**Assignment 1**

**Input:**

**Pass1.java**

import java.io.\*;

import java.util.ArrayList;

import java.util.LinkedHashMap;

public class Pass1 {

int lc = 0;

int libtab\_ptr = 0, pooltab\_ptr = 0;

int symIndex = 0, litIndex = 0;

String out;

LinkedHashMap<String, TableRow> SYMTAB;

ArrayList<TableRow> LITTAB;

ArrayList<Integer> POOLTAB;

private BufferedReader br;

public Pass1() {

SYMTAB = new LinkedHashMap<>();

LITTAB = new ArrayList<>();

POOLTAB = new ArrayList<>();

lc = 0;

POOLTAB.add(0);

}

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

Pass1 obj = new Pass1();

obj.readFile();

}

public void readFile() throws Exception {

String prev = "";

String line;

String code;

new MOT();

br = new BufferedReader(new FileReader("input1.asm"));

while ((line = br.readLine()) != null) {

// System.out.println(line);

String page[] = line.split("\\s+");

if (page[0] != "") {//LABLE VERIFICATION

// System.out.print("\tLable->" + page[0] + "\t");

if (SYMTAB.containsKey(page[0])) {

SYMTAB.put(page[0], new TableRow(page[0], lc, SYMTAB.get(page[0]).getIndex()));

lc++;

} else {

SYMTAB.put(page[0], new TableRow(page[0], lc, ++symIndex));

}

} else {

System.out.print("\t\t\t");

}

if (page[1].equals("START")) //CHECK START

{

lc = Integer.parseInt(page[2]);

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + page[2] + ")";

} else if (page[1].equals("EQU")) {

String split1[] = page[2].split("\\+");

int loc = SYMTAB.get(split1[0]).getAddress() + Integer.parseInt(split1[1]);

//below If conditions are optional as no IC is generated for them

if (page[2].contains("+")) {

String partsslp[] = page[2].split("\\+");

out = "(AD,04)\t(S," + SYMTAB.get(partsslp[0]).getIndex() + ") + " + Integer.parseInt(partsslp[1]);

} else if (page[2].contains("-")) {

String partsslp[] = page[2].split("\\-");

out = "(AD,04)\t(S," + SYMTAB.get(partsslp[0]).getIndex() + ")-" + Integer.parseInt(partsslp[1]);

} else {

out = "(AD,04)\t(C," + Integer.parseInt(page[2] + ")");

}

if (SYMTAB.containsKey(page[0]))

SYMTAB.put(page[0], new TableRow(page[0], loc, SYMTAB.get(page[0]).getIndex()));

else

SYMTAB.put(page[0], new TableRow(page[0], loc, ++symIndex));

} else if (page[1].equals("ORIGIN")) {

String partslp[] = (page[2]).split("\\+");

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(S," + SYMTAB.get(partslp[0]).getIndex() + ") + " + Integer.parseInt(partslp[1]);

} else if (page[1].equals("LTORG")) {

int ptr = POOLTAB.get(pooltab\_ptr);

for (int j = ptr; j < libtab\_ptr; j++) {

lc++;

LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + LITTAB.get(j).symbol + ")";

}

pooltab\_ptr++;

lc++;

POOLTAB.add(libtab\_ptr);

} else if (page[1].equals("DC")) {

int size = Integer.parseInt(page[2].replace("'", ""));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + size + ")";

} else if (page[1].equals("DS")) {

lc++;

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + page[2] + ")";

} else if (page[1].equals("END")) {

int ptr = POOLTAB.get(pooltab\_ptr);

for (int j = ptr; j < pooltab\_ptr; j++) {

lc++;

LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + LITTAB.get(j).symbol + ")";

}

pooltab\_ptr++;

POOLTAB.add(libtab\_ptr);

out = "(AD,02)";

}

if (MOT.checkcls(page[1]).equals("IS")) {

int len = page.length;

String FinalOp = "";

for (int i = 2; i < len; i++) {

page[i] = page[i].replace(",", "");

if (MOT.checkcls(page[i]) == "RG") {

// FinalOp += MOT.checkop(page[i]) + "\t";

FinalOp += " (RG,"+MOT.checkop(page[i])+") ";

} else {

if (page[i].contains("=")) {

page[i] = page[i].replace("=", "");

page[i] = page[i].replace("'", "");

LITTAB.add(new TableRow(page[i], -1, ++litIndex));

libtab\_ptr++;

FinalOp += "(L," + libtab\_ptr + ")";

} else if (SYMTAB.containsKey(page[i])) {

int Sindex = SYMTAB.get(page[i]).getIndex();

FinalOp += "(S," + Sindex + ")";

} else {

SYMTAB.put(page[i], new TableRow(page[i], -1, ++symIndex));

int Sindex = SYMTAB.get(page[i]).getIndex();

FinalOp += "(S," + Sindex + ")";

}

}

}

lc++;

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")" + FinalOp;

}

// System.out.print("Mneumonic->" + page[1] + "\n");

}

Printer("PTab");

Printer("LTab");

Printer("STab");

}

public void Printer(String OPT) throws IOException {

switch(OPT){

case "STab":

BufferedWriter rw=new BufferedWriter(new FileWriter("SYMTAB.txt"));

//Printing Symbol Table

java.util.Iterator<String> iterator = SYMTAB.keySet().iterator();

System.out.println("\n\n\tSYMBOL TABLE");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

while (iterator.hasNext()) {

String key = iterator.next().toString();

TableRow value = SYMTAB.get(key);

System.out.println("|\s\s"+value.getIndex()+"\t|\s\s" + value.getSymbol()+"\t|\s\s\s"+value.getAddress()+"\t\t|");

// System.out.println("|.......|.......|...............|");

rw.write(value.getIndex()+"\t" + value.getSymbol()+"\t"+value.getAddress()+"\n");

}

rw.close();

break;

case "PTab":

BufferedWriter prw=new BufferedWriter(new FileWriter("POOLTAB.txt"));

System.out.println("\n\n\tPOOL TABLE \n");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

for (int i = 0; i <POOLTAB.size()-2; i++) {

System.out.println("|\s\s"+(i+1)+"\t|\s\s#"+(POOLTAB.get(i)+1)+"\t|");

// System.out.println("|.......|...............|");

prw.write((i+1)+"\t#"+(POOLTAB.get(i)+1)+"\n");

}

prw.close();

break;

case "LTab":

BufferedWriter lrw=new BufferedWriter(new FileWriter("LITTAB.txt"));

System.out.println("\n\n\tLiteral Table");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//Processing LITTAB

for(int i=0;i<LITTAB.size();i++)

{

TableRow row=LITTAB.get(i);

System.out.println("|\s\s"+(i+1)+"\t|\s\s"+row.getSymbol()+"\t|\s\s\s"+row.getAddress()+"\t\t|");

// System.out.println("|.......|.......|...............|");

lrw.write((i+1)+"\t"+row.getSymbol()+"\t"+row.getAddress()+"\n");

}

lrw.close();

break;

default:

System.out.println("Data Set Not provided");

break;

}

}

}

**MOT.java**

import java.io.\*;

import java.util.ArrayList;

import java.util.LinkedHashMap;

public class Pass1 {

int lc = 0;

int libtab\_ptr = 0, pooltab\_ptr = 0;

int symIndex = 0, litIndex = 0;

String out;

LinkedHashMap<String, TableRow> SYMTAB;

ArrayList<TableRow> LITTAB;

ArrayList<Integer> POOLTAB;

private BufferedReader br;

public Pass1() {

SYMTAB = new LinkedHashMap<>();

LITTAB = new ArrayList<>();

POOLTAB = new ArrayList<>();

lc = 0;

POOLTAB.add(0);

}

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

Pass1 obj = new Pass1();

obj.readFile();

}

public void readFile() throws Exception {

String prev = "";

String line;

String code;

new MOT();

br = new BufferedReader(new FileReader("input1.asm"));

while ((line = br.readLine()) != null) {

// System.out.println(line);

String page[] = line.split("\\s+");

if (page[0] != "") {//LABLE VERIFICATION

// System.out.print("\tLable->" + page[0] + "\t");

if (SYMTAB.containsKey(page[0])) {

SYMTAB.put(page[0], new TableRow(page[0], lc, SYMTAB.get(page[0]).getIndex()));

lc++;

} else {

SYMTAB.put(page[0], new TableRow(page[0], lc, ++symIndex));

}

} else {

System.out.print("\t\t\t");

}

if (page[1].equals("START")) //CHECK START

{

lc = Integer.parseInt(page[2]);

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + page[2] + ")";

} else if (page[1].equals("EQU")) {

String split1[] = page[2].split("\\+");

int loc = SYMTAB.get(split1[0]).getAddress() + Integer.parseInt(split1[1]);

//below If conditions are optional as no IC is generated for them

if (page[2].contains("+")) {

String partsslp[] = page[2].split("\\+");

out = "(AD,04)\t(S," + SYMTAB.get(partsslp[0]).getIndex() + ") + " + Integer.parseInt(partsslp[1]);

} else if (page[2].contains("-")) {

String partsslp[] = page[2].split("\\-");

out = "(AD,04)\t(S," + SYMTAB.get(partsslp[0]).getIndex() + ")-" + Integer.parseInt(partsslp[1]);

} else {

out = "(AD,04)\t(C," + Integer.parseInt(page[2] + ")");

}

if (SYMTAB.containsKey(page[0]))

SYMTAB.put(page[0], new TableRow(page[0], loc, SYMTAB.get(page[0]).getIndex()));

else

SYMTAB.put(page[0], new TableRow(page[0], loc, ++symIndex));

} else if (page[1].equals("ORIGIN")) {

String partslp[] = (page[2]).split("\\+");

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(S," + SYMTAB.get(partslp[0]).getIndex() + ") + " + Integer.parseInt(partslp[1]);

} else if (page[1].equals("LTORG")) {

int ptr = POOLTAB.get(pooltab\_ptr);

for (int j = ptr; j < libtab\_ptr; j++) {

lc++;

LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + LITTAB.get(j).symbol + ")";

}

pooltab\_ptr++;

lc++;

POOLTAB.add(libtab\_ptr);

} else if (page[1].equals("DC")) {

int size = Integer.parseInt(page[2].replace("'", ""));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + size + ")";

} else if (page[1].equals("DS")) {

lc++;

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + page[2] + ")";

} else if (page[1].equals("END")) {

int ptr = POOLTAB.get(pooltab\_ptr);

for (int j = ptr; j < pooltab\_ptr; j++) {

lc++;

LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + LITTAB.get(j).symbol + ")";

}

pooltab\_ptr++;

POOLTAB.add(libtab\_ptr);

out = "(AD,02)";

}

if (MOT.checkcls(page[1]).equals("IS")) {

int len = page.length;

String FinalOp = "";

for (int i = 2; i < len; i++) {

page[i] = page[i].replace(",", "");

if (MOT.checkcls(page[i]) == "RG") {

// FinalOp += MOT.checkop(page[i]) + "\t";

FinalOp += " (RG,"+MOT.checkop(page[i])+") ";

} else {

if (page[i].contains("=")) {

page[i] = page[i].replace("=", "");

page[i] = page[i].replace("'", "");

LITTAB.add(new TableRow(page[i], -1, ++litIndex));

libtab\_ptr++;

FinalOp += "(L," + libtab\_ptr + ")";

} else if (SYMTAB.containsKey(page[i])) {

int Sindex = SYMTAB.get(page[i]).getIndex();

FinalOp += "(S," + Sindex + ")";

} else {

SYMTAB.put(page[i], new TableRow(page[i], -1, ++symIndex));

int Sindex = SYMTAB.get(page[i]).getIndex();

FinalOp += "(S," + Sindex + ")";

}

}

}

lc++;

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")" + FinalOp;

}

// System.out.print("Mneumonic->" + page[1] + "\n");

}

Printer("PTab");

Printer("LTab");

Printer("STab");

}

public void Printer(String OPT) throws IOException {

switch(OPT){

case "STab":

BufferedWriter rw=new BufferedWriter(new FileWriter("SYMTAB.txt"));

//Printing Symbol Table

java.util.Iterator<String> iterator = SYMTAB.keySet().iterator();

System.out.println("\n\n\tSYMBOL TABLE");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

while (iterator.hasNext()) {

String key = iterator.next().toString();

TableRow value = SYMTAB.get(key);

System.out.println("|\s\s"+value.getIndex()+"\t|\s\s" + value.getSymbol()+"\t|\s\s\s"+value.getAddress()+"\t\t|");

// System.out.println("|.......|.......|...............|");

rw.write(value.getIndex()+"\t" + value.getSymbol()+"\t"+value.getAddress()+"\n");

}

rw.close();

break;

case "PTab":

BufferedWriter prw=new BufferedWriter(new FileWriter("POOLTAB.txt"));

System.out.println("\n\n\tPOOL TABLE \n");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

for (int i = 0; i <POOLTAB.size()-2; i++) {

System.out.println("|\s\s"+(i+1)+"\t|\s\s#"+(POOLTAB.get(i)+1)+"\t|");

// System.out.println("|.......|...............|");

prw.write((i+1)+"\t#"+(POOLTAB.get(i)+1)+"\n");

}

prw.close();

break;

case "LTab":

BufferedWriter lrw=new BufferedWriter(new FileWriter("LITTAB.txt"));

System.out.println("\n\n\tLiteral Table");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//Processing LITTAB

for(int i=0;i<LITTAB.size();i++)

{

TableRow row=LITTAB.get(i);

System.out.println("|\s\s"+(i+1)+"\t|\s\s"+row.getSymbol()+"\t|\s\s\s"+row.getAddress()+"\t\t|");

// System.out.println("|.......|.......|...............|");

lrw.write((i+1)+"\t"+row.getSymbol()+"\t"+row.getAddress()+"\n");

}

lrw.close();

break;

default:

System.out.println("Data Set Not provided");

break;

}

}

}

**TableRow.java**

import java.io.\*;

import java.util.ArrayList;

import java.util.LinkedHashMap;

public class Pass1 {

int lc = 0;

int libtab\_ptr = 0, pooltab\_ptr = 0;

int symIndex = 0, litIndex = 0;

String out;

LinkedHashMap<String, TableRow> SYMTAB;

ArrayList<TableRow> LITTAB;

ArrayList<Integer> POOLTAB;

private BufferedReader br;

public Pass1() {

SYMTAB = new LinkedHashMap<>();

LITTAB = new ArrayList<>();

POOLTAB = new ArrayList<>();

lc = 0;

POOLTAB.add(0);

}

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

Pass1 obj = new Pass1();

obj.readFile();

}

public void readFile() throws Exception {

String prev = "";

String line;

String code;

new MOT();

br = new BufferedReader(new FileReader("input1.asm"));

while ((line = br.readLine()) != null) {

// System.out.println(line);

String page[] = line.split("\\s+");

if (page[0] != "") {//LABLE VERIFICATION

// System.out.print("\tLable->" + page[0] + "\t");

if (SYMTAB.containsKey(page[0])) {

SYMTAB.put(page[0], new TableRow(page[0], lc, SYMTAB.get(page[0]).getIndex()));

lc++;

} else {

SYMTAB.put(page[0], new TableRow(page[0], lc, ++symIndex));

}

} else {

System.out.print("\t\t\t");

}

if (page[1].equals("START")) //CHECK START

{

lc = Integer.parseInt(page[2]);

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + page[2] + ")";

} else if (page[1].equals("EQU")) {

String split1[] = page[2].split("\\+");

int loc = SYMTAB.get(split1[0]).getAddress() + Integer.parseInt(split1[1]);

//below If conditions are optional as no IC is generated for them

if (page[2].contains("+")) {

String partsslp[] = page[2].split("\\+");

out = "(AD,04)\t(S," + SYMTAB.get(partsslp[0]).getIndex() + ") + " + Integer.parseInt(partsslp[1]);

} else if (page[2].contains("-")) {

String partsslp[] = page[2].split("\\-");

out = "(AD,04)\t(S," + SYMTAB.get(partsslp[0]).getIndex() + ")-" + Integer.parseInt(partsslp[1]);

} else {

out = "(AD,04)\t(C," + Integer.parseInt(page[2] + ")");

}

if (SYMTAB.containsKey(page[0]))

SYMTAB.put(page[0], new TableRow(page[0], loc, SYMTAB.get(page[0]).getIndex()));

else

SYMTAB.put(page[0], new TableRow(page[0], loc, ++symIndex));

} else if (page[1].equals("ORIGIN")) {

String partslp[] = (page[2]).split("\\+");

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(S," + SYMTAB.get(partslp[0]).getIndex() + ") + " + Integer.parseInt(partslp[1]);

} else if (page[1].equals("LTORG")) {

int ptr = POOLTAB.get(pooltab\_ptr);

for (int j = ptr; j < libtab\_ptr; j++) {

lc++;

LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + LITTAB.get(j).symbol + ")";

}

pooltab\_ptr++;

lc++;

POOLTAB.add(libtab\_ptr);

} else if (page[1].equals("DC")) {

int size = Integer.parseInt(page[2].replace("'", ""));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + size + ")";

} else if (page[1].equals("DS")) {

lc++;

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + page[2] + ")";

} else if (page[1].equals("END")) {

int ptr = POOLTAB.get(pooltab\_ptr);

for (int j = ptr; j < pooltab\_ptr; j++) {

lc++;

LITTAB.set(j, new TableRow(LITTAB.get(j).getSymbol(), lc));

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")\t(C," + LITTAB.get(j).symbol + ")";

}

pooltab\_ptr++;

POOLTAB.add(libtab\_ptr);

out = "(AD,02)";

}

if (MOT.checkcls(page[1]).equals("IS")) {

int len = page.length;

String FinalOp = "";

for (int i = 2; i < len; i++) {

page[i] = page[i].replace(",", "");

if (MOT.checkcls(page[i]) == "RG") {

// FinalOp += MOT.checkop(page[i]) + "\t";

FinalOp += " (RG,"+MOT.checkop(page[i])+") ";

} else {

if (page[i].contains("=")) {

page[i] = page[i].replace("=", "");

page[i] = page[i].replace("'", "");

LITTAB.add(new TableRow(page[i], -1, ++litIndex));

libtab\_ptr++;

FinalOp += "(L," + libtab\_ptr + ")";

} else if (SYMTAB.containsKey(page[i])) {

int Sindex = SYMTAB.get(page[i]).getIndex();

FinalOp += "(S," + Sindex + ")";

} else {

SYMTAB.put(page[i], new TableRow(page[i], -1, ++symIndex));

int Sindex = SYMTAB.get(page[i]).getIndex();

FinalOp += "(S," + Sindex + ")";

}

}

}

lc++;

out = "(" + MOT.checkcls(page[1]) + "," + MOT.checkop(page[1]) + ")" + FinalOp;

}

// System.out.print("Mneumonic->" + page[1] + "\n");

}

Printer("PTab");

Printer("LTab");

Printer("STab");

}

public void Printer(String OPT) throws IOException {

switch(OPT){

case "STab":

BufferedWriter rw=new BufferedWriter(new FileWriter("SYMTAB.txt"));

//Printing Symbol Table

java.util.Iterator<String> iterator = SYMTAB.keySet().iterator();

System.out.println("\n\n\tSYMBOL TABLE");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

while (iterator.hasNext()) {

String key = iterator.next().toString();

TableRow value = SYMTAB.get(key);

System.out.println("|\s\s"+value.getIndex()+"\t|\s\s" + value.getSymbol()+"\t|\s\s\s"+value.getAddress()+"\t\t|");

// System.out.println("|.......|.......|...............|");

rw.write(value.getIndex()+"\t" + value.getSymbol()+"\t"+value.getAddress()+"\n");

}

rw.close();

break;

case "PTab":

BufferedWriter prw=new BufferedWriter(new FileWriter("POOLTAB.txt"));

System.out.println("\n\n\tPOOL TABLE \n");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

for (int i = 0; i <POOLTAB.size()-2; i++) {

System.out.println("|\s\s"+(i+1)+"\t|\s\s#"+(POOLTAB.get(i)+1)+"\t|");

// System.out.println("|.......|...............|");

prw.write((i+1)+"\t#"+(POOLTAB.get(i)+1)+"\n");

}

prw.close();

break;

case "LTab":

BufferedWriter lrw=new BufferedWriter(new FileWriter("LITTAB.txt"));

System.out.println("\n\n\tLiteral Table");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//Processing LITTAB

for(int i=0;i<LITTAB.size();i++)

{

TableRow row=LITTAB.get(i);

System.out.println("|\s\s"+(i+1)+"\t|\s\s"+row.getSymbol()+"\t|\s\s\s"+row.getAddress()+"\t\t|");

// System.out.println("|.......|.......|...............|");

lrw.write((i+1)+"\t"+row.getSymbol()+"\t"+row.getAddress()+"\n");

}

lrw.close();

break;

default:

System.out.println("Data Set Not provided");

break;

}

}

}

**Pass2.java**

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("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\intermediate.txt"));

BufferedReader b2 = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\symtab.txt"));

BufferedReader b3 = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\littab.txt"));

FileWriter f1 = new FileWriter("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\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();

System.out.println("Pass2 file has been created");

BufferedReader op = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\Pass2.txt"));

while((s=op.readLine())!=null){

System.out.println(s);

}

}

}

**Input1.asm**

START 200

A DS 3

L1 MOVER AREG, B

ADD AREG, C

MOVEM AREG, ='2'

MOVEM AREG, ='3'

D EQU A+1

LTORG

L2 PRINT D

MOVEM AREG, ='4'

MOVEM AREG, ='5'

ORIGIN L2+1

LTORG

B DC '19

C DC '17

END

**Output:**

Intermediate Code Generated Sucessfully!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(AD,1) (C,200)

(DL,1) (C,3)

(IS,4) (RG,1) (S,3)

(IS,1) (RG,1) (S,4)

(IS,5) (RG,1) (L,1)

(IS,5) (RG,1) (L,2)

(AD,04) (S,1) + 1

(AD,5) (C,3)

(IS,10)(S,5)

(IS,5) (RG,1) (L,3)

(IS,5) (RG,1) (L,4)

(AD,3) (S,6) + 1

(AD,5) (C,5)

(DL,2) (C,19)

(DL,2) (C,17)

(AD,02)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

POOL TABLE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| 1 | #1 |

| 2 | #3 |

Literal Table

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| 1 | 2 | 206 |

| 2 | 3 | 207 |

| 3 | 4 | 212 |

| 4 | 5 | 213 |

SYMBOL TABLE

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| 1 | A | 200 |

| 2 | L1 | 201 |

| 3 | B | 214 |

| 4 | C | 215 |

| 5 | D | 201 |

| 6 | L2 | 208 |

**Pass2.txt**

+ 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

+ 00 0 001

**Assignment 2**

**Input:**

**Macro.java**

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("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\intermediate.txt"));

BufferedReader b2 = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\symtab.txt"));

BufferedReader b3 = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\littab.txt"));

FileWriter f1 = new FileWriter("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\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();

System.out.println("Pass2 file has been created");

BufferedReader op = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\Programs\\Expt-2\\Pass2.txt"));

while((s=op.readLine())!=null){

System.out.println(s);

}

}

}

**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("D:\\SITS\_NARHE\\LP-1\\intermediate.txt"));

BufferedReader b2 = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\mnt.txt"));

BufferedReader b3 = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\mdt.txt"));

BufferedReader b4 = new BufferedReader(new FileReader("D:\\SITS\_NARHE\\LP-1\\kpdt.txt"));

FileWriter f1 = new FileWriter("D:\\SITS\_NARHE\\LP-1\\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())),initializedParams[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.txt**

START

MACRO

INCR &ARG3 &ARG2

ADD AREG &ARG1

MOVER BREG &ARG1

MEND

MACRO

PVG &ARG2 &ARG1

SUB AREG &ARG2

MOVER CREG & ARG1

MEND

INCR

DECR

DATA2

END

**Output:**

\*\*\*\*\*\*\*\*\*PASS-1 MACROPROCESSOR\*\*\*\*\*\*\*\*\*\*\*

MACRO NAME TABLE (MNT)

I Macro\_Name Location

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

Intermediate

M1 10,20,&b=CREG

M2 100,200,&u=&AREG,&v=&BREG

KPDT

a AREG

b -

u CREG

v DREG

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

\*/

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'

**Assignment 3**

**Input:**

**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 first process is 0

wt[0] = 0;

// calculating waiting time

for (int i = 1; i < n; i++) {

wt[i] = bt[i - 1] + wt[i - 1];

}

}

// Function to calculate turn around time

static void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[]) {

// calculating turn around 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 turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

System.out.printf("Processes Burst time Waiting"

+" time Turn around time\n");

// Calculate total waiting time and total turn

// around time

for (int i = 0; i < n; i++) {

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

System.out.printf(" %d ", (i + 1));

System.out.printf(" %d ", bt[i]);

System.out.printf(" %d", wt[i]);

System.out.printf(" %d\n", tat[i]);

}

float s = (float)total\_wt /(float) n;

int t = total\_tat / n;

System.out.printf("Average waiting time = %f", s);

System.out.printf("\n");

System.out.printf("Average turn around time = %d ", t);

}

}

//Shortest Remaining Time First(SJF preemptive)

class Process

{

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

public Process(int pid, int bt, int art)

{

this.pid = pid;

this.bt = bt;

this.art = art;

}

}

class SJF

{

// Method to find the waiting time for all

// processes

static void findWaitingTime(Process proc[], int n, int wt[]){

int rt[] = new int[n];

// Copy the burst time into rt[]

for (int i = 0; i < n; i++)

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = Integer.MAX\_VALUE;

int shortest = 0, finish\_time;

boolean check = false;

// Process until all processes gets

// completed

while (complete != n) {

// Find process with minimum

// remaining time among the

// processes that arrives till the

// current time`

for (int j = 0; j < n; j++)

{

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = Integer.MAX\_VALUE;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current

// process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Method to calculate turn around time

static void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

// calculating turn around time by adding

// bt[i] + wt[i]

for (int i = 0; i < n; i++)

tat[i] = proc[i].bt + wt[i];

}

// Method to calculate average time

void findavgTime(Process proc[], int n)

{

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(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all

// details

System.out.println("Processes " +

" Burst time " +

" Waiting time " +

" Turn around time");

// Calculate total waiting time and

// total turn around time

for (int i = 0; i < n; i++) {

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

System.out.println(" " + proc[i].pid + "\t\t"

+ proc[i].bt + "\t\t " + wt[i]

+ "\t\t" + tat[i]);

}

System.out.println("Average waiting time = " +

(float)total\_wt / (float)n);

System.out.println("Average turn around time = " +

(float)total\_tat / (float)n);

}

}

class RR

{

//Method to find the waiting time for all

//processes

static void findWaitingTime(int processes[], int n,

int bt[], int wt[], int quantum)

{

// Make a copy of burst times bt[] to store remaining

// burst times.

int rem\_bt[] = new int[n];

for (int i = 0 ; i < n ; i++)

rem\_bt[i] = bt[i];

int t = 0; // Current time

// Keep traversing processes in round robin manner

// until all of them are not done.

while(true)

{

boolean done = true;

// Traverse all processes one by one repeatedly

for (int i = 0 ; i < n; i++)

{

// If burst time of a process is greater than 0

// then only need to process further

if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

if (rem\_bt[i] > quantum)

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t += quantum;

// Decrease the burst\_time of current process

// by quantum

rem\_bt[i] -= quantum;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process

else

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t = t + rem\_bt[i];

// Waiting time is current time minus time

// used by this process

wt[i] = t - bt[i];

// As the process gets fully executed

// make its remaining burst time = 0

rem\_bt[i] = 0;

}

}

}

// If all processes are done

if (done == true)

break;

}

}

// Method to calculate turn around time

static void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[])

{

// calculating turn around time by adding

// bt[i] + wt[i]

for (int i = 0; i < n ; i++)

tat[i] = bt[i] + wt[i];

}

// Method to calculate average time

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;

// Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt, quantum);

// Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

// Display processes along with all details

System.out.println("Processes " + " Burst time " +

" Waiting time " + " Turn around time");

// Calculate total waiting time and total turn

// around time

for (int i=0; i<n; i++)

{

total\_wt = total\_wt + wt[i];

total\_tat = 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 / (float)n);

System.out.println("Average turn around time = " +

(float)total\_tat / (float)n);

}

}

class Priority {

void priority(String processes[], int n, int burstTime[], int priority[]) {

int numberOfProcess = n;

int temp;

String temp2;

// Sorting process & burst time by priority

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

int TAT[] = new int[numberOfProcess + 1];

int waitingTime[] = new int[numberOfProcess + 1];

// Calculating Waiting Time & Turn Around Time

for (int i = 0; i < numberOfProcess; i++) {

TAT[i] = burstTime[i] + waitingTime[i];

waitingTime[i + 1] = TAT[i];

}

// WT = waiting Time

int totalWT = 0;

int totalTAT = 0;

double avgWT;

double avgTAT;

// Print Table

System.out.println("Process BT WT TAT");

for (int i = 0; i < numberOfProcess; i++) {

System.out.println(processes[i] + " " + burstTime[i] + " " + waitingTime[i] + " " + (TAT[i]));

totalTAT += (waitingTime[i] + burstTime[i]);

totalWT += waitingTime[i];

}

avgWT = totalWT / (double) numberOfProcess;

avgTAT = totalTAT / (double) numberOfProcess;

System.out.println("\n Average Wating Time: " + avgWT);

System.out.println(" Average Turn Around Time: " + avgTAT);

}

}

**All.java**

import java.util.Scanner;

public class All {

public static void main(String[] args){

FCFS fcfs = new FCFS();

SJF sjf = new SJF();

RR rr = new RR();

Priority pr = new Priority();

Scanner scan = new Scanner(System.in);

while(true){

System.out.println("\n The available algorithms are: ");

System.out.println("1. FCFS");

System.out.println("2. SJF");

System.out.println("3. RR");

System.out.println("4. Priority");

System.out.println("5. Exit");

System.out.println("Choose your algorithm: ");

int algo = scan.nextInt();

if (algo==1){

System.out.println("Enter the number of processes: ");

int n = scan.nextInt();

int[] processes = new int[n];

//Burst time of all processes

int[] burst\_time = new int[n];

System.out.println("Enter the processes: ");

for (int i = 0; i < n; i++){

processes[i] = scan.nextInt();

}

System.out.println("Enter the Burst time for he processes: ");

for (int i = 0; i < n; i++){

burst\_time[i] = scan.nextInt();

}

fcfs.findavgTime(processes, n, burst\_time);

}

else if(algo==2){

System.out.println("Enter the number of processes: ");

int n = scan.nextInt();

int[] processes = new int[n];

int[] burst\_time = new int[n];

int[] arr\_time = new int[n];

System.out.println("Enter the processes: ");

for(int i = 0; i < n; i++){

processes[i] = scan.nextInt();

}

System.out.println("Enter the Burst Time of the processes: ");

for(int i = 0; i < n; i++){

burst\_time[i] = scan.nextInt();

}

System.out.println("Enter the Arrival Time of the processes: ");

for(int i = 0; i < n; i++){

arr\_time[i] = scan.nextInt();

}

Process[] proc = new Process[n];

for(int i = 0; i < n; i++){

Process ind\_pr = new Process(processes[i], burst\_time[i],

arr\_time[i]);

proc[i] = ind\_pr;

}

sjf.findavgTime(proc, proc.length);

}

else if(algo==3){

int quantum;

System.out.println("Enter the number of processes: ");

int n = scan.nextInt();

int[] processes = new int[n];

//Burst time of all processes

int[] burst\_time = new int[n];

System.out.println("Enter the processes: ");

for (int i = 0; i < n; i++){

processes[i] = scan.nextInt();

}

System.out.println("Enter the Burst time for the processes: ");

for (int i = 0; i < n; i++){

burst\_time[i] = scan.nextInt();

}

quantum = scan.nextInt();

rr.findavgTime(processes, n, burst\_time, quantum);

}

else if(algo==4){

System.out.println("Enter the number of processes: ");

int n = scan.nextInt();

String processes[] = new String[n];

int burstTime[] = new int[n];

int priority[] = new int[n];

int p = 1;

for (int i = 0; i < n; i++) {

processes[i] = "P" + p;

p++;

}

System.out.print("Enter the Burst time for the processes: ");

for (int i = 0; i < n; i++) {

burstTime[i] = scan.nextInt();

}

System.out.print("Enter Priority for the processes: ");

for (int i = 0; i < n; i++) {

priority[i] = scan.nextInt();

}

pr.priority(processes, n, burstTime, priority);

}

else if(algo==5){

System.out.println("Exiting the code...");

break;

}

else{

System.out.println("Invalid Input");

}

}

scan.close();

}

}

**Output:**

The available algorithms are:

1. FCFS

2. SJF

3. RR

4. Priority

5. Exit

Choose your algorithm:

1

Enter the number of processes:

5

Enter the processes:

2

1

5

7

6

Enter the Burst time for he processes:

5

1

3

6

7

Processes Burst time Waiting time Turn around time

1 5 0 5

2 1 5 6

3 3 6 9

4 6 9 15

5 7 15 22

Average waiting time = 7.000000

Average turn around time = 11

The available algorithms are:

1. FCFS

2. SJF

3. RR

4. Priority

5. Exit

Choose your algorithm:

2

Enter the number of processes:

4

Enter the processes:

2

5

4

6

Enter the Burst Time of the processes:

4

9

7

8

Enter the Arrival Time of the processes:

6

5

3

2

Processes Burst time Waiting time Turn around time

2 4 4 8

5 9 16 25

4 7 11 18

6 8 0 8

Average waiting time = 7.75

Average turn around time = 14.75

The available algorithms are:

1. FCFS

2. SJF

3. RR

4. Priority

5. Exit

Choose your algorithm:

3

Enter the number of processes:

4

Enter the processes:

7

8

9

6

Enter the Burst time for the processes:

2

3

6

4

4

Processes Burst time Waiting time Turn around time

1 2 0 2

2 3 2 5

3 6 9 15

4 4 9 13

Average waiting time = 5.0

Average turn around time = 8.75

The available algorithms are:

1. FCFS

2. SJF

3. RR

4. Priority

5. Exit

Choose your algorithm:

4

Enter the number of processes:

4

Enter the Burst time for the processes: 1

7

5

32

Enter Priority for the processes: 4

1

2

5

Process BT WT TAT

P2 7 0 7

P3 5 7 12

P1 1 12 13

P4 32 13 45

Average Wating Time: 8.0

Average Turn Around Time: 19.25

**Assignment 4**

**Input:**

**PageReplacement.java**

import java.io.\*;

import java.util.Scanner;

class PageReplacement

{

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

do{

System.out.println("Menu");

System.out.println("1.FIFO");

System.out.println("2.Optimal Page Replacement");

System.out.println("3.Least Recently Used ");

System.out.println("4.EXIT");

System.out.println("ENTER YOUR CHOICE: ");

ch=Integer.parseInt(obj.readLine());

switch(ch)

{

case 1:

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());

do{

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;

}

}

if(flag)

{

frame[pt]=page;

pt++;

if(pt==f)

pt=0;

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();

}

chn++;

}

}while(chn!=n);

System.out.println("Page fault:"+pgf);

break;

case 2:

Scanner in = new Scanner(System.in);

int frames = 0;

int pointer = 0;

int numFault = 0;

int ref\_len;

boolean isFull = false;

int buffer[];

boolean hit[];

int fault[];

int reference[];

int mem\_layout[][];

System.out.println("Please enter the number of frames: ");

frames = Integer.parseInt(in.nextLine());

System.out.println("Please enter the length of the reference string: ");

ref\_len = Integer.parseInt(in.nextLine());

reference = new int[ref\_len];

mem\_layout = new int[ref\_len][frames];

buffer = new int[frames];

hit = new boolean[ref\_len];

fault = new int[ref\_len];

for(int j = 0; j < frames; j++)

{

buffer[j] = -1;

}

System.out.println("Please enter the reference string (hit Enter/Return after each number in the string): ");

for(int i = 0; i < ref\_len; i++)

{

reference[i] = Integer.parseInt(in.nextLine());

}

System.out.println();

for(int i = 0; i < ref\_len; i++)

{

int search = -1;

for(int j = 0; j < frames; j++)

{

if(buffer[j] == reference[i])

{

search = j;

hit[i] = true;

fault[i] = numFault;

break;

}

}

if(search == -1)

{

if(isFull)

{

int index[] = new int[frames];

boolean index\_flag[] = new boolean[frames];

for(int j = i + 1; j < ref\_len; j++)

{

for(int k = 0; k < frames; k++)

{

if((reference[j] == buffer[k]) && (index\_flag[k] == false))

{

index[k] = j;

index\_flag[k] = true;

break;

}

}

}

int max = index[0];

pointer = 0;

if(max == 0)

{

max = 200;

}

for(int j = 0; j < frames; j++)

{

if(index[j] == 0)

{

index[j] = 200;

}

if(index[j] > max)

{

max = index[j];

pointer = j;

}

}

}

buffer[pointer] = reference[i];

numFault++;

fault[i] = numFault;

if(!isFull)

{

pointer++;

if(pointer == frames)

{

pointer = 0;

isFull = true;

}

}

}

for(int j = 0; j < frames; j++)

{

mem\_layout[i][j] = buffer[j];

}

}

for(int i = 0; i < ref\_len; i++)

{

System.out.print(reference[i] + ": Memory is: ");

for(int j = 0; j < frames; j++)

{

if (mem\_layout[i][j] == -1)

{

System.out.printf("%3s ", "\*");

} else

{

System.out.printf("%3d ", mem\_layout[i][j]);

}

}

System.out.print(": ");

if (hit[i]) {

System.out.print("Hit");

} else

{

System.out.print("Page Fault");

}

System.out.print(": (Number of Page Faults: " + fault[i] + ")");

System.out.println();

}

System.out.println("Total Number of Page Faults: " + numFault);

break;

case 3:

int k=0;

System.out.println("enter no. of frames: ");

f=Integer.parseInt(obj.readLine());

int frame1[]=new int[f];

int a[]=new int[f];

int b[]=new int[f];

for(int i=0;i<f;i++)

{

frame1[i]=-1;

a[i]=-1;

b[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());

do{

int pg=0;

for(pg=0;pg<n;pg++)

{

page=pages[pg];

flag=true;

for(int j=0;j<f;j++)

{

if(page==frame1[j])

{flag=false; break;}

}

for(int j=0;j<f && flag;j++)

{

if(frame1[j]==a[f-1])

{k=j;

break;}

}

if(flag)

{

frame1[k]=page;

System.out.println("frame :" );

for(int j=0;j<f;j++)

System.out.print(frame1[j]+" ");

pgf++;

System.out.println();

}

else

{

System.out.println("frame :" );

for(int j=0;j<f;j++)

System.out.print(frame1[j]+" ");

System.out.println();

}

int p=1;

b[0]=page;

for(int j=0;j<a.length;j++)

{

if(page!=a[j] && p<f)

{

b[p]=a[j];

p++;

}

}

for(int j=0;j<f;j++)

{

a[j]=b[j];

}

chn++;

}

}while(chn!=n);

System.out.println("Page fault:"+pgf);

break;

case 4:

break;

}

} while(ch!=3);

}

}

**Output:**

Menu

1.FIFO

2.Optimal Page Replacement

3.Least Recently Used

4.EXIT

ENTER YOUR CHOICE:

1

enter no. of frames:

4

enter the no of pages

20

enter the page no

7 0 1 2 0 3 0 4 2 3 0 3 2 1 20 1 7 0 1

frame :7 -1 -1 -1

frame :7 0 -1 -1

frame :7 0 1 -1

frame :7 0 1 2

frame :7 0 1 2

frame :3 0 1 2

frame :3 0 1 2

frame :3 4 1 2

frame :3 4 1 2

frame :3 4 1 2

frame :3 4 0 2

frame :3 4 0 2

frame :3 4 0 2

frame :3 4 0 1

frame :2 4 0 1

frame :2 4 0 1

frame :2 4 0 1

frame :2 7 0 1

frame :2 7 0 1

frame :2 7 0 1

Page fault:10

Menu

1.FIFO

2.Optimal Page Replacement

3.Least Recently Used

4.EXIT

ENTER YOUR CHOICE:

2

Please enter the number of frames:

4

Please enter the length of the reference string:

20

Please enter the reference string (hit Enter/Return after each number in the string):

7 0 1 2 0 3 0 4 2 3 0 3 2 1 20 1 7 0 1

7: Memory is: 7 \* \* \* : Page Fault: (Number of Page Faults: 1)

0: Memory is: 7 0 \* \* : Page Fault: (Number of Page Faults: 2)

1: Memory is: 7 0 1 \* : Page Fault: (Number of Page Faults: 3)

2: Memory is: 7 0 1 2 : Page Fault: (Number of Page Faults: 4)

0: Memory is: 7 0 1 2 : Hit: (Number of Page Faults: 4)

3: Memory is: 3 0 1 2 : Page Fault: (Number of Page Faults: 5)

0: Memory is: 3 0 1 2 : Hit: (Number of Page Faults: 5)

4: Memory is: 3 0 4 2 : Page Fault: (Number of Page Faults: 6)

2: Memory is: 3 0 4 2 : Hit: (Number of Page Faults: 6)

3: Memory is: 3 0 4 2 : Hit: (Number of Page Faults: 6)

0: Memory is: 3 0 4 2 : Hit: (Number of Page Faults: 6)

3: Memory is: 3 0 4 2 : Hit: (Number of Page Faults: 6)

2: Memory is: 3 0 4 2 : Hit: (Number of Page Faults: 6)

1: Memory is: 1 0 4 2 : Page Fault: (Number of Page Faults: 7)

2: Memory is: 1 0 4 2 : Hit: (Number of Page Faults: 7)

0: Memory is: 1 0 4 2 : Hit: (Number of Page Faults: 7)

1: Memory is: 1 0 4 2 : Hit: (Number of Page Faults: 7)

7: Memory is: 1 0 7 2 : Page Fault: (Number of Page Faults: 8)

0: Memory is: 1 0 7 2 : Hit: (Number of Page Faults: 8)

1: Memory is: 1 0 7 2 : Hit: (Number of Page Faults: 8)

Total Number of Page Faults: 8

Menu

1.FIFO

2.Optimal Page Replacement

3.Least Recently Used

4.EXIT

ENTER YOUR CHOICE:

3

enter no. of frames:

4

enter the no of pages

20

enter the page no

7 0 1 2 0 3 0 4 2 3 0 3 2 1 20 1 7 0 1

frame :

7 -1 -1 -1

frame :

7 0 -1 -1

frame :

7 0 1 -1

frame :

7 0 1 2

frame :

7 0 1 2

frame :

3 0 1 2

frame :

3 0 1 2

frame :

3 0 4 2

frame :

3 0 4 2

frame :

3 0 4 2

frame :

3 0 4 2

frame :

3 0 4 2

frame :

3 0 4 2

frame :

3 0 1 2

frame :

3 0 1 2

frame :

3 0 1 2

frame :

3 0 1 2

frame :

7 0 1 2

frame :

7 0 1 2

frame :

7 0 1 2

Page fault:8

**Assignment 1**

**Input:**

import java.io.BufferedInputStream;

import java.io.DataInputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

public class Server {

//initialize socket and input stream

private Socket socket = null;

private ServerSocket server = null;

private DataInputStream in = null;

//constructor with port no.

public Server(int port){

// start server and waits for connection

// starts server and waits for a connection

try

{

server = new ServerSocket(port);

System.out.println("Server started");

System.out.println("Waiting for a client ...");

socket = server.accept();

System.out.println("Client accepted");

// takes input from the client socket

in = new DataInputStream(

new BufferedInputStream(socket.getInputStream()));

String line = "";

// reads message from client until "Over" is sent

while (!line.equals("Over"))

{

try

{

line = in.readUTF();

System.out.println(line);

}

catch(IOException i)

{

System.out.println(i);

}

}

System.out.println("Closing connection");

// close connection

socket.close();

in.close();

}

catch(IOException i)

{

System.out.println(i);

}

}

public static void main(String args[])

{

Server server = new Server(5000);

}

}

// Client

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

import java.net.UnknownHostException;

public class Client {

private Socket socket = null;

private DataInputStream input = null;

private DataOutputStream out = null;

// constructor to put ip address and port

public Client(String address, int port)

{

try{

socket = new Socket(address, port);

System.out.println("Connected");

// takes input from terminal

input = new DataInputStream(System.in);

// sends output to the socket

out = new DataOutputStream(socket.getOutputStream());

}

catch (UnknownHostException u)

{

System.out.println(u);

} catch (IOException e) {

System.out.println(e);

}

// string to read message from input

String line = "";

// keep reading until "Over" is input

while (!line.equals("Over")){

try {

line = input.readLine();

out.writeUTF(line);

} catch (IOException e) {

System.out.println(e);

}

}

//close the connection

try {

input.close();

out.close();

socket.close();

} catch (IOException e) {

System.out.println(e);

}

}

public static void main(String args[]){

Client client = new Client("127.0.0.1",5000);

}

}

**Output:**

Server started

Waiting for a client ...

Client accepted

Hiiii

I am Client!

Bye

Over

Closing connection

Connected

Hiiii

I am Client!

Bye

Over

**Assignment 2**

**Input:**

RPC/add\_server.c

/\*

\* This is sample code generated by rpcgen.

\* These are only templates and you can use them

\* as a guideline for developing your own functions.

\*/

#include "add.h"

int \*

add\_1\_svc(numbers \*argp, struct svc\_req \*rqstp)

{

static int result;

printf("add(%d, %d) is called\n",argp->a, argp->b);

result = argp->a + argp->b;

return &result;

}

RPC/add\_client.c

/\*

\* This is sample code generated by rpcgen.

\* These are only templates and you can use them

\* as a guideline for developing your own functions.

\*/

#include "add.h"

void

add\_prog\_1(char \*host, int x, int y)

{

CLIENT \*clnt;

int \*result\_1;

numbers add\_1\_arg;

#ifndef DEBUG

clnt = clnt\_create (host, ADD\_PROG, ADD\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

#endif /\* DEBUG \*/

add\_1\_arg.a=x;

add\_1\_arg.b=y;

result\_1 = add\_1(&add\_1\_arg, clnt);

if (result\_1 == (int \*) NULL) {

clnt\_perror (clnt, "call failed");

}

else

{

printf("Result:%d\n",\*result\_1);

}

#ifndef DEBUG

clnt\_destroy (clnt);

#endif /\* DEBUG \*/

}

int

main (int argc, char \*argv[])

{

char \*host;

if (argc < 4 ) {

printf ("usage: %s server\_host NUMBER NUMBER\n", argv[0]);

exit (1);

}

host = argv[1];

add\_prog\_1 (host, atoi(argv[2]), atoi(argv[3]));

exit (0);

}

RPC/add\_svc.c

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include "add.h"

#include <stdio.h>

#include <stdlib.h>

#include <rpc/pmap\_clnt.h>

#include <string.h>

#include <memory.h>

#include <sys/socket.h>

#include <netinet/in.h>

#ifndef SIG\_PF

#define SIG\_PF void(\*)(int)

#endif

static void

add\_prog\_1(struct svc\_req \*rqstp, register SVCXPRT \*transp)

{

union {

numbers add\_1\_arg;

} argument;

char \*result;

xdrproc\_t \_xdr\_argument, \_xdr\_result;

char \*(\*local)(char \*, struct svc\_req \*);

switch (rqstp->rq\_proc) {

case NULLPROC:

(void) svc\_sendreply (transp, (xdrproc\_t) xdr\_void, (char \*)NULL);

return;

case add:

\_xdr\_argument = (xdrproc\_t) xdr\_numbers;

\_xdr\_result = (xdrproc\_t) xdr\_int;

local = (char \*(\*)(char \*, struct svc\_req \*)) add\_1\_svc;

break;

default:

svcerr\_noproc (transp);

return;

}

memset ((char \*)&argument, 0, sizeof (argument));

if (!svc\_getargs (transp, (xdrproc\_t) \_xdr\_argument, (caddr\_t) &argument)) {

svcerr\_decode (transp);

return;

}

result = (\*local)((char \*)&argument, rqstp);

if (result != NULL && !svc\_sendreply(transp, (xdrproc\_t) \_xdr\_result, result)) {

svcerr\_systemerr (transp);

}

if (!svc\_freeargs (transp, (xdrproc\_t) \_xdr\_argument, (caddr\_t) &argument)) {

fprintf (stderr, "%s", "unable to free arguments");

exit (1);

}

return;

}

int

main (int argc, char \*\*argv)

{

register SVCXPRT \*transp;

pmap\_unset (ADD\_PROG, ADD\_VERS);

transp = svcudp\_create(RPC\_ANYSOCK);

if (transp == NULL) {

fprintf (stderr, "%s", "cannot create udp service.");

exit(1);

}

if (!svc\_register(transp, ADD\_PROG, ADD\_VERS, add\_prog\_1, IPPROTO\_UDP)) {

fprintf (stderr, "%s", "unable to register (ADD\_PROG, ADD\_VERS, udp).");

exit(1);

}

transp = svctcp\_create(RPC\_ANYSOCK, 0, 0);

if (transp == NULL) {

fprintf (stderr, "%s", "cannot create tcp service.");

exit(1);

}

if (!svc\_register(transp, ADD\_PROG, ADD\_VERS, add\_prog\_1, IPPROTO\_TCP)) {

fprintf (stderr, "%s", "unable to register (ADD\_PROG, ADD\_VERS, tcp).");

exit(1);

}

svc\_run ();

fprintf (stderr, "%s", "svc\_run returned");

exit (1);

/\* NOTREACHED \*/

}

RPC/add\_clnt.c

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include <memory.h> /\* for memset \*/

#include "add.h"

/\* Default timeout can be changed using clnt\_control() \*/

static struct timeval TIMEOUT = { 25, 0 };

int \*

add\_1(numbers \*argp, CLIENT \*clnt)

{

static int clnt\_res;

memset((char \*)&clnt\_res, 0, sizeof(clnt\_res));

if (clnt\_call (clnt, add,

(xdrproc\_t) xdr\_numbers, (caddr\_t) argp,

(xdrproc\_t) xdr\_int, (caddr\_t) &clnt\_res,

TIMEOUT) != RPC\_SUCCESS) {

return (NULL);

}

return (&clnt\_res);

}

**Output:**

Server:

v1ack@fedora: ~/RPC$ rpcgen -a -C add.x

v1ack@fedora: ~/RPC$ make -f Makefile.add

v1ack@fedora: ~/RPC$ sudo ./add\_server

add(50,20) is called

Client:

v1ack@fedora: ~$ cd RPC/

v1ack@fedora: ~/RPC$ sudo ./add\_client localhost 50 20

Result: 70

**Assignment 3**

**Input:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 20

int list[MAX],n,cdr;

void display()

{

   int i;

   printf("\nProcesses-->");

   for(i=1;i<=n;i++)

      printf("\t %d",i);

   printf("\nAlive-->");

   for(i=1;i<=n;i++)

      printf("\t %d",list[i]);

   printf("\ncoordinator is::%d",cdr);

}

void ring()

{

   int msg[20],ring\_n,k,i;

   int ch,crash,activate,gid,flag,subcdr;

   do

   {

         printf("\n1.Crash\n2.Activate\n3.Display\n4.Exit\nEnter You choice::");

         scanf("%d",&ch);

         switch(ch)

         {

            case 1:

               printf("\nEnter Process no. to Crash::");

               scanf("%d",&crash);

               if(list[crash])

                  list[crash]=0;

               else

               {

                  printf("\nProcess is alreaady dead!!");

                  break;

               }

               do

               {

                  printf("\nEnter election generator id::");

                  scanf("%d",&gid);

                  if(gid==cdr)

                  {

                     printf("\nenter a valid generator id::");

                  }

               }while(gid==crash);

               flag=0;

               k=1;

               if(crash==cdr)

               {

                  msg[k++]=gid;

                  for(i=(gid+1)%n;i!=gid;i=(i+1)%n)

                  {

                     if(list[i])

                     {

                        printf("\nmessage is sent to %d k =%d",i,k);

                        msg[k++]=i;

*//                      printf("Response is sent from %d to %d",i,gid);*

                     }

                  }

                  subcdr=0;

                  for(i=1;i<k;i++)

                  {

                     printf("msg::%d\n",msg[i]);

                     if(subcdr<msg[i])

                     {

                        subcdr=msg[i];

                     }

                  }

                  cdr=subcdr;

               }

               display();

               break;

            case 2:

*//activate*

               printf("\nEnter Process no. to Activated::");

               scanf("%d",&activate);

               if(!list[activate])

                  list[activate]=1;

                  else

               {

                  printf("\nProcess is alreaady alive!!");

                  break;

               }

*//*

               if(activate==n)

               {

                  cdr=n;

                  break;

               }

               for(i=activate+1;i<=n;i++)

               {

                  printf("\nmessage is sent from %d to %d",activate,i);

                  if(list[i])

                  {

                     subcdr=i;

                     printf("Response is sent from %d to %d",i,activate);

                     flag=1;

                  }

               }

               if(flag==1)

               {

                  cdr=subcdr;

               }

               else

               {

                  cdr=activate;

               }

            display();

*//*

            break;

         case 3:

            display();

         break;

         case 4:

            break;

         }

   }while(ch!=4);

}

void bully()

{

   int ch,crash,activate,i,gid,flag,subcdr;

   do

   {

         printf("\n1.Crash\n2.Activate\n3.Display\n4.Exit\nEnter You choice::");

         scanf("%d",&ch);

         switch(ch)

         {

            case 1:

               printf("\nEnter Process no. to Crash::");

               scanf("%d",&crash);

               if(list[crash])

                  list[crash]=0;

               else

               {

                  printf("\nProcess is alreaady dead!!");

                  break;

               }

               do

               {

                  printf("\nEnter election generator id::");

                  scanf("%d",&gid);

                  if(gid==cdr)

                  {

                     printf("\nenter a valid generator id::");

                  }

               }while(gid==crash);

               flag=0;

               if(crash==cdr)

               {

                  for(i=gid+1;i<=n;i++)

                  {

                     printf("\nmessage is sent from %d to %d",gid,i);

                     if(list[i])

                     {

                        subcdr=i;

                        printf("Response is sent from %d to %d",i,gid);

                        flag=1;

                     }

                  }

                  if(flag==1)

                  {

                     cdr=subcdr;

                  }

                  else

                  {

                     cdr=gid;

                  }

               }

               display();

               break;

            case 2:

*//activate*

               printf("\nEnter Process no. to Activated::");

               scanf("%d",&activate);

               if(!list[activate])

                  list[activate]=1;

               else

               {

                  printf("\nProcess is alreaady alive!!");

                  break;

               }

*//*

               if(activate==n)

               {

                  cdr=n;

                  break;

               }

               for(i=activate+1;i<=n;i++)

               {

                  printf("\nmessage is sent from %d to %d",activate,i);

                  if(list[i])

                  {

                     subcdr=i;

                     printf("Response is sent from %d to %d",i,activate);

                     flag=1;

                  }

               }

               if(flag==1)

               {

                  cdr=subcdr;

               }

               else

               {

                  cdr=activate;

               }

            display();

*//*

            break;

         case 3:

            display();

         break;

         case 4:

            break;

         }

   }while(ch!=4);

}

int main()

{

   int i,j;

   printf("\nEnter no. of process::");

   scanf("%d",&n);

   for(i=1;i<=n;i++)

   {

      printf("\nEnter Process %d is Alive or not(0/1)::",i);

      scanf("%d",&list[i]);

      if(list[i])

         cdr=i;

   }

   display();

   do

   {

      printf("\n1.BULLY ALGORITHM \n2.RING ALGORITHM\n3.Display\n4.EXIT\nEnter your choice::");

      scanf("%d",&j);

      switch(j)

      {

         case 1:

            bully();

            break;

         case 2:

            ring();

         case 3:

            display();

            break;

         case 4:

            exit(1);

      }

   }while(j!=4);

   return 0;

}

**Output:**

Enter no. of process::3

Enter Process 1 is Alive or not(0/1)::1

Enter Process 2 is Alive or not(0/1)::1

Enter Process 3 is Alive or not(0/1)::0

Processes--> 1 2 3

Alive--> 1 1 0

coordinator is::2

1.BULLY ALGORITHM

2.RING ALGORITHM

3.Display

4.EXIT

Enter your choice::1

1.Crash

2.Activate

3.Display

4.Exit

Enter You choice::2

Enter Process no. to Activated::3

1.Crash

2.Activate

3.Display

4.Exit

Enter You choice::3

Processes--> 1 2 3

Alive--> 1 1 1

coordinator is::3

1.Crash

2.Activate

3.Display

4.Exit

Enter You choice::4

1.BULLY ALGORITHM

2.RING ALGORITHM

3.Display

4.EXIT

Enter your choice::2

1.Crash

2.Activate

3.Display

4.Exit

Enter You choice::2

Enter Process no. to Activated::3

Process is alreaady alive!!

1.Crash

2.Activate

3.Display

4.Exit

Enter You choice::1

Enter Process no. to Crash::2

Enter election generator id::123

Processes--> 1 2 3

Alive--> 1 0 1

coordinator is::3

1.Crash

2.Activate

3.Display

4.Exit

Enter You choice::4

Processes--> 1 2 3

Alive--> 1 0 1

coordinator is::3

1.BULLY ALGORITHM

2.RING ALGORITHM

3.Display

4.EXIT

Enter your choice::3

Processes--> 1 2 3

Alive--> 1 0 1

coordinator is::3

**Assignment 4**

**Input:**

Lamport’s Logical Clock

#include <bits/stdc++.h>

using namespace std;

// Function to find the maximum timestamp

// between 2 events

int max1(int a, int b)

{

// Return the greatest of th two

if (a > b)

return a;

else

return b;

}

// Function to display the logical timestamp

void display(int e1, int e2,

int p1[5], int p2[3])

{

int i;

cout << "\nThe time stamps of "

"events in P1:\n";

for (i = 0; i < e1; i++) {

cout << p1[i] << " ";

}

cout << "\nThe time stamps of "

"events in P2:\n";

// Print the array p2[]

for (i = 0; i < e2; i++)

cout << p2[i] << " ";

}

// Function to find the timestamp of events

void lamportLogicalClock(int e1, int e2,

int m[5][3])

{

int i, j, k, p1[e1], p2[e2];

// Initialize p1[] and p2[]

for (i = 0; i < e1; i++)

p1[i] = i + 1;

for (i = 0; i < e2; i++)

p2[i] = i + 1;

cout << "\t";

for (i = 0; i < e2; i++)

cout << "\te2" << i + 1;

for (i = 0; i < e1; i++) {

cout << "\n e1" << i + 1 << "\t";

for (j = 0; j < e2; j++)

cout << m[i][j] << "\t";

}

for (i = 0; i < e1; i++) {

for (j = 0; j < e2; j++) {

// Change the timestamp if the

// message is sent

if (m[i][j] == 1) {

p2[j] = max1(p2[j], p1[i] + 1);

for (k = j + 1; k < e2; k++)

p2[k] = p2[k - 1] + 1;

}

// Change the timestamp if the

// message is received

if (m[i][j] == -1) {

p1[i] = max1(p1[i], p2[j] + 1);

for (k = i + 1; k < e1; k++)

p1[k] = p1[k - 1] + 1;

}

}

}

// Function Call

display(e1, e2, p1, p2);

}

// Driver Code

int main()

{

int e1 = 5, e2 = 3, m[5][3];

// message is sent and received

// between two process

/\*dep[i][j] = 1, if message is sent

from ei to ej

dep[i][j] = -1, if message is received

by ei from ej

dep[i][j] = 0, otherwise\*/

m[0][0] = 0;

m[0][1] = 0;

m[0][2] = 0;

m[1][0] = 0;

m[1][1] = 0;

m[1][2] = 1;

m[2][0] = 0;

m[2][1] = 0;

m[2][2] = 0;

m[3][0] = 0;

m[3][1] = 0;

m[3][2] = 0;

m[4][0] = 0;

m[4][1] = -1;

m[4][2] = 0;

// Function Call

lamportLogicalClock(e1, e2, m);

return 0;

}

Output:

e21 e22 e23

e11 0 0 0

e12 0 0 1

e13 0 0 0

e14 0 0 0

e15 0 -1 0

The time stamps of events in P1:

1 2 3 4 5

The time stamps of events in P2:

1 2 3