInt();
            System.out.println("\nEnter Burst Time for processes:");
            for(i=0;i<no_p;i++)
            {
                  System.out.print("\tP"+(i+1)+":  ");
                  burst_time[i]=s.nextInt();
            }

            for(i=1;i<no_p;i++)
            {
                  WT[i]=WT[i-1]+burst_time[i-1];
                  avg_wait+=WT[i];
            }
            avg_wait/=no_p;

            for(i=0;i<no_p;i++)
            {
                  TT[i]=WT[i]+burst_time[i];
                  avg_TT+=TT[i];
            }
            avg_TT/=no_p;

      System.out.println("\n*********************************************
******************");
            System.out.println("\tProcesses:");

      System.out.println("***********************************************
****************");
            System.out.println("    Process\tBurst Time\tWaiting
Time\tTurn Around Time");
            for(i=0;i<no_p;i++)
            {
                  System.out.println("\tP"+(i+1)+"\t
"+burst_time[i]+"\t\t  "+WT[i]+"\t\t "+TT[i]);

            }
            System.out.println("\n---------------------------------------
------------------------");
            System.out.println("\nAverage waiting time : "+avg_wait);
            System.out.println("\nAverage Turn Around time :
"+avg_TT+"\n");
      }
}



/*Output:
Enter the number of process:
3

Enter Burst Time for processes:
      P1:  24
      P2:  3
      P3:  3

****************************************************************
      Processes:
****************************************************************
    Process Burst Time  Waiting Time     Turn Around Time
      P1      24          0          24
      P2      3           24         27
      P3      3           27         30

----------------------------------------------------------------
Average waiting time : 17.0
Average Turn Around time : 27.0  */
```

```
/*                2. SJF(Non-Preemptive)           */
import java.util.Scanner;
class SJF1{
public static void main(String args[]){
int burst_time[],process[],waiting_time[],tat[],i,j,n,total=0,pos,temp;
float wait_avg,TAT_avg;
Scanner s = new Scanner(System.in);

System.out.print("Enter number of process: ");
n = s.nextInt();

process = new int[n];
burst_time = new int[n];
waiting_time = new int[n];
tat = new int[n];

System.out.println("\nEnter Burst time:");
for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)+"]: ");
burst_time[i] = s.nextInt();;
process[i]=i+1; //Process Number
}

//Sorting
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(burst_time[j]<burst_time[pos])
pos=j;
}

temp=burst_time[i];
burst_time[i]=burst_time[pos];
burst_time[pos]=temp;

temp=process[i];
process[i]=process[pos];
process[pos]=temp;
}
//First process has 0 waiting time
waiting_time[0]=0;
//calculate waiting time
for(i=1;i<n;i++)
{
waiting_time[i]=0;
for(j=0;j<i;j++)
waiting_time[i]+=burst_time[j];
total+=waiting_time[i];
}

//Calculating Average waiting time
wait_avg=(float)total/n;
total=0;
System.out.println("\nProcess\t Burst Time \tWaiting Time\tTurnaround
Time");
for(i=0;i<n;i++)
{
tat[i]=burst_time[i]+waiting_time[i]; //Calculating Turnaround Time
total+=tat[i];
System.out.println("\n p"+process[i]+"\t\t "+burst_time[i]+"\t\t
"+waiting_time[i]+"\t\t "+tat[i]);
}

//Calculation of Average Turnaround Time
TAT_avg=(float)total/n;
System.out.println("\n\nAverage Waiting Time: "+wait_avg);
System.out.println("\nAverage Turnaround Time: "+TAT_avg);

}
}
```

Output:

```
Enter number of process: 3

Enter Burst time:

Process[1]: 5

Process[2]: 2

Process[3]: 3

Process      Burst Time      Waiting Time     Turnaround Time

 p2          2           0           2

 p3          3           2           5

 p1          5           5           10


Average Waiting Time: 2.3333333

Average Turnaround Time: 5.6666665
```

```
/* 2. SJF(Preemptive)*/
import java.util.Scanner;

class sjf_swap1{
public static void main(String args[])

{
int
burst_time[],process[],waiting_time[],tat[],arr_time[],completion_time[],
i,j,n,total=0,total_comp=0,pos,temp;
float wait_avg,TAT_avg;
Scanner s = new Scanner(System.in);
 System.out.print("Enter number of process: ");
n = s.nextInt();
 process = new int[n];
burst_time = new int[n];
waiting_time = new int[n];
arr_time=new int[n];
tat = new int[n];
completion_time=new int[n];

//burst time
System.out.println("\nEnter Burst time:");
for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)+"]: ");
burst_time[i] = s.nextInt();;
process[i]=i+1; //Process Number
}

//arrival time
System.out.println("\nEnter arrival time:");
for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)+"]: ");
arr_time[i] = s.nextInt();;
process[i]=i+1; //Process Number
}

//Sorting
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(burst_time[j]<burst_time[pos])
pos=j;
}

temp=burst_time[i];
burst_time[i]=burst_time[pos];
burst_time[pos]=temp;

temp=process[i];
process[i]=process[pos];
process[pos]=temp;

System.out.println("process"+process[i]);
}
//completion time new
for(i=1;i<n;i++)
{
completion_time[i]=0;
for(j=0;j<i;j++)
completion_time[i]+=burst_time[j];
 total_comp+=completion_time[i];
}

//First process has 0 waiting time
waiting_time[0]=0;
//calculate waiting time
for(i=1;i<n;i++)
{
waiting_time[i]=0;
for(j=0;j<i;j++)
waiting_time[i]+=burst_time[j];
total+=waiting_time[i];
}


//Calculating Average waiting time
wait_avg=(float)total/n;
total=0;

System.out.println("\nPro_number\t Burst Time \tcompletion_time\tWaiting
Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=burst_time[i]+waiting_time[i];
 //Calculating Turnaround Time
total+=tat[i];
System.out.println("\n"+process[i]+"\t\t "+burst_time[i]+"\t\t
"+completion_time[i]+"\t\t"+waiting_time[i]+"\t\t "+tat[i]);
}

//Calculation of Average Turnaround Time
TAT_avg=(float)total/n;
System.out.println("\n\nAWT: "+wait_avg);
System.out.println("\nATAT: "+TAT_avg);

}
}
```

Output:

```
Enter number of process: 3

Enter Burst time:

Process[1]: 6

Process[2]: 2

Process[3]: 3

Enter arrival time:

Process[1]: 0

Process[2]: 1

Process[3]: 2
process2
process3
process1

Pro_number  Burst Time      completion_time  Waiting Time     Turnaround
Time

2           2           0          0           2

3           3           2          2           5

1           6           5          5           11


AWT: 2.3333333

ATAT: 6.0
```

```
/* Round Robin */
import java.util.Scanner;
public class RR
{
public static void main(String args[])
{
int n,i,qt,count=0,temp,sq=0,bt[],wt[],tat[],rem_bt[];
float awt=0,atat=0;
bt = new int[10];
wt = new int[10];
tat = new int[10];
rem_bt = new int[10];
Scanner s=new Scanner(System.in);
System.out.print("Enter the number of process (maximum 10) = ");
n = s.nextInt();
System.out.print("Enter the burst time of the process\n");
for (i=0;i<n;i++)
{
System.out.print("P"+i+" = ");
bt[i] = s.nextInt();
rem_bt[i] = bt[i];
}
System.out.print("Enter the quantum time: ");
qt = s.nextInt();
while(true)
{
for (i=0,count=0;i<n;i++)
{
temp = qt;
if(rem_bt[i] == 0)
{
count++;
continue;
}
if(rem_bt[i]>qt)
rem_bt[i]= rem_bt[i] - qt;
else
if(rem_bt[i]>=0)
{
temp = rem_bt[i];
rem_bt[i] = 0;
}
sq = sq + temp;
tat[i] = sq;
}
if(n == count)
break;
}
System.out.print("--------------------------------------------------------
------------------------");
System.out.print("\nProcess\t      Burst Time\t       Turnaround Time\t
Waiting Time\n");
System.out.print("--------------------------------------------------------
------------------------");
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
awt=awt+wt[i];
atat=atat+tat[i];
System.out.print("\n "+(i+1)+"\t "+bt[i]+"\t\t "+tat[i]+"\t\t
"+wt[i]+"\n");
}
awt=awt/n;
atat=atat/n;
System.out.println("\nAverage waiting Time = "+awt+"\n");
System.out.println("Average turnaround time = "+atat);
}
}
```

Output:

```
Enter the number of process (maximum 10) = 3
Enter the burst time of the process
P0 = 5
P1 = 6
P2 = 2
Enter the quantum time: 2
-------------------------------------------------------------------------
-------
Process          Burst Time        Turnaround Time          Waiting
Time
-------------------------------------------------------------------------
-------
 1     5          11          6

 2     6          13          7

 3     2          6           4

Average waiting Time = 5.6666665

Average turnaround time = 10.0
```

```
/* Priority */
import java.util.Scanner;
public class priority {
public static void main(String args[]) {
Scanner s = new Scanner(System.in);
int x,n,p[],pp[],bt[],w[],t[],i;
float awt,atat;
p = new int[10];
pp = new int[10];
bt = new int[10];
w = new int[10];
t = new int[10];
//n is number of process
//p is process
//pp is process priority
//bt is process burst time
//w is wait time
// t is turnaround time
//awt is average waiting time
//atat is average turnaround time
System.out.print("Enter the number of process : ");
n = s.nextInt();
System.out.print("\n\t Enter CPU time---priority \n");
for(i=0;i<n;i++)
{
System.out.print("\nProcess["+(i+1)+"]:");
bt[i] = s.nextInt();
pp[i] = s.nextInt();p[i]=i+1;
}
//sorting on the basis of priority
for(i=0;i<n-1;i++)
{
for(int j=i+1;j<n;j++)
{
if(pp[i]<pp[j])
{
x=pp[i];
pp[i]=pp[j];
pp[j]=x;
x=bt[i];
bt[i]=bt[j];
bt[j]=x;
x=p[i];
p[i]=p[j];
p[j]=x;
}
}
}
w[0]=0;
awt=0;
t[0]=bt[0];
atat=t[0];
for(i=1;i<n;i++)
{
w[i]=t[i-1];
awt+=w[i];
t[i]=w[i]+bt[i];
atat+=t[i];
}
//Displaying the process
System.out.println("-----------------------------------------------------
------------------");
System.out.print("\n\nProcess \t\t |Burst Time \t\t |Wait Time \t\t |Turn
Time \n");
System.out.println("-----------------------------------------------------
------------------");
for(i=0;i<n;i++)
System.out.print("\n"+p[i]+"\t\t| "+bt[i]+"\t\t|
"+w[i]+"\t\t|"+t[i]+"\t\t| "+pp[i]+"\n");
System.out.println("-----------------------------------------------------
------------------");
awt/=n;
atat/=n;
System.out.print("\n Average Wait Time : "+awt);
System.out.print("\n Average Turn Around Time : "+atat);
}
}
```

Output:

```
lab-a-26@laba26-Vostro-3669:~/Documents/sp os/spos/c10/priority$ java
priority
Enter the number of process : 5

       Enter CPU time---priority

Process[1]:10 3

Process[2]:1 1

Process[3]:2 3

Process[4]:1 4

Process[5]:5 2
-----------------------------------------------------------------------


Process            |Burst Time            |Wait Time              |Turn
Time
-----------------------------------------------------------------------

4           | 1         | 0         |1          | 4

3           | 2         | 1         |3          | 3

1           | 10        | 3         |13         | 3

5           | 5         | 13        |18         | 2

2           | 1         | 18        |19         | 1
-----------------------------------------------------------------------

 Average Wait Time : 7
 Average Turn Around Time : 10lab-a-26@laba26-Vostro-3669:~/Documents/sp
os/spos/c10/priority$
```

```
/* First Fit */

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

 //Java implementation of First - Fit algorith
//Java implementation of First - Fit algorithm
class firstFit
{
// Method to allocate memory to
// blocks as per First fit algorithm
static void firstFit(int blockSize[], int m, int processSize[], int n)
{
// Stores block id of the
// block allocated to a process
int allocation[] = new int[n];
// Initially no block is assigned to any process
for (int i = 0; i < allocation.length; i++)
allocation[i] = -1;
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i = 0; i <n; i++)
{
for (int j = 0; j < m; j++)
{
if (blockSize[j] >= processSize[i])
{
// allocate block j to p[i] process
allocation[i] =j;
// Reduce available memory in this block.
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();
}
}
// Driver Code
public static void main(String args)
{
int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
int m= blockSize.length;
int n= processSize.length;
firstFit(blockSize, m, processSize, n);
}
}
```

Output:

```
Process No.  Process Size  Block No.
1                   212        2
2                   417        5
3                   112        2
4                   426        Not Allocated
```

```
// Java program for next fit
// memory management algorithm
import java.util.Arrays;

public class nextFit {

// Function to allocate memory to blocks as per Next fit
// algorithm
    static void NextFit(int blockSize[], int m, int processSize[], int n)
{
        // Stores block id of the block allocated to a
        // process
        int allocation[] = new int[n], j = 0;

        // Initially no block is assigned to any process
        Arrays.fill(allocation, -1);

        // pick each process and find suitable blocks
        // according to its size ad assign to it
        for (int i = 0; i < n; i++) {

            // Do not start from beginning
            int count =0;
            while (j < m) {
                count++;    //makes sure that for every process we
traverse through entire array maximum once only.This avoids the problem
of going into infinite loop if memory is not available
                if (blockSize[j] >= processSize[i]) {

                    // allocate block j to p[i] process
                    allocation[i] = j;

                    // Reduce available memory in this block.
                    blockSize[j] -= processSize[i];

                    break;
                }

                // mod m will help in traversing the blocks from
                // starting block after we reach the end.
                j = (j + 1) % m;
            }
        }
        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("");
        }
    }

// Driver program
    static public void main(String[] args) {
        int blockSize[] = {5, 10, 20};
        int processSize[] = {10, 20, 5};
        int m = blockSize.length;
        int n = processSize.length;
        NextFit(blockSize, m, processSize, n);
    }
}

// This code is contributed by Rajput-Ji
```

Output:

```
Process No.    Process Size    Block no.
 1               10              2
 2               20              3
 3               5               1
```

```
// Java implementation of Best - Fit algorithm

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

public class bestFit
{
    // Method to allocate memory to blocks as per Best fit
    // algorithm
    static void bestFit(int blockSize[], int m, int processSize[],
                                                     int n)
    {
        // Stores block id of the block allocated to a
        // process
        int allocation[] = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;

     // pick each process and find suitable blocks
        // according to its size ad assign to it
        for (int i=0; i<n; i++)
        {
            // Find the best fit block for current process
            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 we could find a block for current process
            if (bestIdx != -1)
            {
                // allocate block j to p[i] process
                allocation[i] = bestIdx;

                // Reduce available memory in this block.
                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();
        }
    }

    // Driver Method
    public static void main(String[] args)
    {
         int blockSize[] = {100, 500, 200, 300, 600};
         int processSize[] = {212, 417, 112, 426};
         int m = blockSize.length;
         int n = processSize.length;

         bestFit(blockSize, m, processSize, n);
    }
}
```

Output:

```
Process No.    Process Size    Block no.
 1        212        4
 2        417        2
 3        112        3
 4        426        5
```

// Java implementation of worst - Fit algorithm

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

public class worstFit
{
    // Method to allocate memory to blocks as per worst fit
    // algorithm
    static void worstFit(int blockSize[], int m, int processSize[],
                                                     int n)
    {
        // Stores block id of the block allocated to a
        // process
        int allocation[] = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;

        // pick each process and find suitable blocks
        // according to its size ad assign to it
        for (int i=0; i<n; i++)
        {
            // Find the best fit block for current process
            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 we could find a block for current process
            if (wstIdx != -1)
            {
                // allocate block j to p[i] process
                allocation[i] = wstIdx;

                // Reduce available memory in this block.
                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();
        }
    }

    // Driver Method
    public static void main(String[] args)
    {
         int blockSize[] = {100, 500, 200, 300, 600};
         int processSize[] = {212, 417, 112, 426};
         int m = blockSize.length;
         int n = processSize.length;

         worstFit(blockSize, m, processSize, n);
    }
}
```

Output:

```
Process No.    Process Size    Block no.
   1        212        5
   2        417        2
   3        112        5
   4        426        Not Allocated
```

```
/* Page Replacement FIFO */

import java.io.*;
class fifo
{

        public static void main(String args[]) throws IOException
        {

                int n;
                int f;

                float rat;
                BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
                System.out.println("Enter the number of Frames :");
                f=Integer.parseInt(br.readLine());
                int fifo[]=new int[f];
                System.out.println("Enter the number of Pages :");
                n=Integer.parseInt(br.readLine());
                int inp[]=new int[n];
                System.out.println("Enter Pages:");
                for(int i=0;i<n;i++)
                inp[i]=Integer.parseInt(br.readLine());
                System.out.println("----------------------");
                for(int i=0;i<f;i++)
                        fifo[i]=-1;
                int Hit=0;
                int Fault=0;
                int j=0;
                boolean check;
                for(int i=0;i<n;i++)
                {
                        check=false;


                                for(int k=0;k<f;k++)
                                if(fifo[k]==inp[i])
                                {
                                        check=true;
                                        Hit=Hit+1;
                                }
                                if(check==false)
                                {
                                        fifo[j]=inp[i];
                                        j++;
                                        if(j>=f)
                                        j=0;
                                        Fault=Fault+1;
                                }

                }
                rat = (float)Hit/(float)n;
                System.out.println("HIT:"+Hit+"  FAULT:"+Fault+"   HIT
RATIO:"+rat);
        }
}
```

Output:

```
Enter the number of Frames :
3
Enter the number of Pages :
10
Enter Pages:
0
1
2
3
4
5
6
7
8
9
----------------------
HIT:0  FAULT:10   HIT RATIO:0.0
```

```
/****LRU****/
import java.io.*;
class lru
{
public static void main(String args[])throws IOException
{
BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
int f,page=0,ch,pgf=0,n,chn=0;
boolean flag;
int pages[]; //pgf-page fault
System.out.println("1.LRU");
int pt=0;
System.out.println("enter no. of frames: ");
f=Integer.parseInt(obj.readLine());
int frame[]=new int[f];
for(int i=0;i<f;i++)
{
frame[i]=-1;
}
System.out.println("enter the no of pages ");
n=Integer.parseInt(obj.readLine());
pages=new int[n];
System.out.println("enter the page no ");
for(int j=0;j<n;j++)
pages[j]=Integer.parseInt(obj.readLine());

int pg=0;
for(pg=0;pg<n;pg++)
{
page=pages[pg];
flag=true;
for(int j=0;j<f;j++)
{
if(page==frame[j])

{
flag=false;
break;
}
}
int temp,h=3,i;
if(flag)
{
if( frame[1]!=-1 && frame[2]!=-1 && frame[0]!=-1)
{
temp=pages[pg-3];
if(temp==pages[pg-2] || temp==pages[pg-1])
temp=pages[pg-4];
for(i=0;i<f;i++)
if(temp==frame[i])
break;
frame[i]=pages[pg];
}
else
{
if(frame[0]==-1)
frame[0]=pages[pg];
else if(frame[1]==-1)
frame[1]=pages[pg];
else if(frame[2]==-1)
frame[2]=pages[pg];
}
System.out.print("frame :");
for(int j=0;j<f;j++)
System.out.print(frame[j]+" ");
System.out.println();
pgf++;
}
else
{
System.out.print("frame :");
for(int j=0;j<f;j++)
System.out.print(frame[j]+" ");
System.out.println();
}
}//for
System.out.println("Page fault:"+pgf);

}//main
}//class


/*
OUTPUT:-
enter no. of frames:
4
enter the no of pages
10
enter the page no
1
0
1
2
3
7
8
1
5
2
frame :1 -1 -1 -1
frame :1 0 -1 -1
frame :1 0 -1 -1
frame :1 0 2 -1
frame :1 3 2 -1
frame :7 3 2 -1
frame :7 3 8 -1
frame :7 1 8 -1
frame :5 1 8 -1
frame :5 1 2 -1
Page fault:9
*/
```

```
 /****Optimal****/
import java.util.*;
import java.io.*;
class Optimal
{
public static void main(String args[])throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
int numberOfFrames, numberOfPages, flag1, flag2, flag3, i, j, k, pos = 0,
max;
int faults = 0;
int temp[] = new int[10];
System.out.println("Enter number of Frames: ");
numberOfFrames = Integer.parseInt(br.readLine());
int frame[] = new int[numberOfFrames];
System.out.println("Enter number of Pages: ");
numberOfPages = Integer.parseInt(br.readLine());
int pages[] = new int[numberOfPages];
System.out.println("Enter the pages: ");
for(i=0; i<numberOfPages; i++)
pages[i] = Integer.parseInt(br.readLine());
for(i = 0; i < numberOfFrames; i++)
frame[i] = -1;
for(i = 0; i < numberOfPages; ++i){
flag1 = flag2 = 0;
for(j = 0; j < numberOfFrames; ++j){
if(frame[j] == pages[i]){
flag1 = flag2 = 1;
break;
}
}
if(flag1 == 0){
for(j = 0; j < numberOfFrames; ++j){
if(frame[j] == -1){
faults++;
frame[j] = pages[i];
flag2 = 1;
break;
}
}
}
if(flag2 == 0){
flag3 =0;
for(j = 0; j < numberOfFrames; ++j){
temp[j] = -1;

for(k = i + 1; k < numberOfPages; ++k){
if(frame[j] == pages[k]){
temp[j] = k;
break;
}
}
}

for(j = 0; j < numberOfFrames; ++j){
if(temp[j] == -1){
pos = j;
flag3 = 1;
break;
}
}
if(flag3 ==0){
max = temp[0];
pos = 0;
for(j = 1; j < numberOfFrames; ++j){
if(temp[j] > max){
max = temp[j];
pos = j;
}
}
}
frame[pos] = pages[i];
faults++;
}
// System.out.print();
for(j = 0; j < numberOfFrames; ++j){
System.out.print("\t"+ frame[j]);
}
}
System.out.println("\n\nTotal Page Faults: "+ faults);
}
}
```

Output:-

```
Enter number of Pages:
10
Enter the pages:
1
0
1
2
3
7
8
1
5
2
• 1 -1 -1 -1
• 1 0 -1 -1
• 1 0 -1 -1
• 1 0 2 -1
• 1 0 2 3
• 1 7 2 3

Total Page Faults: 7
```