Practical No. 01

Name: Tantak Samruddhi Sunil

Roll No. 65

Title: Design suitable data structures and implement pass1 and pass2 of a two pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of pass1 (intermediate code file and symbol table) should be input for pass2

```
//program

//main code

import java.io.*;

class Main {

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

        // Use command-line argument if provided, else use default Input.txt

        FileReader FP;

        if (args.length > 0) {

            FP = new FileReader(args[0]);

        } else {

            FP = new FileReader("Input.txt");

        }

        BufferedReader bufferedReader = new BufferedReader(FP);

        String line = null;

        String line2 = null;

        int line_count = 0, LC = 0, LC1 = 0;

        int symTabLine = 0, opTabLine = 0, litTabLine = 0, poolTabLine = 0, MachineTabLine = 0;

        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];

        String Machine[][] = new String[MAX][4];

        System.out.println("_");
        while ((line = bufferedReader.readLine()) != null) {
            String tokens[] = line.trim().split("[\t ]+");
            if (tokens.length == 0) continue;
            if (line_count == 0) {
                LC = Integer.parseInt(tokens[2]);
                LC = LC - 1;
            }
            for (String token : tokens)
                System.out.print(token + "\t");
            System.out.println();
            if (line_count > 0) {
                if (!tokens[0].equalsIgnoreCase("")) {
                    SymbolTab[symTabLine][0] = tokens[0];
                    SymbolTab[symTabLine][1] = Integer.toString(LC);
                    SymbolTab[symTabLine][2] = "1";
                    symTabLine++;
                }
                if (tokens.length > 1 && (tokens[1].equalsIgnoreCase("DS") ||
tokens[1].equalsIgnoreCase("DC"))) {
                    SymbolTab[symTabLine][0] = tokens[0];
                    SymbolTab[symTabLine][1] = Integer.toString(LC);
                    SymbolTab[symTabLine][2] = "1";
                    symTabLine++;
                }
                if (tokens.length > 1 && tokens[1].startsWith("=")) {
                    LitTab[litTabLine][0] = tokens[1];
                    LitTab[litTabLine][1] = Integer.toString(LC);
                    litTabLine++;
                }
                if (tokens.length > 1) {
                    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")) {
                        OpTab[opTabLine][1] = "AD";
                        OpTab[opTabLine][2] = getOpcodeInfo(tokens[1]);
                    } else if (tokens[1].equalsIgnoreCase("DS") ||
tokens[1].equalsIgnoreCase("DC")) {
                        OpTab[opTabLine][1] = "DL";
                        OpTab[opTabLine][2] = "R#7";
                    } else {
                        OpTab[opTabLine][1] = "IS";
                        OpTab[opTabLine][2] = "(" + getOpcodeInfo(tokens[1]) + ",1)";
                    }
                    opTabLine++;
                }
            }
            line_count++;
            LC++;
        }
        bufferedReader.close();
        // 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("------------------");
// Pool Table Initialization
if (litTabLine > 0) PoolTab[poolTabLine++] = 0;
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("------------------");
// Pass 2 - Machine Code Generation
System.out.println("\n\n MACHINE CODE ");
System.out.println("-----------------");
FileReader FP2 = new FileReader("Input.txt");
BufferedReader br2 = new BufferedReader(FP2);
```

```
        line_count = 0;
        LC1 = 0;
        while ((line = br2.readLine()) != null) {
            String tokens1[] = line.trim().split("[\t ]+");
            if (tokens1.length == 0) continue;
            if (line_count == 0) {
                LC1 = Integer.parseInt(tokens1[2]) - 1;
            } else if (tokens1[1].equalsIgnoreCase("END")) {
                break;
            } else {
                Machine[MachineTabLine][0] = Integer.toString(LC1);
                Machine[MachineTabLine][1] = getOpcodeInfo(tokens1[1]);
                Machine[MachineTabLine][2] = (tokens1.length > 2) ? tokens1[2] : "0";
                Machine[MachineTabLine][3] = (tokens1.length > 3) ? tokens1[3] : "0";
                MachineTabLine++;
            }
            LC1++;
            line_count++;
        }
        for (int i = 0; i < MachineTabLine; i++)
            System.out.println(Machine[i][0] + "\t" + Machine[i][1] + "\t" + Machine[i][2] + "\t" + Machine[i][3]);
        br2.close();
    }
    // Reads opcode info from opcode.txt file
    static String getOpcodeInfo(String mnemonic) {
        try {
            FileReader fp = new FileReader("opcode.txt");
            BufferedReader br = new BufferedReader(fp);
            String line;
            while ((line = br.readLine()) != null) {
```

```
            String OPS[] = line.trim().split("[\t ]+");
            if (OPS.length >= 2 && OPS[0].equalsIgnoreCase(mnemonic)) {
                br.close();
                return OPS[1];
            }
        }
        br.close();
    } catch (Exception e) {
        return "00";
    }
    return "00";
  }
}
```

//Input.txt

```
START 100
READ A
READ B
LABLE MOVER AREG, A
ADD AREG, B
LTORG
='5'
='1'
='6'
MOVEM AREG, C
LTORG
='2'
PRINT C
A DS 1
B DS 1
```

```
C DS 1
END
```

//Opcode.txt

```
START R#1
END R#2
ORIGIN R#3
EQU R#4
LTORG R#5
DC R#7
STOP 00
ADD 01
SUB 02
MULT 03
MOVER 04
MOVEM 05
COMP 06
BC 07
DIV 08
READ 09
PRINT 10
JUMP 11
AREG 1
BREG 1
CREG 1
```

Output:

```
START   100
READ    A
READ    B
LABLE   MOVER   AREG,   A
ADD     AREG,   B
LTORG
='5'
='1'
='6'
MOVEM   AREG,   C
LTORG
='2'
PRINT   C
A       DS      1
B       DS      1
C       DS      1
END

 SYMBOL TABLE
--------------------------
SYMBOL  ADDRESS LENGTH
--------------------------
LABLE   103     1
A       111     1
B       112     1
C       113     1
--------------------------
```

```
OPCODE TABLE
----------------------------
MNEMONIC   CLASS   INFO
----------------------------
START      AD      R#1
READ       IS      (09,1)
MOVER      IS      (04,1)
ADD        IS      (01,1)
LTORG      AD      R#5
MOVEM      IS      (05,1)
PRINT      IS      (10,1)
DS         DL      R#7
END        AD      R#2
----------------------------


 LITERAL TABLE
-----------------
LITERAL ADDRESS
-----------------
='5'   105
='1'   106
='6'   107
='2'   110
------------------
POOL TABLE
-----------------
LITERAL NUMBER
-----------------
0
3
```

```
------------------
 MACHINE CODE
-----------------
100   R#1    0    0
101   09     0    111
102   09     0    112
103   04     1    111
104   01     1    112
105   05     1    113
106   10     0    113
107   00     0    000
------------------
```

Practical No. 02

Name: Tantak Samruddhi Sunil

Roll No. 65

Title: Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro-processor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II.

```
//program

//main code

import java.util.*;

import java.io.*;

class twopassmacro {

   static String mnt[][] = new String[5][3];  // Macro Name Table

   static String ala[][] = new String[10][2]; // Argument List Array

   static String mdt[][] = new String[20][1]; // Macro Definition Table

   static int mntc = 0, mdtc = 0, alac = 0;

   public static void main(String args[]) {

      pass1();

      System.out.println("Macro Name Table (MNT)");

      display(mnt, mntc, 3);

      System.out.println("\nArgument List Array (ALA) after Pass 1");

      display(ala, alac, 2);

      System.out.println("\nMacro Definition Table (MDT)");

      display(mdt, mdtc, 1);

      pass2();

      System.out.println("\nArgument List Array (ALA) after Pass 2");

      display(ala, alac, 2);

      System.out.println("\nNote: All tables are displayed here whereas intermediate Pass 1
output & expanded Pass 2 output are stored in files.");

   }

   // ------------------------------------------------------------

   // PASS 1: Build MNT, ALA, MDT
// ------------------------------------------------------------
static void pass1() {
    int index = 0, i;
    String s, prev = "", substring;
    try {
        BufferedReader inp = new BufferedReader(new FileReader("input1.txt"));
        BufferedWriter output = new BufferedWriter(new FileWriter("pass1_output.txt"));
        while ((s = inp.readLine()) != null) {
            s = s.trim();
            if (s.equalsIgnoreCase("MACRO")) {
                prev = s;
                s = inp.readLine();
                while (s != null && !s.equalsIgnoreCase("MEND")) {
                    if (prev.equalsIgnoreCase("MACRO")) {
                        // First line after MACRO -> name and args
                        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 index
                        mnt[mntc][1] = str[0];          // Macro name
                        mnt[mntc++][2] = (++mdtc) + "";  // MDT pointer
                        // Parse arguments
                        if (str.length > 1) {
                            st = new StringTokenizer(str[1], ",");
                            while (st.hasMoreTokens()) {
                                String arg = st.nextToken();
                                index = arg.indexOf('=');
                                if (index != -1)
                                    ala[alac][1] = arg.substring(0, index);
                        else
                            ala[alac][1] = arg;
                        ala[alac][0] = String.valueOf(alac);
                        alac++;
                    }
                }
            } else {
                // Macro definition line
                for (i = 0; i < alac; i++) {
                    substring = "&" + ala[i][1];
                    if (s.contains(substring)) {
                        s = s.replace(substring, "#" + i);
                    }
                }
            }
            mdt[mdtc++][0] = s;
            prev = s;
            s = inp.readLine();
        }
        // Add MEND to MDT
        if (s != null) {
            mdt[mdtc++][0] = "MEND";
        }
    } else {
        output.write(s);
        output.newLine();
    }
}
output.close();
inp.close();
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
}
// ------------------------------------------------------------
// PASS 2: Macro Expansion
// ------------------------------------------------------------
static void pass2() {
    int alap = 0, mdtp = 0, flag = 0, i, j;
    String s;
    try {
        BufferedReader inp = new BufferedReader(new FileReader("pass1_output.txt"));
        BufferedWriter output = new BufferedWriter(new FileWriter("pass2_output.txt"));
        while ((s = inp.readLine()) != null) {
            s = s.trim();
            flag = 0;
            StringTokenizer st = new StringTokenizer(s);
            String str[] = new String[st.countTokens()];
            for (i = 0; i < str.length; i++)
                str[i] = st.nextToken();
            if (str.length == 0)
                continue;
            for (j = 0; j < mntc; j++) {
                if (str[0].equalsIgnoreCase(mnt[j][1])) {
                    mdtp = Integer.parseInt(mnt[j][2]);
                    alap = 0;
                    if (str.length > 1) {
                        st = new StringTokenizer(str[1], ",");
                        while (st.hasMoreTokens()) {
                            ala[alap][1] = st.nextToken();
                            alap++;
                        }
                    }
                    // Expand macro until MEND
                    for (i = mdtp; i < mdtc && !mdt[i][0].equalsIgnoreCase("MEND"); i++) {
                        String temp = mdt[i][0];
                        if (temp.contains("#")) {
                            int pos = temp.indexOf("#");
                            int num = Integer.parseInt("" + temp.charAt(pos + 1));
                            temp = temp.substring(0, pos) + ala[num][1];
                        }
                        output.write(temp);
                        output.newLine();
                    }
                    flag = 1;
                    break;
                }
            }
            if (flag == 0) {
                output.write(s);
                output.newLine();
            }
        }
        output.close();
        inp.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
    // ------------------------------------------------------------
    // DISPLAY UTILITY
    // ------------------------------------------------------------
    static void display(String a[][], int n, int m) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                System.out.print((a[i][j] == null ? "-" : a[i][j]) + "\t");
            }
            System.out.println();
        }
    }
}
```

//Input1.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,DATA2
INCR2 DATA3,DATA4
FOUR DC F'4'
FIVE DC F'5'
```

```
BASE EQU 8

TEMP DS 1F

DROP 8

END
```

Output:

```
Macro Name Table (MNT)

1          INCR1 1

2          INCR2 4


Argument List Array (ALA) after Pass 1

0          &FIRST

1          &SECOND

2          &ARG1

3          &ARG2


Macro Definition Table (MDT)

A 1,#0

L 2,#1

MEND

L 3,#2

ST 4,#3

MEND


Argument List Array (ALA) after Pass 2

0          DATA1

1          DATA2

2          DATA3

3          DATA4
```

Practical No. 03

Name: Tantak Samruddhi Sunil

Roll No. 65

Title: Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).

1. FCFS (First Come First Serve)

```
//program
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;
public class FCFSMain {
   // Helper class to store process information
   static class Process {
      private String id;
      private int arrivalTime;
      private int burstTime;
      private int completionTime;
      private int turnAroundTime;
      private int waitingTime;
      public Process(String id, int arrivalTime, int burstTime) {
         this.id = id;
         this.arrivalTime = arrivalTime;
         this.burstTime = burstTime;
      }
      public String getId() { return id; }
      public int getArrivalTime() { return arrivalTime; }
      public int getBurstTime() { return burstTime; }
      public int getCompletionTime() { return completionTime; }
```

```
        public int getTurnAroundTime() { return turnAroundTime; }
        public int getWaitingTime() { return waitingTime; }
        public void setCompletionTime(int completionTime) { this.completionTime = completionTime; }
        public void setTurnAroundTime(int turnAroundTime) { this.turnAroundTime = turnAroundTime; }
        public void setWaitingTime(int waitingTime) { this.waitingTime = waitingTime; }
    }
    // main method — JVM will find this when you run `java FCFSMain`
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of processes: ");
        int numProcesses = scanner.nextInt();
        List<Process> processes = new ArrayList<>();
        for (int i = 0; i < numProcesses; i++) {
            System.out.println("\nEnter details for Process " + (i + 1) + ":");
            System.out.print("Enter Process ID (e.g., P1): ");
            String id = scanner.next();
            System.out.print("Enter Arrival Time: ");
            int arrivalTime = scanner.nextInt();
            System.out.print("Enter Burst Time: ");
            int burstTime = scanner.nextInt();
            processes.add(new Process(id, arrivalTime, burstTime));
        }
        // Sort processes by arrival time for FCFS
        Collections.sort(processes, Comparator.comparingInt(Process::getArrivalTime));
        int currentTime = 0;
        float totalWaitingTime = 0;
        float totalTurnAroundTime = 0;
        System.out.println("\n--- FCFS Scheduling Simulation ---");
        System.out.println("Process\tArrival\tBurst\tCompletion\tTurnaround\tWaiting");
        for (Process process : processes) {
            // If CPU is idle until next arrival
            if (currentTime < process.getArrivalTime()) {
                currentTime = process.getArrivalTime();
            }
            // Completion time
            process.setCompletionTime(currentTime + process.getBurstTime());
            currentTime = process.getCompletionTime();
            // Turnaround & waiting time
            process.setTurnAroundTime(process.getCompletionTime() -
process.getArrivalTime());
            process.setWaitingTime(process.getTurnAroundTime() - process.getBurstTime());
            totalWaitingTime += process.getWaitingTime();
            totalTurnAroundTime += process.getTurnAroundTime();
            System.out.printf("%s\t%d\t%d\t%d\t\t%d\t\t%d\n",
                    process.getId(), process.getArrivalTime(), process.getBurstTime(),
                    process.getCompletionTime(), process.getTurnAroundTime(),
process.getWaitingTime());
        }
        System.out.printf("\nAverage Waiting Time: %.2f\n", (totalWaitingTime /
numProcesses));
        System.out.printf("Average Turnaround Time: %.2f\n", (totalTurnAroundTime /
numProcesses));
        scanner.close();
    }
```

Output:

```
Enter the number of processes: 4

Enter details for Process 1:

Enter Process ID (e.g., P1): 1

Enter Arrival Time: 10

Enter Burst Time: 2

Enter details for Process 2:

Enter Process ID (e.g., P1): 2

Enter Arrival Time: 10

Enter Burst Time: 2

Enter details for Process 3:

Enter Process ID (e.g., P1): 3

Enter Arrival Time: 11

Enter Burst Time: 1

Enter details for Process 4:

Enter Process ID (e.g., P1): 4

Enter Arrival Time: 13

Enter Burst Time: 3


--- FCFS Scheduling Simulation ---

Process         Arrival Burst   Completion      Turnaround      Waiting

1       10      2       12              2               0

2       10      2       14              4               2

3       11      1       15              4               3

4       13      3       18              5               2

Average Waiting Time: 1.75

Average Turnaround Time: 3.75

=== Code Execution Successful ===
```

2. SJF (Shortest Job First)

```
//program
import java.util.*;
// Non-preemptive SJF (Shortest Job First) Scheduling Algorithm
public class SJFScheduling {
    // Helper class to store process details
    static class Process {
        int id;
        int arrivalTime;
        int burstTime;
        int completionTime;
        int turnAroundTime;
        int waitingTime;
        public Process(int id, int arrivalTime, int burstTime) {
            this.id = id;
            this.arrivalTime = arrivalTime;
            this.burstTime = burstTime;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();
        List<Process> processes = new ArrayList<>();
        // Input processes
        for (int i = 0; i < n; i++) {
            System.out.println("\nEnter details for Process " + (i + 1) + ":");
            System.out.print("Arrival Time: ");
            int arrivalTime = sc.nextInt();
            System.out.print("Burst Time: ");
            int burstTime = sc.nextInt();
            processes.add(new Process(i + 1, arrivalTime, burstTime));
        }
        // Sort processes by arrival time
        processes.sort(Comparator.comparingInt(p -> p.arrivalTime));
        int currentTime = 0;
        int completed = 0;
        List<Process> completedList = new ArrayList<>();
        while (completed < n) {
            Process shortest = null;
            int minBurst = Integer.MAX_VALUE;
            // Find the process with the shortest burst time among arrived ones
            for (Process p : processes) {
                if (p.arrivalTime <= currentTime && p.completionTime == 0) {
                    if (p.burstTime < minBurst) {
                        minBurst = p.burstTime;
                        shortest = p;
                    }
                }
            }
            if (shortest == null) {
                currentTime++; // CPU is idle
            } else {
                currentTime += shortest.burstTime;
                shortest.completionTime = currentTime;
                shortest.turnAroundTime = shortest.completionTime - shortest.arrivalTime;
                shortest.waitingTime = shortest.turnAroundTime - shortest.burstTime;
                completedList.add(shortest);
                completed++;
```

```
            }
        }
        // Display output
        double totalWT = 0, totalTAT = 0;
        System.out.println("\nProcess\tArrival\tBurst\tCompletion\tTurnaround\tWaiting");
        for (Process p : completedList) {
            System.out.printf("P%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t\t%d\n",
                    p.id, p.arrivalTime, p.burstTime, p.completionTime, p.turnAroundTime,
p.waitingTime);
            totalWT += p.waitingTime;
            totalTAT += p.turnAroundTime;
        }
        System.out.printf("\nAverage Waiting Time: %.2f\n", totalWT / n);
        System.out.printf("Average Turnaround Time: %.2f\n", totalTAT / n);
        sc.close();
    }
}
```

Output:

```
Enter the number of processes: 5

Enter details for Process 1:

Arrival Time: 0

Burst Time: 3

Enter details for Process 2:

Arrival Time: 1

Burst Time: 5

Enter details for Process 3:

Arrival Time: 3

Burst Time: 2

Enter details for Process 4:

Arrival Time: 9

Burst Time: 5

Enter details for Process 5:

Arrival Time: 12

Burst Time: 5


Process     ArrivalBurst  Completion   Turnaround   Waiting

P1          0             3            3                 3                  0

P3          3             2            5                 2                  0

P2          1             5            10                9                  4

P4          9             5            15                6                  1

P5          12            5            20                8                  3

Average Waiting Time: 1.60

Average Turnaround Time: 5.60


=== Code Execution Successful ===
```

3. RR (Round Robin)

```
//program
import java.util.*;
// Public class name must match the filename: RRScheduler.java
public class RRScheduler {
    // Process as a static inner class to avoid confusion with running the wrong class
    static class Process {
        int processId;
        int arrivalTime;
        int burstTime;
        int remainingTime;
        int waitingTime;
        int turnaroundTime;
        public Process(int processId, int arrivalTime, int burstTime) {
            this.processId = processId;
            this.arrivalTime = arrivalTime;
            this.burstTime = burstTime;
            this.remainingTime = burstTime;
            this.waitingTime = 0;
            this.turnaroundTime = 0;
        }
    }
    // Main method JVM will find when you run `java RRScheduler`
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();
        List<Process> processes = new ArrayList<>();
        for (int i = 0; i < n; i++) {
```

```
    System.out.println("\nEnter details for Process " + (i + 1) + ":");
    System.out.print("Arrival Time: ");
    int at = sc.nextInt();
    System.out.print("Burst Time: ");
    int bt = sc.nextInt();
    processes.add(new Process(i + 1, at, bt));
}
System.out.print("\nEnter Time Quantum: ");
int timeQuantum = sc.nextInt();
// Sort by arrival time
processes.sort(Comparator.comparingInt(p -> p.arrivalTime));
Queue<Process> queue = new LinkedList<>();
int time = 0;
int completed = 0;
boolean[] added = new boolean[n]; // track which processes have been enqueued
// If earliest arrival > 0, advance time to it
if (!processes.isEmpty() && processes.get(0).arrivalTime > 0) {
    time = processes.get(0).arrivalTime;
}
while (completed < n) {
    // Add all processes that have arrived by 'time' and are not yet added
    for (int i = 0; i < n; i++) {
        Process p = processes.get(i);
        if (!added[i] && p.arrivalTime <= time && p.remainingTime > 0) {
            queue.add(p);
            added[i] = true;
        }
    }
    if (queue.isEmpty()) {
        // No process is ready; jump to next arrival time
    int nextArrival = Integer.MAX_VALUE;
    for (Process p : processes) {
        if (p.remainingTime > 0 && p.arrivalTime < nextArrival) {
            nextArrival = p.arrivalTime;
        }
    }
    // If found, fast-forward time; otherwise break (shouldn't happen)
    time = (nextArrival == Integer.MAX_VALUE) ? time + 1 : nextArrival;
    continue;
}
Process cur = queue.poll();
int exec = Math.min(cur.remainingTime, timeQuantum);
cur.remainingTime -= exec;
time += exec;
// Enqueue any newly arrived processes during execution
for (int i = 0; i < n; i++) {
    Process p = processes.get(i);
    if (!added[i] && p.arrivalTime <= time && p.remainingTime > 0) {
        queue.add(p);
        added[i] = true;
    }
}
if (cur.remainingTime == 0) {
    completed++;
    cur.turnaroundTime = time - cur.arrivalTime;
    cur.waitingTime = cur.turnaroundTime - cur.burstTime;
} else {
    // Not finished — re-add to queue
    queue.add(cur);
}
```

```
        }
        // Print results
        double totalWT = 0, totalTAT = 0;
        System.out.println("\nProcess\tArrival\tBurst\tWaiting\tTurnaround");
        for (Process p : processes) {
            System.out.printf("P%d\t\t%d\t%d\t%d\t%d\n",
                    p.processId, p.arrivalTime, p.burstTime, p.waitingTime, p.turnaroundTime);
            totalWT += p.waitingTime;
            totalTAT += p.turnaroundTime;
        }
        System.out.printf("\nAverage Waiting Time: %.2f\n", totalWT / n);
        System.out.printf("Average Turnaround Time: %.2f\n", totalTAT / n);
        sc.close();
    }
}
```

Output:

```
Enter the number of processes: 3
Enter details for Process 1:
Arrival Time: 0
Burst Time: 5
Enter details for Process 2:
Arrival Time: 1
Burst Time: 3
Enter details for Process 3:
Arrival Time: 2
Burst Time: 1
Enter the time quantum: 2

--- Round Robin Scheduling Simulation ---
Process Arrival Burst Waiting Turnaround
P1      0       5     6       11
P2      1       3     4       7
P3      2       1     1       2

Average Waiting Time: 3.67
Average Turnaround Time: 6.67
```

Practical No. 04

Name: Tantak Samruddhi Sunil

Roll No. 65

Title: Write a Java Program to implement paging simulation using 1. FIFO 2. Least Recently Used (LRU) 3. Optimal algorithm

1.FIFO (First In First Out)

```
//program
import java.util.*;
public class FIFOPageReplacement{
    // initialsie seq and frameSize only, run main
    static int[] seq = {1, 2, 3, 4, 2, 3, 4, 1, 2, 1, 1, 3, 1, 4};
    static int[] frame;
    static int frameSize = 3;
    static int pageFault = 0;
    static String pattern = "";
    public static void main(String[] args){
        System.out.println("FIFO : ");
        System.out.println(Integer.MIN_VALUE+" indicates empty memory array !");
        frame = new int[frameSize];
        for(int i = 0; i < frameSize; i++){
            frame[i] = Integer.MIN_VALUE;
        }
        int index = 0;
        int findex = 0;
        System.out.println("Initial Frame : "+Arrays.toString(frame));
        while(index < seq.length){
            int page = seq[index];
            if(!inFrame(page)){
                pageFault++;
                frame[findex] = page;
                pattern += "Y";
                findex = (findex+1)%frameSize;
                System.out.println("Frame : "+Arrays.toString(frame));
            }
            else{
                pattern += "N";
            }
            index++;
        }
        System.out.println(System.lineSeparator()+"Final Frame : "+Arrays.toString(frame));
        System.out.println("no of page faults = "+pageFault);
        int hit_count = seq.length - pageFault;
        System.out.println("no of page hits:"+ hit_count);
    }
    public static boolean inFrame(int page){
        for(int each : frame){
            if(each == page){return true;}
        }
        return false;
    }
}
```

Output:

```
FIFO :
-2147483648 indicates empty memory array !
Initial Frame : [-2147483648, -2147483648, -2147483648]
Frame : [1, -2147483648, -2147483648]
Frame : [1, 2, -2147483648]
Frame : [1, 2, 3]
Frame : [4, 2, 3]
Frame : [4, 1, 3]
Frame : [4, 1, 2]
Frame : [3, 1, 2]
Frame : [3, 4, 2]

Final Frame : [3, 4, 2]
no of page faults = 8
no of page hits:6

=== Code Execution Successful ===
```

2. LRU (Least Recently Used)

```
//Program
package PageReplacement;
import java.util.*;
public class LRUPageReplacement{
    public static void main(String[] args) {
        Scanner myinp = new Scanner(System.in);
        int references, frames, hit=0, fault_count=0, page_get, cap=0, repeat=0;
        System.out.print("Number of Frames: ");
        frames = myinp.nextInt();
        System.out.print("Number of Reference: ");
        references = myinp.nextInt();
        int reference_string[] = new int [references];
        int page[] = new int[references];
        int lru[] = new int[references];
        for (int i = 0; i < references; i++) {
            System.out.print("Reference String "+i+": ");
            reference_string[i] = myinp.nextInt();
        }
        System.out.print("\n==================================");
        for (int i = 0; i < frames; i++) {
            page[i] = 9999;
        }
        for (int i = 0; i < references; i++) {
            System.out.println();
            hit = 0;
            for (int j = 0; j < frames; j++) {
                if (page[j] == reference_string[i]) {
                    hit = 1;
                    break;
                }
            }
            if (hit == 0) {
                for (int j = 0; j < frames; j++) {
                    page_get = page[j];
                    for (int j2 = i-1; j2 >= 0 ; j2--) {
                        if (page_get == reference_string[j2]) {
                            lru[j] = j2;
                            cap = 1;
                            break;
                        } else {
                            cap = 0;
                        }
                    }
                    if (cap == 0) {
                        lru[j] = -9999;
                    }
                }
                int minimum = 9999;
                for (int j = 0; j < frames; j++) {
                    if (lru[j] < minimum) {
                        minimum = lru[j];
                        repeat = j;
                    }
                }
                page[repeat] = reference_string[i];
                fault_count++;
                for (int j = 0; j < frames; j++) {
                    if (page[j] != 9999) {
```

```
                        System.out.print(page[j]+"\t");
                    }
                }
            } else {
                for (int j = 0; j < frames; j++) {
                    System.out.print("-\t");
                }
            }
        }
        System.out.print("\n============================");
        System.out.println("\nFault: "+fault_count);
        int hit_count = references - fault_count;
        System.out.println("\nHit: "+hit_count);
    }
}
```

Output:

```
Number of Frames: 3

Number of Reference: 8

Reference String 0: 3

Reference String 1: 0

Reference String 2: 2

Reference String 3: 1

Reference String 4: 0

Reference String 5: 4

Reference String 6: 1

Reference String 7: 0


===============================

3

3       0

3       0       2

1       0       2

-       -       -

1       0       4

-       -       -

-       -       -
===============================

Fault: 5


Hit: 3


=== Code Execution Successful ===
```

3. OPRA (Optimal Page Replacement Algorithm)

```
//Program
package PageReplacement;
import java.util.*;
public class OPTPageReplacement{
   public static void main(String[] args) {
      Scanner myinp = new Scanner(System.in);
      int references, frames, hit=0, fault_count=0, page_get, cap=0, repeat=0;
      System.out.print("Number of Frames: ");
      frames = myinp.nextInt();
      System.out.print("Number of Reference: ");
      references = myinp.nextInt();
      int reference_string[] = new int [references];
      int page[] = new int[references];
      int opt[] = new int[references];
      for (int i = 0; i < references; i++) {
         System.out.print("Reference String "+i+": ");
         reference_string[i] = myinp.nextInt();
      }
      System.out.print("\n=================================");
      for (int i = 0; i < frames; i++) {
         page[i] = 9999;
      }
      for (int i = 0; i < references; i++) {
         System.out.println();
         hit = 0;
         for (int j = 0; j < frames; j++) {
            if (page[j] == reference_string[i]) {
               hit = 1;
```

```
        break;
    }
}
if (hit == 0) {
    for (int j = 0; j < frames; j++) {
        page_get = page[j];
        for (int j2 = i; j2 < references; j2++) {
            if (page_get == reference_string[j2]) {
                opt[j] = j2;
                cap = 1;
                break;
            } else {
                cap = 0;
            }
        }
        if (cap == 0) {
            opt[j] = 9999;
        }
    }
    int maximum = -9999;
    for (int j = 0; j < frames; j++) {
        if (opt[j] > maximum) {
            maximum = opt[j];
            repeat = j;
        }
    }
    page[repeat] = reference_string[i];
    fault_count++;
    for (int j = 0; j < frames; j++) {
        if (page[j] != 9999) {
```

```
                        System.out.print(page[j]+"\t");
                    }
                }
            } else {
                for (int j = 0; j < frames; j++) {
                    System.out.print("-\t");
                }
            }
        }
        System.out.print("\n===================================");
        System.out.println("\nFault: "+fault_count);
        int hit_count = references - fault_count;
        System.out.println("\nHit: "+hit_count);
    }
}
```

Output:

```
Number of Frames: 4
Number of Reference: 8
Reference String 0: 4
Reference String 1: 1
Reference String 2: 0
Reference String 3: 5
Reference String 4: 0
Reference String 5: 1
Reference String 6: 2
Reference String 7: 4

================================
4
4       1
4       1       0
4       1       0       5
-       -       -       -
-       -       -       -
4       2       0       5
-       -       -       -
================================
Fault: 5

Hit: 3

=== Code Execution Successful ===
```