PRACTICAL 1

Problem Statement –

Design suitable Data structures and implement Pass-I and Pass-II of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives.

SymTab.java –

```
import java.io.*;
class SymTab
{
	public static void main(String args[])throws Exception
	{
		FileReader FP=new FileReader(args[0]);
		BufferedReader bufferedReader = new BufferedReader(FP);

		String line=null;
		int line_count=0,LC=0,symTabLine=0,opTabLine=0,litTabLine=0,poolTabLine=0;

		 //Data Structures
		 final int MAX=100;
		 String SymbolTab[][]=new String[MAX][3];
		 String OpTab[][]=new String[MAX][3];
		 String LitTab[][]=new String[MAX][2];
		 int PoolTab[]=new int[MAX];
		 int litTabAddress=0;
/*-------------------------------------------------------------------------------------------*/

		 System.out.println("_____");
		   while((line = bufferedReader.readLine()) != null)
		    {
			 String[] tokens = line.split("\t");
			if(line_count==0)
			{
				LC=Integer.parseInt(tokens[2]);
				//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].equals
IgnoreCase("ORIGIN")||tokens[1].equalsIgnoreCase("EQU")||tokens[1].equalsIgnoreCase("LTORG"))
        //if Assembler Directive
                                        {
                                            OpTab[opTabLine][1]="AD";
                                            OpTab[opTabLine][2]="R11";

                                        }
```

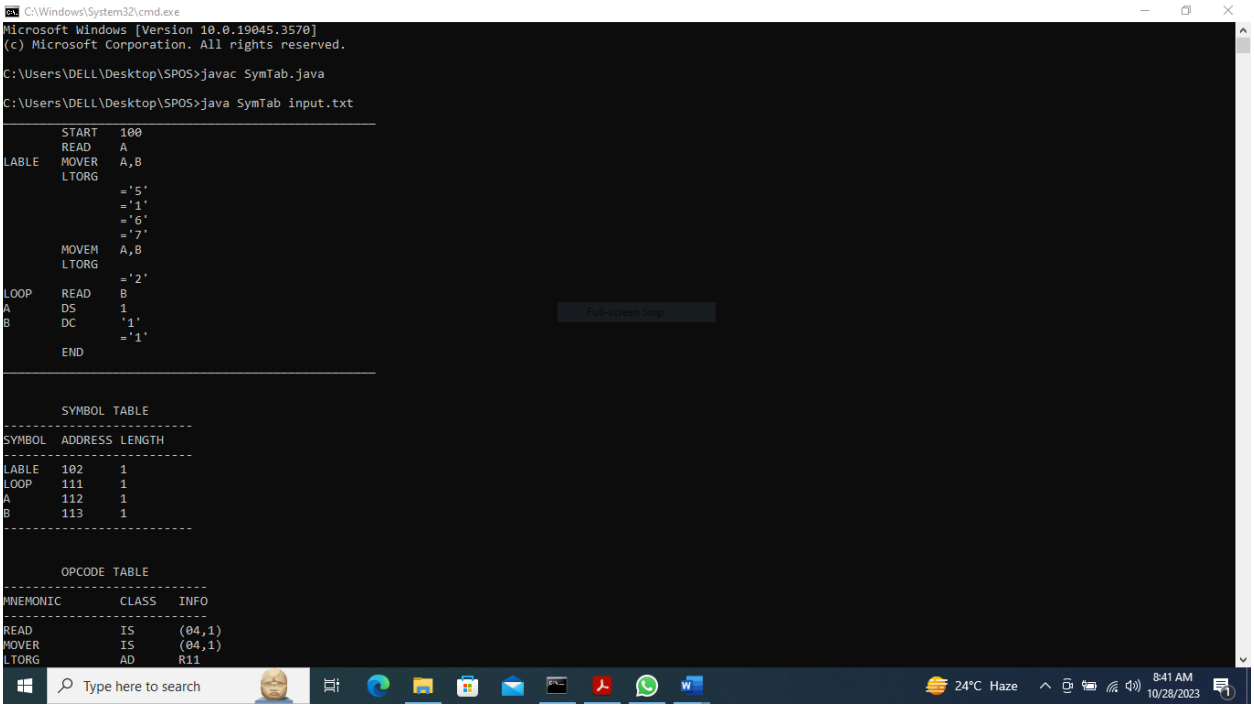
```
                                        else
if(tokens[1].equalsIgnoreCase("DS")||tokens[1].equalsIgnoreCase("DC"))
                                        {
                                                OpTab[opTabLine][1]="DL";
                                                OpTab[opTabLine][2]="R7";

                                        }
                                        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                ");
```

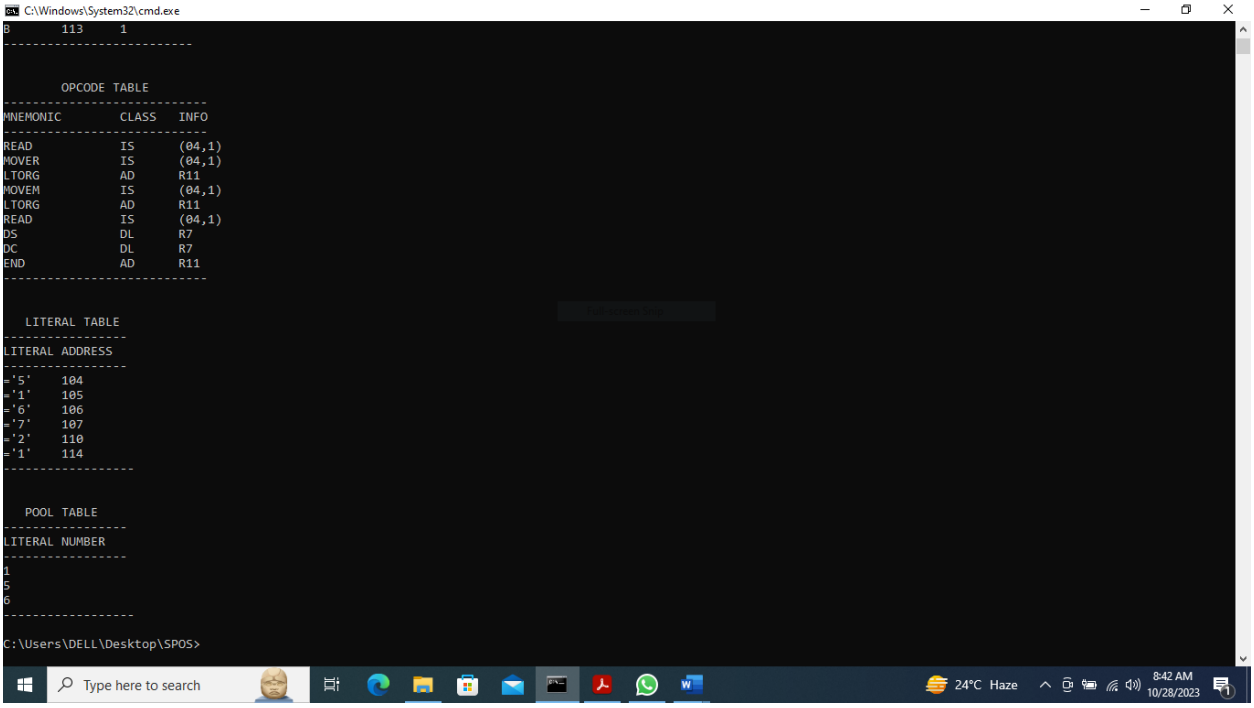
```
                        System.out.println("-----------------");
                        System.out.println("LITERAL\tADDRESS");
                        System.out.println("-----------------");
                        for(int i=0;i<litTabLine;i++)
                                System.out.println(LitTab[i][0]+"\t"+LitTab[i][1]);
                        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();
        }
}
```

input.txt –

```
        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
```

Output –


C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.
C:\Users\DELL\Desktop\SPOS>javac SymTab.java
C:\Users\DELL\Desktop\SPOS>java SymTab input.txt
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
Type here to search
24°C Haze
8:41 AM
10/28/2023


C:\Windows\System32\cmd.exe
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
C:\Users\DELL\Desktop\SPOS>
Type here to search
24°C Haze
8:42 AM
10/28/2023

PRACTICAL 2

Problem Statement –

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.

macroPass1.java –

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

public class macroPass1 {
	public static void main(String[] Args) throws IOException{
		BufferedReader b1 = new BufferedReader(new FileReader("input.txt"));
		FileWriter f1 = new FileWriter("intermediate.txt");
		FileWriter f2 = new FileWriter("mnt.txt");
		FileWriter f3 = new FileWriter("mdt.txt");
		FileWriter f4 = new FileWriter("kpdt.txt");
		HashMap<String,Integer> pntab=new HashMap<String,Integer>();
		String s;
		int paramNo=1,mdtp=1,flag=0,pp=0,kp=0,kpdtp=0;
		while((s=b1.readLine())!=null){
			String word[]=s.split("\\s");		//separate by space
			if(word[0].compareToIgnoreCase("MACRO")==0){
				flag=1;
				if(word.length<=2){

	f2.write(word[1]+"\t"+pp+"\t"+kp+"\t"+mdtp+"\t"+(kp==0?kpdtp:(kpdtp+1))+"\n");
					continue;
				}
				String params[]=word[2].split(",");
				for(int i=0;i<params.length;i++){
					if(params[i].contains("=")){
						kp++;
						String keywordParam[]=params[i].split("=");

	pntab.put(keywordParam[0].substring(1,keywordParam[0].length()),paramNo++);
						if(keywordParam.length==2)

	f4.write(keywordParam[0].substring(1,keywordParam[0].length())+"\t"+keywordParam[1]+"\n");
						else

	f4.write(keywordParam[0].substring(1,keywordParam[0].length())+"\t"+"-"+"\n");
```

```
                                        }
                                        else{

pntab.put(params[i].substring(1,params[i].length()),paramNo++);
                                                pp++;
                                        }
                                }

f2.write(word[1]+"\t"+pp+"\t"+kp+"\t"+mdtp+"\t"+(kp==0?kpdtp:(kpdtp+1))+"\n");
                                kpdtp+=kp;
                        }
                        else if(word[0].compareToIgnoreCase("MEND")==0){
                                f3.write(s+'\n');
                                flag=pp=kp=0;
                                mdtp++;
                                paramNo=1;
                                pntab.clear();
                        }
                        else if(flag==1){
                                for(int i=0;i<s.length();i++){
                                        if(s.charAt(i)=='&'){
                                                i++;
                                                String temp="";
                                                while(!(s.charAt(i)==' '||s.charAt(i)==',')){
                                                        temp+=s.charAt(i++);
                                                        if(i==s.length())
                                                                break;

                                                }
                                                i--;
                                                f3.write("#"+pntab.get(temp));
                                        }
                                        else
                                                f3.write(s.charAt(i));
                                }
                                f3.write("\n");
                                mdtp++;
                        }
                        else{
                                f1.write(s+'\n');
                        }
                }
                b1.close();
                f1.close();
                f2.close();
                f3.close();
```

```
                f4.close();
        }
}
```

macroPass2.java -

```
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));
```

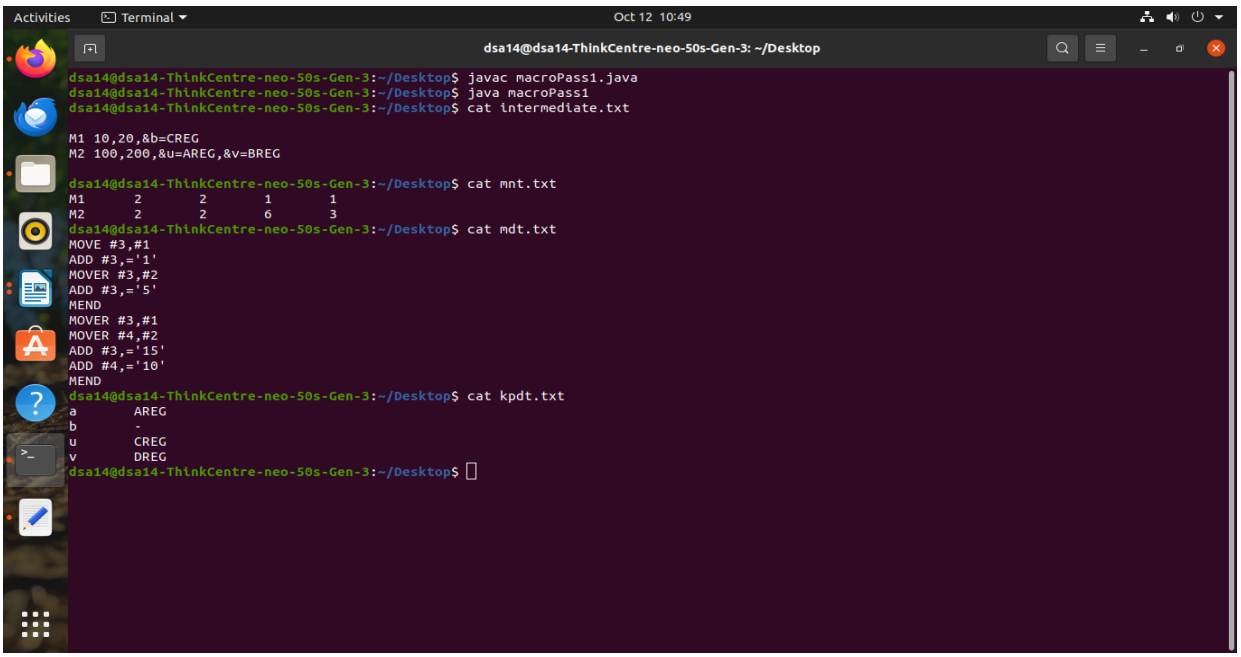
```
                                kpdtp=kpdtpHash.get(b1Split[0]+Integer.toString(pp));
                                String actualParams[]=b1Split[1].split(",");
                                paramNo=1;
                                for(int j=0;j<pp;j++){
                                        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,initializedParams[0].length())),initiali
zedParams[1].substring(0,initializedParams[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();
```

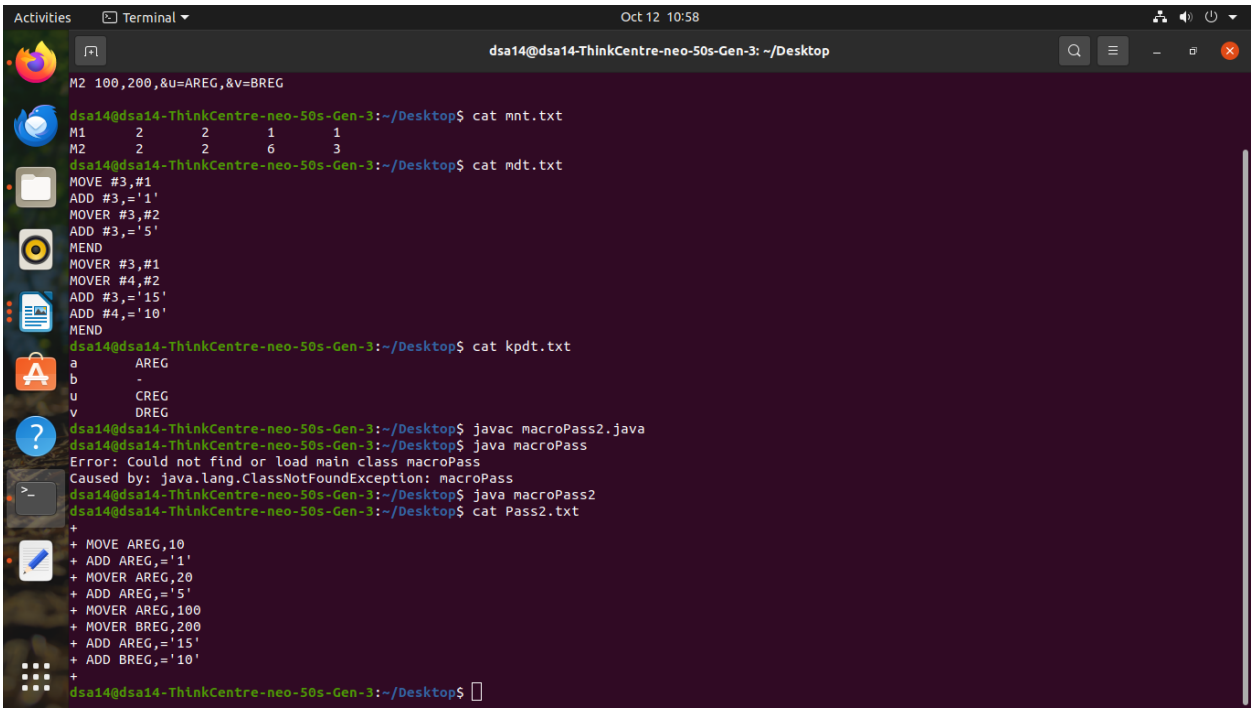
```
                f1.close();
        }
}
```

Input file –

```
MACRO M1 &x,&y,&a=AREG,&b=
MOVE &a,&x
ADD &a,='1'
MOVER &a,&y
ADD &a,='5'
MEND
MACRO M2 &p,&q,&u=CREG,&v=DREG
MOVER &u,&p
MOVER &v,&q
ADD &u,='15'
ADD &v,='10'
MEND
M1 10,20,&b=CREG
M2 100,200,&u=AREG,&v=BREG
```

Output –




Activities
Terminal
Oct 12 10:58
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3: ~/Desktop
M2 100,200,&u=AREG,&v=BREG
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$ cat mnt.txt
M1 2 2 1 1
M2 2 2 6 3
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$ cat mdt.txt
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
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$ cat kpdt.txt
a AREG
b -
u CREG
v DREG
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$ javac macroPass2.java
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$ java macroPass
Error: Could not find or load main class macroPass
Caused by: java.lang.ClassNotFoundException: macroPass
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$ java macroPass2
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$ cat Pass2.txt
+
+ MOVE AREG,10
+ ADD AREG,='1'
+ MOVER AREG,20
+ ADD AREG,='5'
+ MOVER AREG,100
+ MOVER BREG,200
+ ADD AREG,='15'
+ ADD BREG,='10'
+
dsa14@dsa14-ThinkCentre-neo-50s-Gen-3:~/Desktop$

PRACTICAL 3

Problem Statement –

Write a program to solve Classical Problems of Synchronization using Mutex and Semaphore.

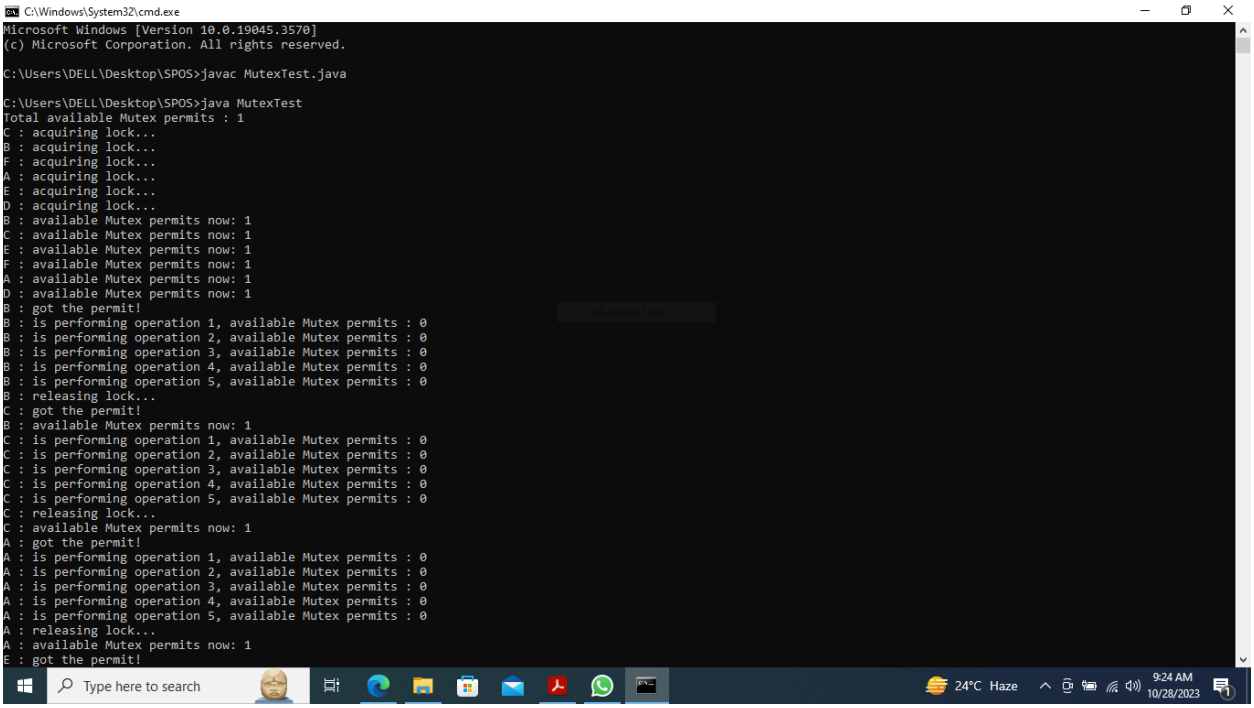
MutexTest.java –

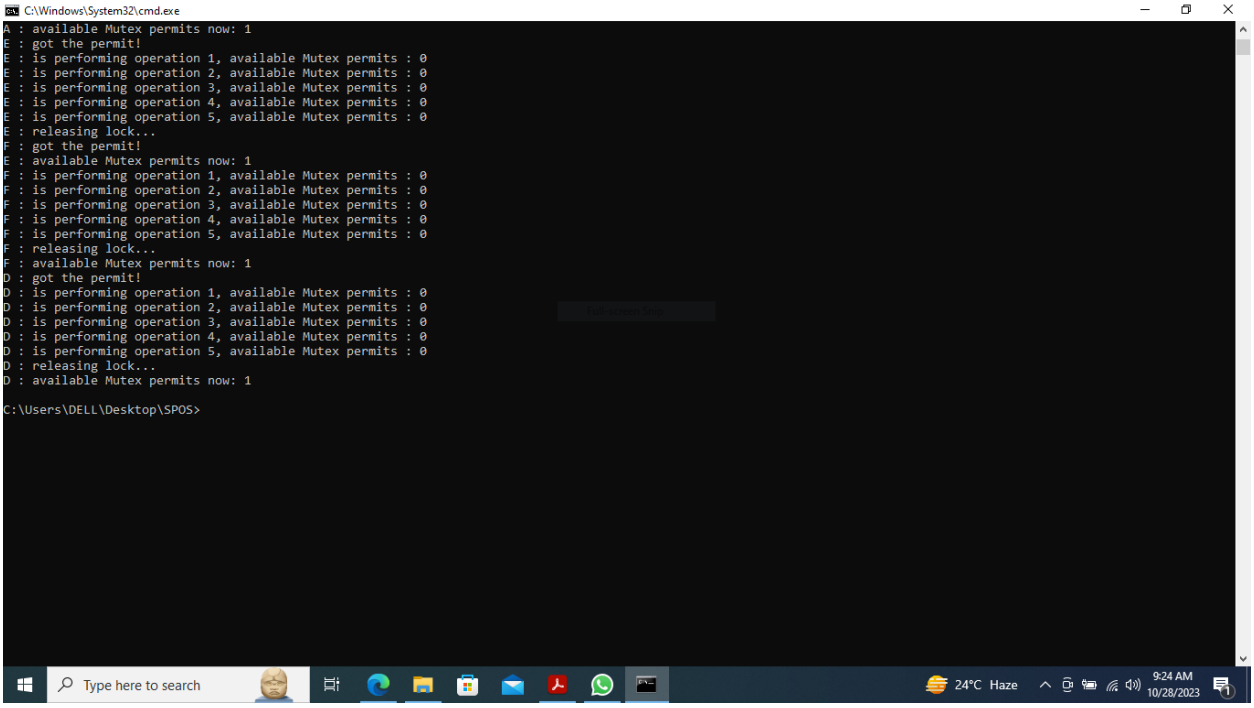
```
import java.util.concurrent.Semaphore;


public class MutexTest {
	static Semaphore semaphore = new Semaphore(1);

	static class MyLockerThread extends Thread {
		String name = "";

		MyLockerThread(String name) {
		this.name = name;
		}

		public void run() {
			try {
				System.out.println(name + " : acquiring lock...");
				System.out.println(name + " : available Mutex permits now: " +
				semaphore.availablePermits());
				semaphore.acquire();
				System.out.println(name + " : got the permit!");
				try {
					for (int i = 1; i <= 5; i++) {
						System.out.println(name + " : is performing operation "
+ i + ", available Mutex permits : " + semaphore.availablePermits());

						// sleep 1 second
```

```
                                        Thread.sleep(1000);
                                }
                        }
                        finally {
                                // Release the permit after a successful acquire
                                System.out.println(name + " : releasing lock...");
                                semaphore.release();
                                System.out.println(name + " : available Mutex permits now: " +
                                semaphore.availablePermits());
                        }
                }
                catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
public static void main(String[] args) {
        System.out.println("Total available Mutex permits : " + semaphore.availablePermits());
        MyLockerThread t1 = new MyLockerThread("A");
        t1.start();
        MyLockerThread t2 = new MyLockerThread("B");
        t2.start();
        MyLockerThread t3 = new MyLockerThread("C");
        t3.start();
        MyLockerThread t4 = new MyLockerThread("D");
        t4.start();
        MyLockerThread t5 = new MyLockerThread("E");
        t5.start();
        MyLockerThread t6 = new MyLockerThread("F");
        t6.start();
```

```
            // Join all threads to ensure they complete before the main program exits
            try {
                t1.join();
                t2.join();
                t3.join();
                t4.join();
                t5.join();
                t6.join();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
}
```

Output –




C:\Windows\System32\cmd.exe
A : available Mutex permits now: 1
E : got the permit!
E : is performing operation 1, available Mutex permits : 0
E : is performing operation 2, available Mutex permits : 0
E : is performing operation 3, available Mutex permits : 0
E : is performing operation 4, available Mutex permits : 0
E : is performing operation 5, available Mutex permits : 0
E : releasing lock...
F : got the permit!
E : available Mutex permits now: 1
F : is performing operation 1, available Mutex permits : 0
F : is performing operation 2, available Mutex permits : 0
F : is performing operation 3, available Mutex permits : 0
F : is performing operation 4, available Mutex permits : 0
F : is performing operation 5, available Mutex permits : 0
F : releasing lock...
F : available Mutex permits now: 1
D : got the permit!
D : is performing operation 1, available Mutex permits : 0
D : is performing operation 2, available Mutex permits : 0
D : is performing operation 3, available Mutex permits : 0
D : is performing operation 4, available Mutex permits : 0
D : is performing operation 5, available Mutex permits : 0
D : releasing lock...
D : available Mutex permits now: 1
C:\Users\DELL\Desktop\SPOS>
Type here to search
24°C Haze
9:24 AM
10/28/2023

PRACTICAL 4

Problem Statement –

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

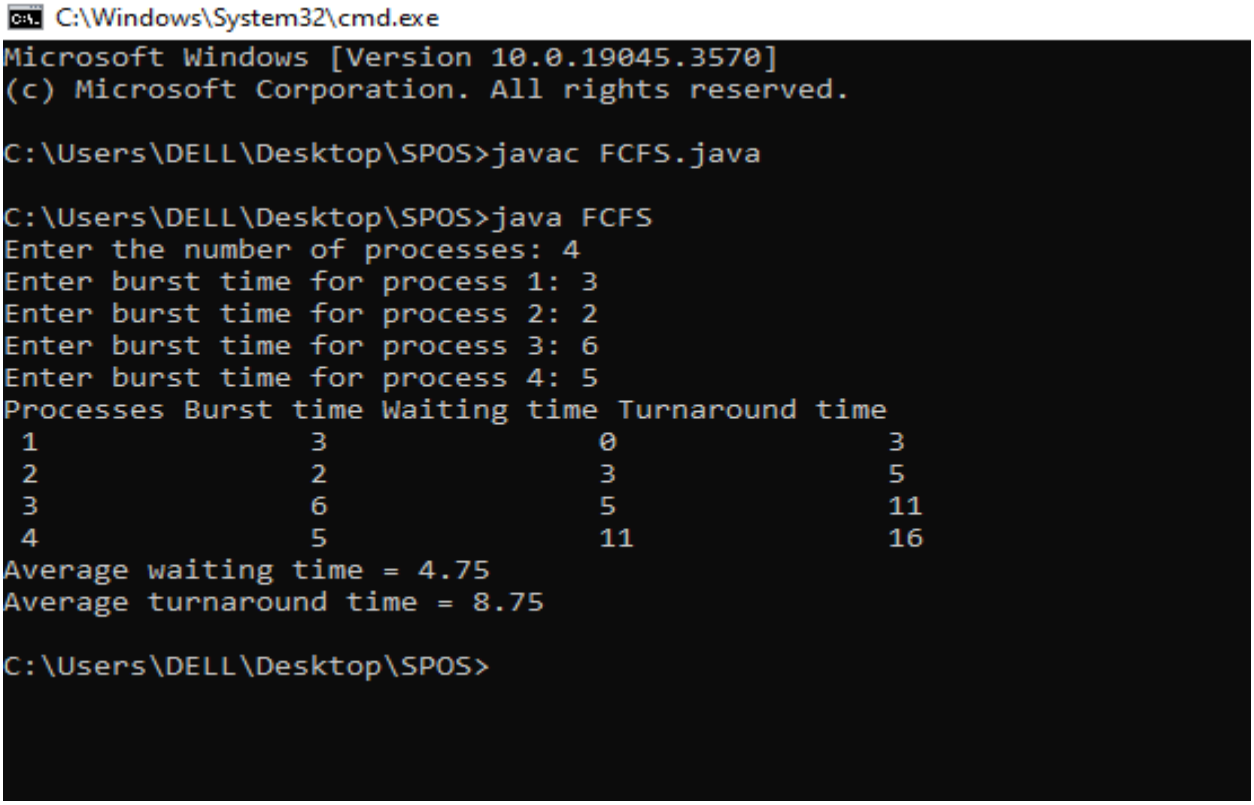
FCFS.java –

```
import java.util.Scanner;


class FCFS {
	// Function to find the waiting time for all processes
	static void findWaitingTime(int processes[], int n, int bt[], int wt[]) {
		// Waiting time for the first process is 0wt[0] = 0;
		// Calculating waiting time
		for (int i = 1; i < n; i++) {
			wt[i] = bt[i - 1] + wt[i - 1];
		}
	}
	// Function to calculate the turnaround time
	static void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
		// Calculating turnaround time by adding bt[i] + wt[i]
		for (int i = 0; i < n; i++) {
			tat[i] = bt[i] + wt[i];
		}
	}
	// Function to calculate average time
	void findavgTime(int processes[], int n, int bt[]) {
		int wt[] = new int[n], tat[] = new int[n];
		int total_wt = 0, total_tat = 0;
		// Function to find waiting time of all processes
```

```
        findWaitingTime(processes, n, bt, wt);
        // Function to find turnaround time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);
        // Display processes along with all details
        System.out.printf("Processes Burst time Waiting time Turnaround time\n");
        // Calculate total waiting time and total turnaround time
        for (int i = 0; i < n; i++) {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                System.out.printf(" %d \t\t %d \t\t %d \t\t %d\n", (i + 1), bt[i], wt[i], tat[i]);
        }
        // Calculate and display average waiting time and average turnaround time
        float avg_wt = (float) total_wt / n;
        float avg_tat = (float) total_tat / n;
        System.out.printf("Average waiting time = %.2f\n", avg_wt);
        System.out.printf("Average turnaround time = %.2f\n", avg_tat);
}
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();
        int processes[] = new int[n];
        int burst_time[] = new int[n];
        for (int i = 0; i < n; i++) {
                System.out.print("Enter burst time for process " + (i + 1) + ": ");
                burst_time[i] = sc.nextInt();
        }
        FCFS fcfs = new FCFS();
        fcfs.findavgTime(processes, n, burst_time);
}
```

}

Output –



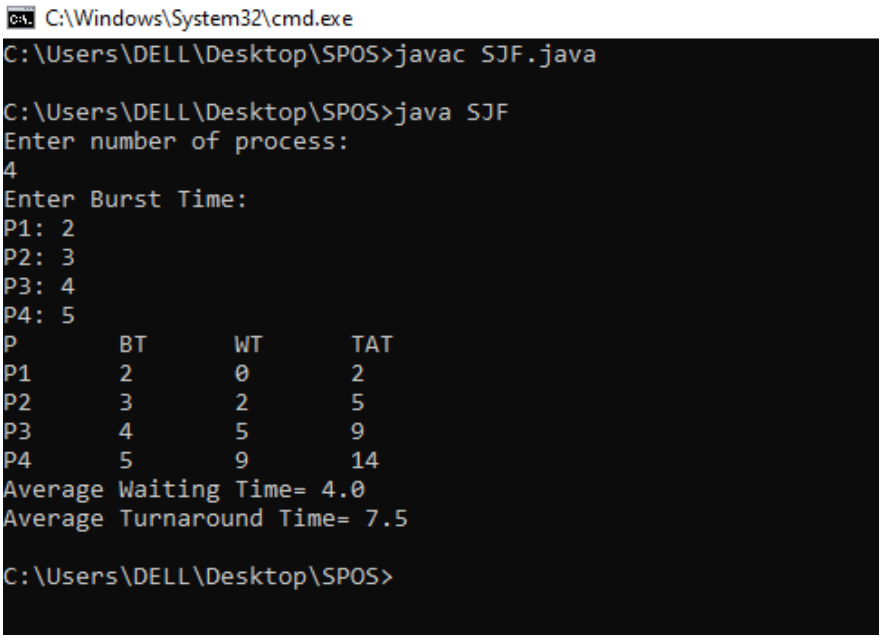
SJF.java –

```
import java.io.*;
import java.util.*;
public class Main {
	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		int n;
		// Matrix for storing Process Id, Burst
```

```
// Time, Average Waiting Time & Average
// Turn Around Time.
int[][] A = new int[100][4];
int total = 0;
float avg_wt, avg_tat;
System.out.println("Enter number of process:");
n = input.nextInt();
System.out.println("Enter Burst Time:");
for (int i = 0; i < n; i++) {
        // User Input Burst Time and alloting
        // Process Id.
        System.out.print("P" + (i + 1) + ": ");
        A[i][1] = input.nextInt();
        A[i][0] = i + 1;
}
for (int i = 0; i < n; i++) {
        // Sorting process according to their
        // Burst Time.
        int index = i;
        for (int j = i + 1; j < n; j++) {
                if (A[j][1] < A[index][1]) {
                        index = j;
                }
        }
        int temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;
        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
```

```
        }
        A[0][2] = 0;
        // Calculation of Waiting Times
        for (int i = 1; i < n; i++) {
            A[i][2] = 0;
            for (int j = 0; j < i; j++) {
                A[i][2] += A[j][1];
            }
            total += A[i][2];
        }
        avg_wt = (float)total / n;
        total = 0;
        // Calculation of Turn Around Time and printing the data.
        System.out.println("P\tBT\tWT\tTAT");
        for (int i = 0; i < n; i++) {
            A[i][3] = A[i][1] + A[i][2];
            total += A[i][3];
            System.out.println("P" + A[i][0] + "\t" + A[i][1] + "\t" + A[i][2] + "\t" + A[i][3]);
        }
        avg_tat = (float)total / n;
        System.out.println("Average Waiting Time= "+ avg_wt);
        System.out.println("Average Turnaround Time= "+ avg_tat);
    }
}
```

Output –



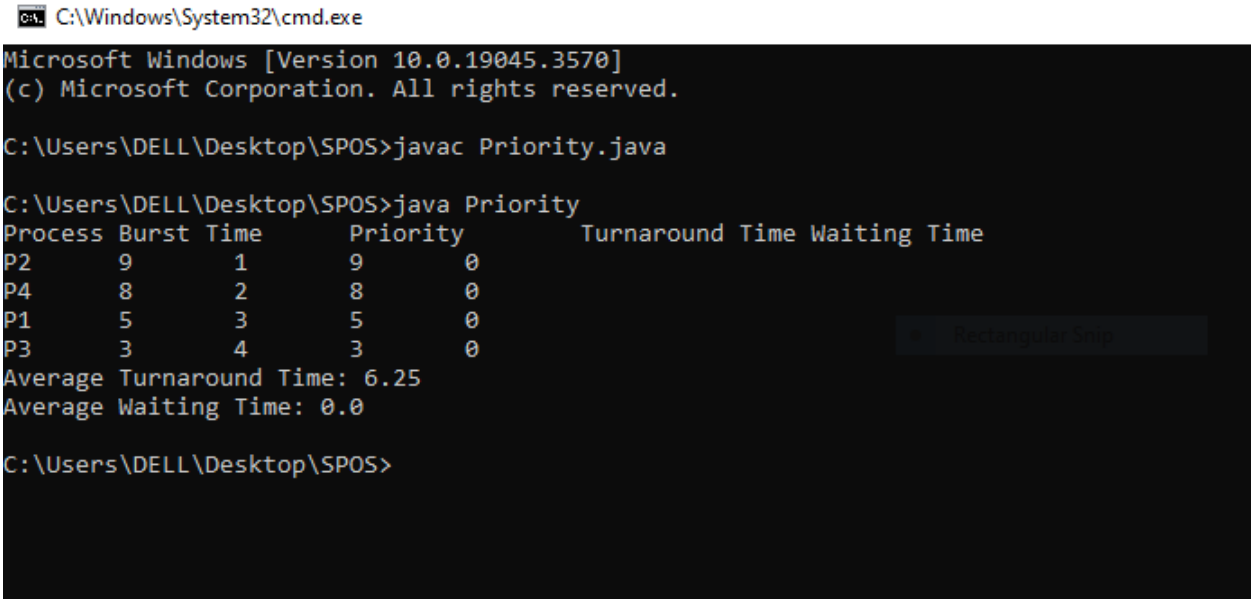
Priority –

```
class Priority {
        void priority(String processes[], int n, int burstTime[], int priority[]) {
                int numberOfProcess = n;
                int temp;
                String temp2;
                // Sorting processes by priority using Bubble Sort
                for (int i = 0; i < numberOfProcess - 1; i++) {
                        for (int j = 0; j < numberOfProcess - 1; j++) {
                                if (priority[j] > priority[j + 1]) {
                                        temp = priority[j];
                                        priority[j] = priority[j + 1];
                                        priority[j + 1] = temp;
```

```
                    temp = burstTime[j];
                    burstTime[j] = burstTime[j + 1];
                    burstTime[j + 1] = temp;
                    temp2 = processes[j];
                    processes[j] = processes[j + 1];
                    processes[j + 1] = temp2;
                }
            }
        }
        // TAT - Turn Around Time and Waiting Time
        int TAT[] = new int[numberOfProcess];
        int waitingTime[] = new int[numberOfProcess];
        TAT[0] = burstTime[0];
        waitingTime[0] = 0;
        for (int i = 1; i < numberOfProcess; i++) {
            TAT[i] = waitingTime[i - 1] + burstTime[i];
            waitingTime[i] = TAT[i] - burstTime[i];
        }
        // Calculate Average Turnaround Time and Average Waiting Time
        int totalTAT = 0;
        int totalWaitingTime = 0;
        for (int i = 0; i < numberOfProcess; i++) {
            totalTAT += TAT[i];
            totalWaitingTime += waitingTime[i];
        }
        double avgTAT = (double) totalTAT / numberOfProcess;
        double avgWaitingTime = (double) totalWaitingTime / numberOfProcess;
        // Display the results
        System.out.println("Process\tBurst Time\tPriority\tTurnaround Time\tWaiting Time");
        for (int i = 0; i < numberOfProcess; i++) {
```

```
                        System.out.println(processes[i] + "\t" + burstTime[i] + "\t" + priority[i] + "\t" +
TAT[i] + "\t"+ waitingTime[i]);
                }
                System.out.println("Average Turnaround Time: " + avgTAT);
                System.out.println("Average Waiting Time: " + avgWaitingTime);
        }

        public static void main(String[] args) {
                Priority scheduler = new Priority();
                String processes[] = {"P1", "P2", "P3", "P4"};
                int burstTime[] = {5, 9, 3, 8};
                int priority[] = {3, 1, 4, 2};
                int n = processes.length;
                scheduler.priority(processes, n, burstTime, priority);
        }
}
```

Output -

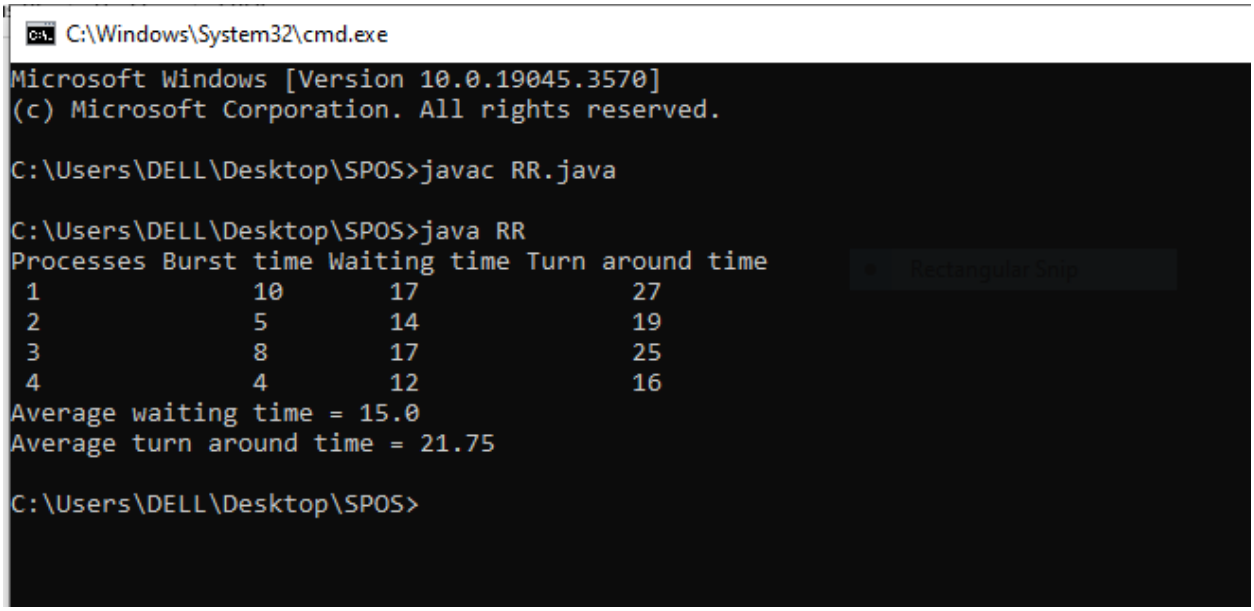


RR –

```
class RR {
	static void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum) {
		int rem_bt[] = new int[n];
		for (int i = 0; i < n; i++)
		rem_bt[i] = bt[i];
		int t = 0;
		while (true) {
			boolean done = true;
			for (int i = 0; i < n; i++) {
				if (rem_bt[i] > 0) {
				done = false;
				if (rem_bt[i] > quantum) {
					t += quantum;
					rem_bt[i] -= quantum;
				}
				else {
					t += rem_bt[i];
					wt[i] = t - bt[i];
					rem_bt[i] = 0;
				}
			}
		}
		if (done)
		break;
	}
}
 static void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
	for (int i = 0; i < n; i++)
		tat[i] = bt[i] + wt[i];
```

```
    }
    void findavgTime(int processes[], int n, int bt[], int quantum) {
        int wt[] = new int[n], tat[] = new int[n];
        int total_wt = 0, total_tat = 0;
        findWaitingTime(processes, n, bt, wt, quantum);
        findTurnAroundTime(processes, n, bt, wt, tat);
        System.out.println("Processes Burst time Waiting time Turn around time");
        for (int i = 0; i < n; i++) {
            total_wt += wt[i];
            total_tat += tat[i];
            System.out.println(" " + (i + 1) + "\t\t" + bt[i] + "\t " + wt[i] + "\t\t " + tat[i]);
        }
        System.out.println("Average waiting time = " + (float) total_wt / n);
        System.out.println("Average turn around time = " + (float) total_tat / n);
    }
    public static void main(String[] args) {
        RR scheduler = new RR();
        int processes[] = {1, 2, 3, 4};
        int burstTime[] = {10, 5, 8, 4};
        int quantum = 2;
        int n = processes.length;
        scheduler.findavgTime(processes, n, burstTime, quantum);
    }
}
```

Output –


C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.
C:\Users\DELL\Desktop\SPOS>javac RR.java
C:\Users\DELL\Desktop\SPOS>java RR
Processes Burst time Waiting time Turn around time
1 10 17 27
2 5 14 19
3 8 17 25
4 4 12 16
Average waiting time = 15.0
Average turn around time = 21.75
C:\Users\DELL\Desktop\SPOS>