Navsahyadri Group of Institutions
Faculty of Engineering, PUNE

DEPARTMENT OF COMPUTER ENGINEERING

LAB MANUAL

LP-1

(System Programming and Operating System)

Academic Year 2022-23

T.E. COMPUTER (SEM — I)

Prepared By - Prof. R. R. Bhuvad

INDEX

ASSN NO.	TITLE
Part I: Systems Programming and Operating System	
Group A	
1	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. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.
2	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.
Group B	
3	Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).
4	Write a program to simulate Page replacement algorithm.

Prepared By - Prof. R. R. Bhuvad

GROUP - A
EXPERIMENT NO: 01

Aim: 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. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.

Objectives:

- To understand data structures to be used in pass I of an assembler.
- To implement pass I of an assembler

Problem Statement: Write a program to create pass-I Assembler.

Outcomes: After completion of this assignment students will be able to:

- Understand the concept of Pass-I Assembler
- Understand the Programming language of Java.

Software Requirements: Windows OS, JDK1.7

Theory Concepts:

Assembly language :

A low-level programming language in which a mnemonic is used to represent each of the machine language instructions for a particular computer. An assembly language is machine dependent. It differs from computer to computer. Writing programs in assembly language is very easy as compared to machine(binary) language.

Assembly Language Program (ALP):

Assembly Language is a kind of low level programming language, which uses symbolic codes or mnemonics as instructions. Some examples of mnemonics are ADD,SUB,MOVER etc. For processing of an assembly language program we need a language translator called as Assembler.

Assembler:

Assembler is a translator which translates assembly language program into machine language. An assembly language program can be translated into machine language.

It involves following steps:

1. Find addresses of variable.
2. Replace symbolic addresses by numeric addresses.
3. Replace symbolic opcode by machine operation codes.
4. Reserve storage for data.

Language translator:

A language translator bridges an execution gap to machine language of computer system. An assembler is a language translator whose source language is assembly language. An Assembler is a program that accepts as input an Assembly language program and converts it into machine language.

Language processing activity consists of two phases,

1. Analysis phase
2. Synthesis phase.

Analysis of source program consists of three components, Lexical rules, syntax rules and semantic rules. Lexical rules govern the formation of valid statements in source language. Semantic rules associate the formation meaning with valid statements of language.

Synthesis phase is concerned with construction of target language statements, which have the same meaning as source language statements. This consists of memory allocation and code generation.

TWO PASS TRANSLATION SCHEME:

In a 2-pass assembler, the first pass constructs an intermediate representation of the source program for use by the second pass.

This representation consists of two main components - data structures like Symbol table, Literal table and processed form of the source program called as intermediate code(IC).

This intermediate code is represented by the syntax of Variant –I.

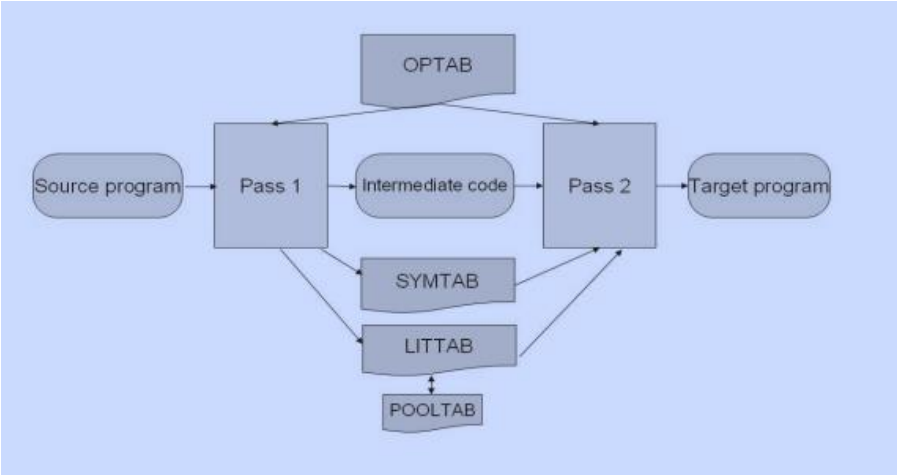
Forward reference of a program entity is a reference to the entity, which precedes its definition in the program.

While processing a statement containing a forward reference, language processor does not posses all relevant information concerning referenced entity.

This creates difficulties in synthesizing the equivalent target statements. This problem can be solved by postponing the generation of target code until more information concerning the entity is available.

This also reduces memory requirements of LP and simplifies its organization. This leads to multi-pass model of language processing.

DATA STRUCTURES OF A TWO PASS ASSEMBLER:



Data Structure of Assembler:

a) **Operation code table (OPTAB) :**This is used for storing mnemonic, operation code and class of instruction Structure of OPTAB is as follows.

b) **Data structure updated during translation**: Also called as translation time data structure.

They are

I. SYMBOL TABLE (SYMTAB) : Ii contains entries such as symbol,it's address and value. SYMBOL TABLE have following fields

Name of symbol	Symbol Address	Value

II. LITERAL TABLE (LITTAB) : it contains entries such as literal and it's value. Literal Table has following fields

literal	**Address of Literal**

III . POOL TABLE (POOLTAB): Contains literal number of the starting literal of each literal pool. Pool TABLE (pooltab) have following fields.

LITERAL_NO

Design of a Two Pass Assembler: -

Tasks performed by the passes of two-pass assembler are as follows:

Pass I: -

- Separate the symbol, mnemonic opcode and operand fields.
- Determine the storage-required for every assembly language statement and update the location counter.
- Build the symbol table and the literal table.
- Construct the intermediate code for every assembly language statement.

Pass II: -

- Synthesize the target code by processing the intermediate code generated during pass1.
- Data Structure used by Pass II:
  1.OPTAB: A table of mnemonic opcodes and related information.
  2.SYMTAB: The symbol table
  3.POOL_TAB and LITTAB: A table of literals used in the program
  4.Intermediate code generated by Pass I
  5.Output file containing Target code / error listing.
- INTERMEDIATE CODE REPRESENTATION
  The intermediate code consists of a set of IC units, each IC unit consisting of the following three fields
  1. Address
  2. Representation of the mnemonic opcode
  3. Representation of operands
- Where statement class can be one of IS,DL and AD standing for imperative statement, declaration statement and assembler directive , respectively.

- For imperative statement, code is the instruction opcode in the machine language For declaration and assembler directives , following are the codes

Declaration Statements		Assembler directives	
DC	01	START	01
DS	02	END	02
		ORIGIN	03
		EQU	04
		LTORG	05

- Mnemonic Operation Codes :

Instruction Opcode	Assembly Mnemonic	Remarks
00	**STOP**	**STOP EXECUTION**
01	**ADD**	**FIRST OPERAND IS MODIFIED CONDITION CODE IS SET**
02	**SUB**	**FIRST OPERAND IS MODIFIED CONDITION CODE IS SET**
03	**MULT**	**FIRST OPERAND IS MODIFIED CONDITION CODE IS SET**
04	**MOVER**	**REGISTER ← MEMORY MOVE**
05	**MOVEM**	**MEMORY MOVE → REGISTER MOVE**
06	**COMP**	**SETS CONDITION CODE**
07	**BC**	**BRANCH ON CONDITION**
08	**DIV**	**ANALOGOUS TO SUB**
09	**READ**	**FIRST OPERAND IS NOT USED**
10	**PRINT**	**FIRST OPERAND IS NOT USED**

Branch On Condition : BC Transfer Control to the Memory word With Address < memory address>

Condition code	Opcode
LT	01
LE	02
EQ	03
GT	04
GE	05
ANY	06

Algorithms(procedure) :

PASS 1

1. Initialize location counter, entries of all tables as zero.
2. Read statements from input file one by one.
3. While next statement is not END statement

I. Tokenize or separate out input statement as label,numonic,operand1,operand2 II. If label is present insert label into symbol table.
III. If the statement is LTORG statement processes it by making it's entry into literal table, pool table and allocate memory.
IV. If statement is START or ORIGEN Process location counter accordingly.
V. If an EQU statement, assign value to symbol by correcting entry in symbol table.
VI. For declarative statement update code, size and location counter.
VII. Generate intermediate code.
VIII. Pass this intermediate code to pass -2.

PASS 2

1.code_area_address=address of code area;
Pooltab_ptr:=1;
loc_cntr=0;
2.While next statement is not an END statement
a) clear the machine_code_buffer
b) if an LTORG statement
I) process literals in LITTAB[POOLTAB[pooltab_ptr]]…
LITTAB[POOLTAB[pooltab_ptr+1]]-1 similar to processing of constants in a dc statement.
II) size=size of memory area required for literals
III) pooltab_ptr=pooltab_ptr+1
c) if a START or ORIGIN statement then
I) loc_cntr = value specified in operand field
II) size=0;
d) if a declaration statement
I) if a DC statement then assemble the constant in machine_code_buffer
II) size=size of memory area required by DC or DS:
e) if an imperative statement then
I) get operand address from SYMTAB or LITTAB
II) Assemble instruction in machine code buffer.
III) size=size of instruction;
f) if size # 0 then
I) move contents of machine_code_buffer to the address code_area_address+loc_cntr ;
II) loc_cntr=loc_cntr+size;
3. (Processing of END statement)

Conclusion: Thus, we have successfully implemented Pass-I and Pass-II of a two-pass assembler.

Pass-I Program-

```
class symtab
{
	int index;
	String name;
	int addr;
	symtab(int i,String s,int a)
	{
		index = i;
		name = s;
		addr = a;
	}
}
class littab
{
	int index;
	String name;
	int addr;
	 littab(int i,String s,int a)
	 {
		index = i;
		name = s;
		addr = a;
	 }
	 void setaddr(int a)
	 {
		 addr = a;
	 }
}
class pooltab
{
	int p_index;
	int l_index;

	pooltab(int i,int a)
	{
		p_index = i;
		l_index = a;
	}
}
public class pass1
{
	public static void main(String args[])
	{
		String input[][]={{null,"START","100",null},{null,"MOVER","AREG","A"},

	{"AGAIN","ADD","AREG","='2'"},{null,"ADD","AREG","B"},

	{"AGAIN","ADD","AREG","='3'"},{null,"LTORG",null,null},
```

```
                                        {"AGAIN2","ADD","AREG","BREG"},
                                        {"AGAIN2","ADD","AREG","CREG"},

{"AGAIN","ADD","AREG","='2'"},{null,"DC","B","3"},
                                        {"LOOP","DS","A","1"},{null,"END",null,null}};

        symtab s[]=new symtab[20];
        littab l[] = new littab[20];
        pooltab p[] = new pooltab[20];

        int loc=0,i=0;
        String m,op1,op2;
        int sn=0,ln=0,lnc=0,pn=0;

        loc = Integer.parseInt(input[0][2]);
        m=input[1][1];
        i=1;
        while(!m.equals("END"))
        {
                if(check(m)==1)
                {
                        if (input[i][0]==null)
                        {
                                op1 =input[i][2];
                                op2 = input[i][3];
                                if(comp(op2,s,sn)==1)
                                {
                                        s[sn]=new symtab(sn,op2,0);
                                        sn++;
                                }
                                else if(comp(op2,s,sn)==2)
                                {
                                        l[ln]=new littab(ln,op2,0);
                                        ln++;
                                }
                                loc++;
                                i++;
                        }
                        else
                        {
                           op1 = input[i][0];
                                s[sn]=new symtab(sn,op1,loc);
                                sn++;
                                op1=input[i][2];
                                op2=input[i][3];
                                if(comp(op2,s,sn)==1)
                                {
                                        s[sn]=new symtab(sn,op2,0);
                                        sn++;
                                }
                                else if(comp(op2,s,sn)==2)
```

```
                {
                        l[ln]= new littab(ln,op2,0);
                        ln++;
                }
                loc++;
                i++;
        }
}
else if(check(m)== 2)
        {
                if(input[i][0] == null)
                {
                        int temp;
                        op1 = input[i][2];
                        op2 = input[i][3];
                        temp=comps(op1,s,sn);
                        if(temp!=99)
                        {
                                s[temp]=new symtab(temp,op1,loc);
                        }
                        loc=loc+Integer.parseInt(op2);
                        i++;
                }
                else
                {
                        int temp;
                        op1=input[i][0];
                        s[sn]= new symtab(sn,op1,loc);
                        sn++;
                        op1=input[i][2];
                        op2=input[i][3];
                        temp = comps(op1,s,sn);
                        if(temp!=99)
                        {
                                s[temp] = new symtab(temp,op1,loc);
                        }
                        loc= loc+Integer.parseInt(op2);
                        i++;
                }
        }
        else if(check(m)== 3)
        {
                if(input[i][0] == null)
                {
                        int temp;
                        op1 = input[i][2];
                        op2 = input[i][3];
                        temp = comps(op1,s,sn);
                        if(temp!=99)
                        {
                                s[temp]=new symtab(temp,op1,loc);
```

```
                                        }
                                        loc++;
                                        i++;
                                }
                                else
                                {
                                        int temp;
                                        op1 = input[i][0];
                                        s[sn]=new symtab(sn,op1,loc);
                                        sn++;
                                        op1 = input[i][2];
                                        op2 =input[i][3];
                                        temp= comps(op1,s,sn);
                                        if(temp!=99)
                                        {
                                                s[temp]= new symtab(temp,op1,loc);
                                        }
                                        loc++;
                                        i++;
                                }
                        }
                        else if(check(m)==4)
                        {
                                if(lnc !=ln)
                                {
                                        p[pn] = new pooltab(pn,lnc);
                                        pn++;
                                }
                                while(lnc !=ln)
                                {
                                        l[lnc].setaddr(loc);
                                        lnc++;
                                        loc++;
                                }
                                i++;
                   }
                        m = input[i][1];
        }
                if(lnc !=ln)
                {
                        p[pn]=new pooltab(pn,lnc);
                        pn++;
                }
                while(lnc!=ln)
                {
                        l[lnc].setaddr(loc);
                        lnc++;
                        loc++;
                }
                System.out.print("Symbol Table\nIndex\tSymbol\tAddress\n");
for(i=0;i<sn;i++)
```

```
		{
			System.out.println(s[i].index+"\t"+s[i].name+""+s[i].addr);
		}
		System.out.print("\nLiteralTable\nIndex\tLiteral\tAddress\n");
		for(i=0;i<ln;i++)
		{
			System.out.println(l[i].index+"\t"+l[i].name+"\t"+s[i].addr);
		}
		System.out.print("\nPool Table\nPool Index\tLiteral Index\n");

		for(i=0;i<pn;i++)
		{
			System.out.println("\t"+p[i].p_index+"\t\t"+p[i].l_index);
		}
		System.out.print("\n\n Intermediate Code \n");
		i = 0;
		m = input[i][1];
		op1 = input[i][2];
	op2 = input[i][3];
		int point=0,in1,in2,j=0;
		System.out.print(ic(m)+ic(op1));
		while(!m.equals("END"))
		{
			if(check(m)== 1)
			{
				System.out.print(ic(m)+ic(op1));
				if(comp(op2,s,sn)==0&&comps(op2,s,sn)==99)
				{
					System.out.print(ic(op2));
				}
				else if(comp(op2,s,sn)== 2)
				{
					int temp;
					temp = compl(op2,l,ln,j);
					System.out.print("(L,"+temp+")");
					j++;
				}
				else if(comp(op2,s,sn)!=1)
				{
					int temp;
					temp=comps(op2,s,sn);
					System.out.print("(S,"+temp+")");
				}
				else if(check(m) ==2||check(m)==3)
				{
					System.out.print(ic(m)+ic(op2));
						if(comp(op1,s,sn)!=1)
						{
							int temp;
							temp=
comps(op1,s,sn);System.out.print("(S,"+temp+")");
```

```
                                        }
                                }
                        }
                                else if(check(m)==4)
                                {
                                        if(point+1!=pn)
                                        {
                                                in1=p[point+1].l_index-p[point].l_index;
                                                in2=p[point].l_index;
                                                point++;
                                                while(in1>0)
                                                {

System.out.print(ic(m)+ic(l[in2].name));
                                                        in2++;
                                                        in1--;
                                                        System.out.print("\n");
                                                }
                                        }
                                        else
                                        {
                                                in2 =p[point].l_index;
                                                while(in2!=ln)
                                                {

System.out.print(ic(m)+ic(l[in2].name));
                                                        in2++;
                                                        System.out.print("\n");
                                                }
                                        }
                                }
                                        i++;
                                        m= input[i][1];
                                        op1 = input[i][2];
                                        op2 = input[i][3];
                                        System.out.print("\n");
                        }
                                System.out.print(ic(m));
                                m = "LTORG";
                                if(point+1!=pn)
                                {
                                        in1=p[point+1].l_index-p[point].l_index;
                                        in2=p[point].l_index;
                                        point++;
                                        while(in1>0)
                                        {
                                                System.out.println(ic(m)+ic(l[in2].name));
                                                in2++;
                                                in1--;
                                        }
                                }
```

```
                else
                {
                        in2=p[point].l_index;
                        while(in2!=ln)
                        {
                                System.out.print(ic(m)+ic(l[in2].name));
                                in2++;
                        }
                }
        }

                public static int check(String m)
                {
                        if(m.equals("MOVER")||m.equals("ADD"))
                        {
                                return 1;
                        }
                        else if(m.equals("DS"))
                        {
                                return 2;
                        }
                        else if(m.equals("DC"))
                        {
                                return 3;
                        }
                        else if(m.equals("LTORG"))
                        {
                                return 4;
                        }
                        else
                        {
                         return -1;
                        }
                }
                        public static int comp(String m,symtab s[],int sn)
                        {

if(m.equals("AREG")||m.equals("BREG")||m.equals("CREG"))
                                return 0;
                        else if(m.toCharArray()[0]=='=')
                                return 2;
                        else if(comps(m,s,sn) == 99)
                                return 1;
                        else
                                return 0;
                        }
                        public static int compl(String m,littab l[],int ln,int j)
                        {
                                int i;
                                for(i=j;i<ln;i++)
                                {
```

```
                if(m.equals(l[i].name))
                        return l[i].index;
        }
        return 99;
}
public static int comps(String m,symtab s[],int sn)
{
        int i;
        for(i=0;i<sn;i++)
        {
                if(m.equals(s[i].name))
                        return s[i].index;
        }
        return 99;
}
public static String ic(String m)
{
        if(m =="START")
                return"(AD,01)";
        else if(m =="END")
                return"(AD,02)";
        else if(m =="ORIGIN")
                return"(AD,03)";
        else if(m =="EQU")
                return"(AD,04)";
        else if(m =="LTORG")
                return"(DL,02)";
        else if(m =="ADD")
                return"(IS,01)";
        else if(m =="SUB")
                return"(IS,02)";
        else if(m =="MOVER")
                return"(IS,04)";
        else if(m =="MOVEM")
                return"(AD,05)";
        else if(m =="AREG")
                return"(RG,01)";
        else if(m =="BREG")
                return"(RG,02)";
        else if(m =="CREG")
                return"(RG,03)";
        else if(m =="DS")
                return"(DL,01)";
        else if(m =="DC")
                return"(DL,02)";
        else if(m.toCharArray()[0]=='=')
                return("(C,"+m.toCharArray()[2]+")");
        else
        {
                return("(C,"+m+")");
        }
```

```
                    }
}
```

OUTPUT:

```
C:\Users\R.R.B\Desktop>javac pass1.java
C:\Users\R.R.B\Desktop>java pass1
```

Symbol Table

Index	Symbol	Address
0	A	110
1	AGAIN	101
2	B	109
3	AGAIN	103
4	AGAIN2	106
5	AGAIN2	107
6	AGAIN	108
7	LOOP	110

LiteralTable

Index	Literal	Address
0	='2'	110
1	='3'	101
2	='2'	109

Pool Table

Pool Index	Literal Index
0	0
1	2

Intermediate Code

```
(AD,01)(C,100)
(IS,04)(RG,01)(S,0)
(IS,01)(RG,01)(L,0)
(IS,01)(RG,01)(S,2)
(IS,01)(RG,01)(L,1)
(DL,02)(C,2)
(DL,02)(C,3)

(IS,01)(RG,01)(RG,02)
(IS,01)(RG,01)(RG,03)
(IS,01)(RG,01)(L,2)

(AD,02)(DL,02)(C,2)
C:\Users\R.R.B\Desktop>
```

Pass-II Program-

```
import java.io.*;
import java.nio.channels.SeekableByteChannel;
import java.nio.file.Files;
import java.util.*;


import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

class data{
public  String seq;
public  String value;
public  String addr;

}

public class Pass2  {


static String lc;
static int reg;

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

File ic = new File("/home/ccoew/3908/Pass2/ic.txt");

BufferedReader br1 = new BufferedReader(new FileReader(ic));

File sym = new File("/home/ccoew/3908/Pass2/sym.txt");

BufferedReader br2 = new BufferedReader(new FileReader(sym));

File lit = new File("/home/ccoew/3908/Pass2/lit.txt");

BufferedReader br3 = new BufferedReader(new FileReader(lit));

File pool = new File("/home/ccoew/3908/Pass2/pool.txt");

BufferedReader br4 = new BufferedReader(new FileReader(pool));
String str1;


File tc1=new File("/home/ccoew/3908/Pass2/tc.txt");

if(tc1.exists()){
        tc1.delete();
```

```
}
File tc=new File("/home/ccoew/3908/Pass2/tc.txt");


FileWriter fw=new FileWriter(tc);
int cnt=0;

//--------------------------------DATA STRUCTURES------------------------------
String str=new String();

//--------------------------------literals----------------------
ArrayList<data>l=new ArrayList<data>();
while((str=br3.readLine())!=null)
{
	StringTokenizer st=new StringTokenizer(str," ");
	data a=new data();
	a.seq=st.nextToken();
	a.value=st.nextToken();
	a.addr=st.nextToken();
	l.add(a);
}
br3.close();
//-------------------------------------symbols--------------------------

ArrayList<data>s=new ArrayList<data>();
while((str=br2.readLine())!=null)
{
	StringTokenizer st=new StringTokenizer(str," ");
	data a=new data();
	a.seq=st.nextToken();
	a.value=st.nextToken();
	a.addr=st.nextToken();
	s.add(a);
}
br2.close();
//-------------------------LOOP-----------------------------------

str1=br1.readLine();
while((str1=br1.readLine())!=null)
{
	StringTokenizer st=new StringTokenizer(str1," ,()");
	//System.out.println(st.nextToken());
	String arr[]=new String[st.countTokens()];
	for(int i=0;i<arr.length;i++)
	{
		arr[i]=st.nextToken();

	}


	if(arr.length==6)
```

```
{

        String ad=new String();

        lc=arr[0];

        for(int i=0;i<l.size();i++)
        {
                if(l.get(i).seq.equals(arr[5]))
                {
                        ad=l.get(i).addr;
                        break;
                }
        }
        String r=arr[3];
        switch(r)
        {
        case "AREG":reg=1;
                break;
        case "BREG":reg=2;
                break;
        case "CREG":reg=3;
                break;
        case "DREG":reg=4;
                break;
        }
        fw.write(lc+" "+arr[2]+" "+reg+" "+ad+"\n");

}
else if(arr.length==5)
{

        String ad=new String();

        lc=arr[0];
        for(int i=0;i<s.size();i++)
        {
                if(s.get(i).value.equals(arr[4]))
                {
                        ad=s.get(i).addr;
                        break;
                }
        }
        String r=arr[3];
        switch(r)
        {
        case "AREG":reg=1;
                break;
        case "BREG":reg=2;
                break;
        case "CREG":reg=3;
```

```
                        break;
                case "DREG":reg=4;
                        break;
                }
                fw.write(lc+" "+arr[2]+" "+reg+" "+ad+"\n");
        }
        else if(arr.length==4)
        {
                lc=arr[0];
                fw.write(lc+"\n");
        }
        else if(arr.length==3)
        {
                if(arr[2].equals("00"))
                {
                        fw.write(arr[0]+" "+arr[2]+"\n");
                }
                else
                {
                        fw.write("\n");
                }
        }
        else if(arr.length==2)
        {
                if(arr[1].equals("05")||(arr[1].equals("02"))||(arr[1].equals("04")))
                fw.write("\n");
                else
                {
                        fw.write(arr[0]+"    "+arr[1]+"\n");
                }
        }


}

fw.close();
}
}
```

OUTPUT:

Input.txt

```
START 200
MOVER AREG, ='5'
MOVEM AREG,A
MOVEM CREG,B
ADD CREG,='1'
BC CREG,NEXT
LTORG
    ='5'
    ='1'
NEXT: SUB AREG,='1'
```

```
BC AREG,='1'
LAST: STOP
ORIGIN LOOP+2
MULT CREG,B
ORIGIN LAST+1
A: DS 1
B: DS 1
END ='1'
```

Intermediate code

```
ADD IS 01 3
SUB IS  02 3
MULT IS 03 3
MOVER IS 04 3
MOVEM IS 05 3
STOP IS 00 1
BC IS 07 3
```

Target code

```
200 04 1 218
203 05 1
206 04 1
209 04 3
212 01 3 219
215 07 3

218    5
219    1
220 02 1 229
223 07 1
226 00

208 03 3

227

228
```

GROUP - A
EXPERIMENT NO: 02

Aim: Design suitable data structures and implement pass-I of a two-pass macro-processor using OOP features in Java. The output of Pass-I (MNT, MDT and file without any macro definitions) should be input for Pass-II.

Objectives:

- To Identify and create the data structures required in the design of macro processor.
- To Learn parameter processing in macro
- To implement pass II of macroprocessor

Problem Statement: Write a program to create pass-I Macro-processor

Outcomes: After completion of this assignment students will be able to:

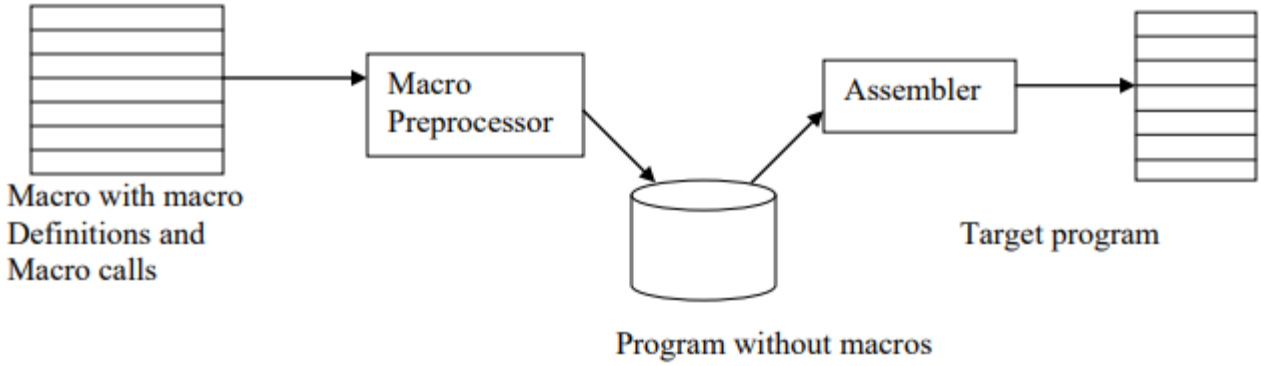
- Understand the Programming language of Java
- Understand the concept of Pass-I Macro-processor

Software Requirements: Windows OS, JDK1.7

Theory Concepts:

MACRO:-

- Macro allows a sequence of source language code to be defined once & then referred to by name each time it is to be referred.



- Macro Preprocessor Macro preprocessor is system software.
- It is actually a program, which is a part of the assembler program.
- A macro preprocessor accepts an assembly level program containing macro definitions and calls and translate it into an assembly program which does not contain any macro definition and macro call.
- Each time this name occurs is a program the sequence of codes is substituted at that point.
- A macro consist of
  1. Name of the macro
  2. Set of parameters
  3. Body of macro
- Macros are typically defined at the start of program. Macro definition consists of
  1.MACRO Start
  2.MACRO name
  3.Sequence of statements(MACRO Body)

4.MEND End

- A macro is called by writing the macro name with actual parameter is an assembly program. The macro call has following syntax .

  <macro name>[<formal parameters>]
- Macro processor takes a source program containing macro definition & macro calls and translates into an assembly language program without any macro definition or calls.
- This program can now be handled over to a conventional assembler to obtain the target language.

MACRO DEINITION:-

- Macros are typically defined at the start of a program.
- A macro definition consists of
  1. MACRO pseudo code 2. MACRO name 3. Sequence of statement 4. MEND pseudo opcode terminating macro definition Structure of a macro

Macro instruction definition

MACRO →start of definition
INCR →macro name

----- } →Sequence to be abbreviated
MEND →end of definition

- **Example**

MACRO INCR & ARG
    ADD AREG,& ARG
    ADD BRA,& ARG
    ADD CREG, & ARG
MEND

MACRO CALL:-

Syntax of macro call

<macro name>[<formal parameters>]

MACRO Expansion:-

- During macro expansion each statement forming the body of the macro as picked up one by one sequentially.

  a. Each statement inside macro may have as it is during expansion.

  b. The name of a formal parameter which is preceded by the character '&' during macro expansion an ordinary starting is retained without any modification. Formal parameters are replaced by actual parameters value.

- When a call is found the call processor sets a pointer the macro definition table pointer to the corresponding macro definition started in MDT. The initial value of MDT is obtained from MDT index.

Design of macro processor:-

Pass I:

Generate Macro Name Table (MNT)
Generate Macro Definition Table (MDT)
Generate IC i.e. a copy of source code without macro definitions

MNT:

Sr.No	Macro Name	MDT Index

MDT:

Sr. No	MACRO STATEMENT

ALA:

Index	Argument

- **Specification of Database**
  - Pass 1 data bases
    1. The input macro source desk.
    2. The output macro source desk copy for use by passes 2.
    3. The macro definition table (MDT) used to store the names of defined macros.
    4. Macro name table (MDT) used to stare the name of defined macros.
    5. The Macro definition table counter used to indicate the next available entry in MNT.
    6. The macro name table counter counter(MNTC) used to indicate next available entry in MNT.
    7. The arguments list array (ALA) used to substitute index markers for dummy arguments before starting a macro definition.

Pass II:

- Replace every occurrence of macro call with macro definition. (Expanded Code)
- There are four basic tasks that any macro instruction process must perform:
  1. **Recognize macro definition:** A macro instruction processor must recognize macro definition identified by the MACRO and MEND pseudo-ops. This tasks can be complicated when macro definition appears within macros. When MACROs and MENDs are nested, as the macro processor must recognize the nesting and correctly match the last or or outer

MEND with first MACRO. All intervening text, including nested MACROs and MENDs defines a single macro instruction.

2. **Save the definition:** The processor must store the macro instruction definition, which it will need for expanding macro calls.
3. **Recognize calls:** The processor must recognize the macro calls that appear as operation mnemonics. This suggests that macro names be handled as a type of op-code.
4. **Expand calls and substitute arguments:** The processor must substitute for dummy or macro definition arguments the corresponding arguments from a macro call; the resulting symbolic text is then substitute for macro call. This text may contain additional macro definition or call.

- **Specification of Database**
  - Pass 2 database:
    1. The copy of the input source deck obtained from Pass- I
    2. The output expanded source deck to be used as input to the assembler
    3. The Macro Definition Table (MDT), created by pass 1
    4. The Macro Name Table (MNT), created by pass 1
    5. The Macro Definition Table Counter (MNTC), used to indicate the next line of text to be used during macro expansion
    6. The Argument List Array (ALA), used to substitute macro call arguments for the index markers in stored macro definition

Eg:- Consider the following Macro definition and the tables constructed in pass I for this macro

```
MACRO
  &LAB  INCR     &ARG1,&ARG2,&ARG3
  &LAB    ADD  1,&ARG1
            ADD
            2,&ARG2
            ADD
            3,&ARG3
MEND
```

Output:-

MNT

Index	Macro Name	MDT Index
1	INCR	1

MDT

Index	Label	Instruction	operands
1	***#0***	ADD	1, ***#1***
2		ADD	1, ***#2***
3		ADD	1, ***#3***
4		MEND	

ALA

Argument	Dummy name
&Lab	#0
&Arg1	#1
&Arg2	#2
&Arg3	#3

Steps to follow:-

1. Read the ALP word by word using fgetc.
2. If keyword MACRO is found then
   - Store the arguments in ALA in sequence(0^{th} argument for a label).
   - Store all the instructions in MDT using dummy names of the arguments(prepared in ALA).
   - Store the MDT index in a variable temp.
   - Store name of macro and its MDT index using temp in MNT.
3. Write all the tables onto a file (as these will be required in pass II)
4. Go to Pass I

The task of pass II is macro expansion i.e deal with macro calls. For this we use the tables created in pass I of a macro-processor. Whenever a call to a macro is found for eg:-

LOOP INCR Data1,Data2,Data3

(with respect to the macro definition INCR in previous assignment)

Then this call should lead to a macro expansion as follows:-

```
LOOP :  ADD    1 , Data1
        ADD    2 ,  Data2
        ADD    3 , Data3
```

Here the formal parameters are replaced by actual parameters.

For this ALA has to be updated with new parameter names.

ALA

Argument	Actual Parameters
#0	LOOP
#1	Data1
#2	Data2
#3	Data3

Using these values we expand the macro with the help of MNT and MDT.

Steps to follow:-

1. Read the source program.
2. If an macro name is encounterd it means there is a call to the macro so do the following.
    - Search for the macro in the MNT.
    - Extract its MDT index and go to that index position in MDT.
    - Update ALA with actual parameters.
3. Read from MDT and replace dummy parameters with actual parameters.
4. Do step 3 till end of macro.

PASS-1 Algorithm

• Pass1 of macro processor makes a line-by-line scan over its input.

• Set MDTC = 1 as well as MNTC = 1.

• Read next line from input program.

• If it is a MACRO pseudo-op, the entire macro definition except this (MACRO) line is stored in MDT.

• The name is entered into Macro Name Table along with a pointer to the first location of MDT entry of the definition.

• When the END pseudo-op is encountered all the macro-definitions have been processed, so control is transferred to pass2

PASS-2 Algorithm

• This algorithm reads one line of input program at a time.

• for each Line it checks if op-code of that line matches any of the MNT entry.

• When match is found (i.e. when call is pointer called MDTF to corresponding macro definitions stored in MDT.

• The initial value of MDTP is obtained from MDT index field of MNT entry.

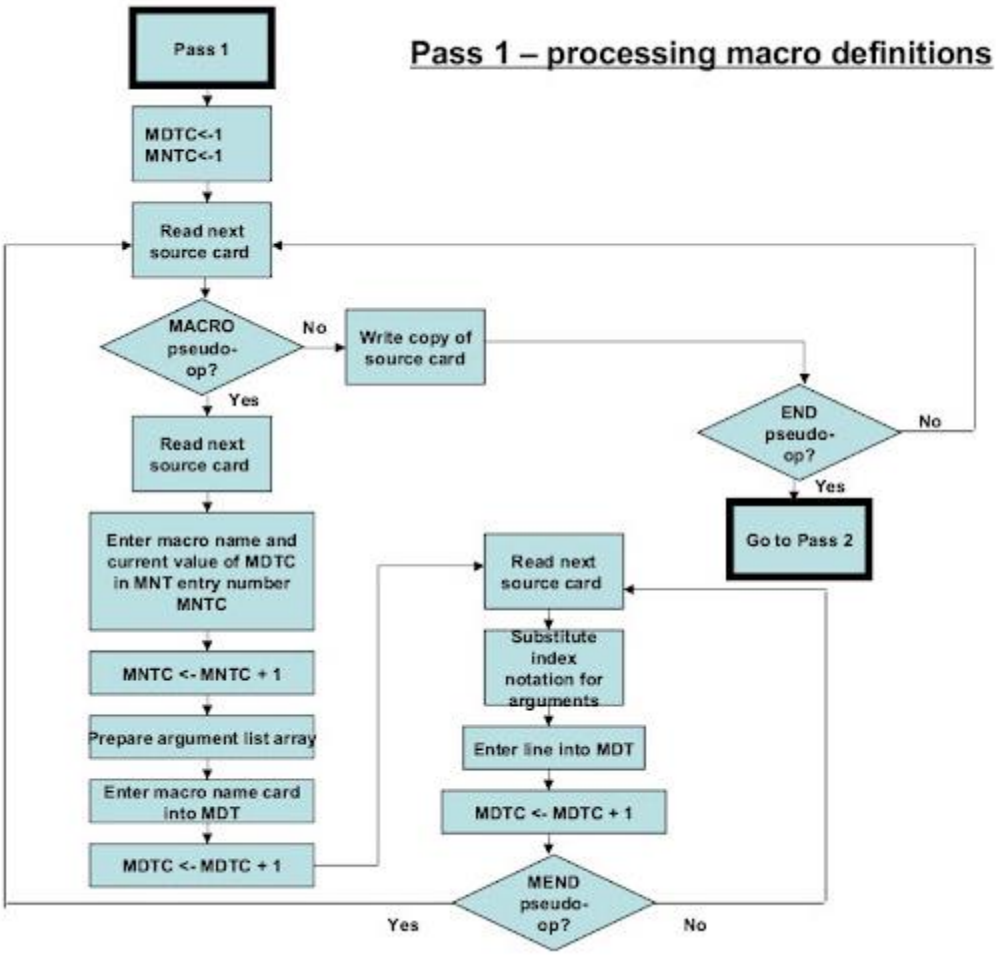
•The macro expander prepares the ALA consisting of a table of dummy argument indices & corresponding arguments to the call.

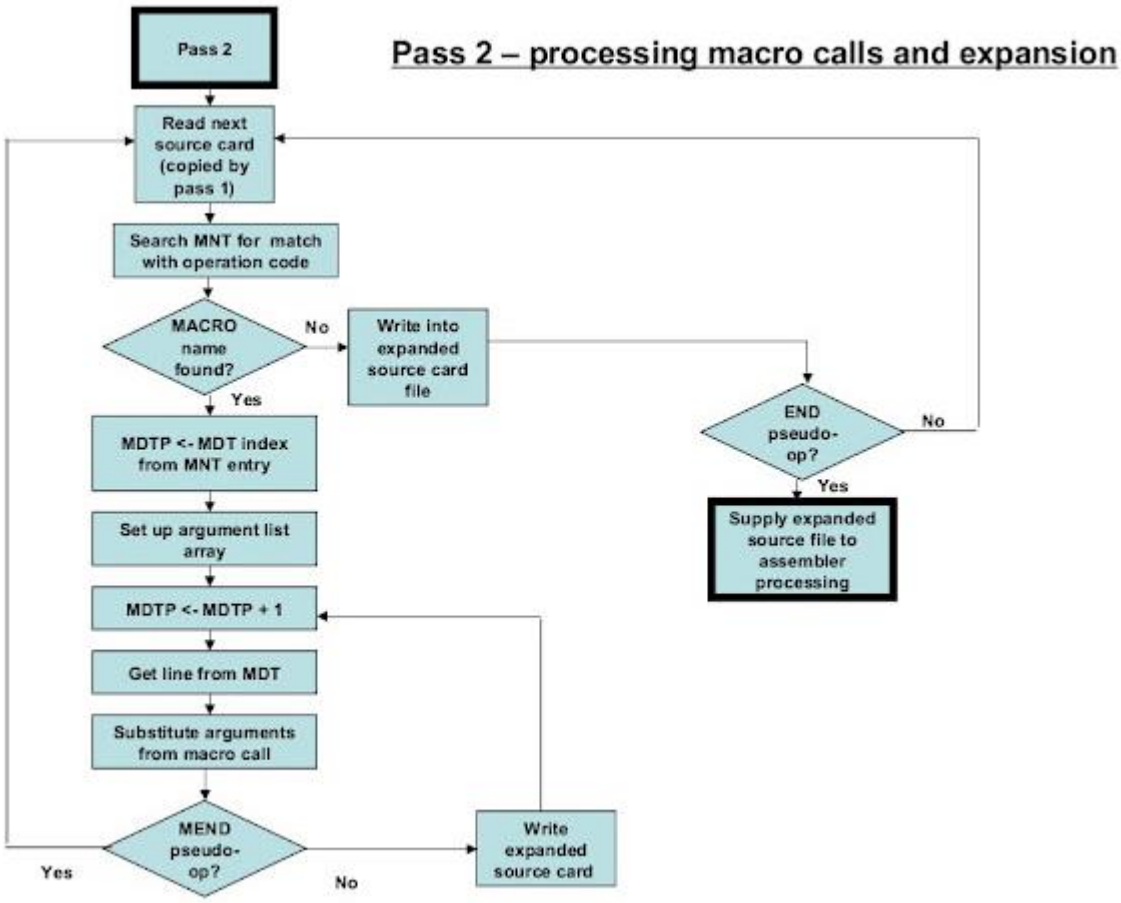
•Reading proceeds from the MDT, as each successive line is read, The values form the argument list one substituted for dummy arguments indices in the macro definition.

• Reading MEND line in MDT terminates expansion of macro & scanning continues from the input file.

• When END pseudo-op encountered , the expanded source program is given to the assembler.

Pass 1 – processing macro definitions





Conclusion: Thus, we have studied and implemented Pass-I and Pass-II of Macro Processor.

Program code:

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

class MntTuple {
String name;
int index;

MntTuple(String s, int i) {
        name = s;
        index = i;
}

public String toString() {
        return("[" + name + ", " + index + "]");
}
}

class MacroProcessor {
static List<MntTuple> mnt;
static List<String> mdt;
static int mntc;
static int mdtc;
static int mdtp;
static BufferedReader input;
static List<List <String>> ala;
static Map<String, Integer> ala_macro_binding;

public static void main(String args[]) throws Exception {
        initializeTables();
        System.out.println("===== PASS 1 =====\n");
        pass1();
        System.out.println("\n===== PASS 2 =====\n");
        pass2();
}

static void pass1() throws Exception {
        String s = new String();
        input = new BufferedReader(new InputStreamReader(new FileInputStream("input.txt")));
        PrintWriter output = new PrintWriter(new FileOutputStream("output_pass1.txt"), true);
        while((s = input.readLine()) != null) {
                if(s.equalsIgnoreCase("MACRO")) {
                        processMacroDefinition();
                } else {
                        output.println(s);
                }
        }
        System.out.println("ALA:");
        showAla(1);
        System.out.println("\nMNT:");
```

```
        showMnt();
        System.out.println("\nMDT:");
        showMdt();
}

static void processMacroDefinition() throws Exception {
        String s = input.readLine();
        String macro_name = s.substring(0, s.indexOf(" "));
        mnt.add(new MntTuple(macro_name, mdtc));
        mntc++;
        pass1Ala(s);
        StringTokenizer st = new StringTokenizer(s, " ,", false);
        String x = st.nextToken();
        for(int i=x.length() ; i<12 ; i++) {
                x += " ";
        }
        String token = new String();
        int index;
        token = st.nextToken();
        x += token;
        while(st.hasMoreTokens()) {
                token = st.nextToken();
                x += "," + token;
        }
        mdt.add(x);
        mdtc++;
        addIntoMdt(ala.size()-1);
}

static void pass1Ala(String s) {
        StringTokenizer st = new StringTokenizer(s, " ,", false);
        String macro_name = st.nextToken();
        List<String> l = new ArrayList<>();
        int index;
        while(st.hasMoreTokens()) {
                String x = st.nextToken();
                if((index = x.indexOf("=")) != -1) {
                        x = x.substring(0, index);
                }
                l.add(x);
        }
        ala.add(l);
        ala_macro_binding.put(macro_name, ala_macro_binding.size());
}

static void addIntoMdt(int ala_number) throws Exception {
        String temp = new String();
        String s = new String();
        List l = ala.get(ala_number);
        boolean isFirst;
        while(!s.equalsIgnoreCase("MEND")) {
```

```
                isFirst = true;
                s = input.readLine();
                String line = new String();
                StringTokenizer st = new StringTokenizer(s, " ,", false);
                temp = st.nextToken();
                for(int i=temp.length() ; i<12 ; i++) {
                        temp += " ";
                }
                line += temp;
                while(st.hasMoreTokens()) {
                        temp = st.nextToken();
                        if(temp.startsWith("&")) {
                                int x = l.indexOf(temp);
                                temp = ",#" + x;
                                isFirst = false;
                        } else if(!isFirst) {
                                temp = "," + temp;
                        }
                        line += temp;
                }
                mdt.add(line);
                mdtc++;
        }
}

static void showAla(int pass) throws Exception {
        PrintWriter out = new PrintWriter(new FileOutputStream("out_ala_pass" + pass + ".txt"), true);
        for(List l : ala) {
                System.out.println(l);
                out.println(l);
        }
}

static void showMnt() throws Exception {
        PrintWriter out = new PrintWriter(new FileOutputStream("out_mnt.txt"), true);
        for(MntTuple l : mnt) {
                System.out.println(l);
                out.println(l);
        }
}

static void showMdt() throws Exception {
        PrintWriter out = new PrintWriter(new FileOutputStream("out_mdt.txt"), true);
        for(String l : mdt) {
                System.out.println(l);
                out.println(l);
        }
}

static void pass2() throws Exception {
        input = new BufferedReader(new InputStreamReader(new FileInputStream("output_pass1.txt")));
```

```
PrintWriter output = new PrintWriter(new FileOutputStream("output_pass2.txt"), true);
String token = new String();
String s;
while((s = input.readLine()) != null) {
	StringTokenizer st = new StringTokenizer(s, " ", false);
	while(st.hasMoreTokens()) {
		token = st.nextToken();
		if(st.countTokens() > 2) {
			token = st.nextToken();
		}
		MntTuple x = null;
		for(MntTuple m : mnt) {
			if(m.name.equalsIgnoreCase(token)) {
				x = m;
				break;
			}
		}
		if(x != null) {
			mdtp = x.index;
			List<String> l = pass2Ala(s);
			mdtp++;
			String temp = new String();
			while(!(temp = mdt.get(mdtp)).trim().equalsIgnoreCase("MEND")) {
				String line = new String();
				StringTokenizer st2 = new StringTokenizer(temp, " ,",false);
				for(int i=0 ; i<12 ; i++) {
					line += " ";
				}
				String opcode = st2.nextToken();
				line += opcode;
				for(int i=opcode.length() ; i<24 ; i++) {
					line += " ";
				}
				line += st2.nextToken();
				while(st2.hasMoreTokens()) {
					String token2 = st2.nextToken();
					int index;
					if((index = token2.indexOf("#")) != -1) {
						line += "," + l.get(Integer.parseInt
(token2.substring(index+1,index+2)));
					}
				}
				mdtp++;
				output.println(line);
				System.out.println(line);
			}
			break;
		} else {
			output.println(s);
			System.out.println(s);
			break;
```

```
                        }
                    }
                }
                System.out.println("\nALA:");
                showAla(2);
        }

        static List<String> pass2Ala(String s) {
                StringTokenizer st = new StringTokenizer(s, " ", false);
                int num_tokens = st.countTokens();
                String macro_name = st.nextToken();
                int ala_no = ala_macro_binding.get(macro_name);
                List<String> l = ala.get(ala_no);
                int ctr = 0;
                StringTokenizer st2 = null;
                try {
                        st2 = new StringTokenizer(st.nextToken(), ",", false);
                        while(st2.hasMoreTokens()) {
                                l.set(ctr, st2.nextToken());
                                ctr++;
                        }
                } catch(Exception e) {
                        // do nothing
                }
                if(ctr < num_tokens) {
                        String s2 = mdt.get(mdtp);
                        StringTokenizer st3 = new StringTokenizer(s2, " ,", false);
                        String token = new String();
                        int index = 0;
                        while(st3.hasMoreTokens()) {
                                token = st3.nextToken();
                                if((index = token.indexOf("=")) != -1) {
                                        try {
                                                l.set(ctr++, token.substring(index+1, token.length()));
                                        } catch(Exception e) {
                                                // do nothing
                                        }
                                }
                        }
                }
                ala.set(ala_no, l);
                return l;
        }

        static void initializeTables() {
                mnt = new LinkedList<>();
                mdt = new ArrayList<>();
                ala = new LinkedList<>();
                mntc = 0;
                mdtc = 0;
                ala_macro_binding = new HashMap<>();
```

```
}
}
```

OUTPUT::

```
**********input.txt**********
MACRO
INCR1      &FIRST,&SECOND=DATA9
A          1,&FIRST
L          2,&SECOND
MEND
MACRO
INCR2      &ARG1,&ARG2=DATA5
L          3,&ARG1
ST         4,&ARG2
MEND
PRG2       START
           USING              *,BASE
           INCR1              DATA1
           INCR2              DATA3,DATA4
FOUR       DC                 F'4'
FIVE       DC                 F'5'
BASE       EQU                8
TEMP       DS                 1F
           DROP               8
           END

**********output pass1**********
PRG2       START
           USING              *,BASE
           INCR1              DATA1
           INCR2              DATA3,DATA4
FOUR       DC                 F'4'
FIVE       DC                 F'5'
BASE       EQU                8
TEMP       DS                 1F
           DROP               8
           END

**********output pass2**********
PRG2       START
           USING              *,BASE
           A                  1,DATA1
           L                  2,DATA9
           L                  3,DATA3
           ST                 4,DATA4
FOUR       DC                 F'4'
FIVE       DC                 F'5'
BASE       EQU                8
TEMP       DS                 1F
```

```
DROP            8
END
```

```
C:\Users\COMP\Downloads\SPCC-2passmacro>javac MacroProcessor.java

C:\Users\COMP\Downloads\SPCC-2passmacro>java MacroProcessor
===== PASS 1 =====

ALA:
[&FIRST, &SECOND]
[&ARG1, &ARG2]

MNT:
[INCR1, 0]
[INCR2, 4]

MDT:
INCR1       &FIRST,&SECOND=DATA9
A           1,#0
L           2,#1
MEND
INCR2       &ARG1,&ARG2=DATA5
L           3,#0
ST          4,#1
MEND

===== PASS 2 =====

PRG2        START
            USING                   *,BASE
            A                       1,DATA1
            L                       2,DATA9
            L                       3,DATA3
            ST                      4,DATA4
FOUR        DC                      F'4'
FIVE        DC                      F'5'
BASE        EQU                     8
TEMP        DS                      1F
            DROP                    8
            END

ALA:
[DATA1, DATA9]
[DATA3, DATA4]

C:\Users\COMP\Downloads\SPCC-2passmacro>
```

GROUP - B
EXPERIMENT NO: 01

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

Objectives:

- To study preemptive and non-preemptive scheduling algorithm.
- To implement FCFS and SJF with preemptive scheduling.
- To implement Priority and Round Robin non preemptive scheduling using JAVA programming language.

Problem Statement: Write a program to simulate CPU Scheduling Algorithms

Outcomes:

- Understand difference between preemptive and non-preemptive scheduling algorithm.
- Study different scheduling algorithm.
- Implement scheduling algorithm using JAVA programming language.

Software Requirements: Windows OS, JDK1.7

Theory Concepts:

- **Scheduling:-**
  - A multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time-multiplexing.
  - Part of the reason for using multiprogramming is that the operating system itself is implemented as one or more processes, so there must be a way for the operating system and application processes to share the CPU.
  - Another main reason is the need for processes to perform I/O operations in the normal course of computation.
  - Since I/O operations ordinarily require orders of magnitude more time to complete than do CPU instructions, multiprogramming systems allocate the CPU to another process whenever a process invokes an I/O operation

- **Necessity of scheduling**
  - Scheduling is required when no. of jobs are to be performed by CPU.
  - Scheduling provides mechanism to give order to each work to be done.
  - Primary objective of scheduling is to optimize system performance.
  - Scheduling provides the ease to CPU to execute the processes in efficient manner.

- **Preemptive and Non-Preemptive Scheduling**

  - **Preemptive scheduling**: The preemptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized.

  - **Non-Preemptive scheduling**: When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.

- **Types of schedulers**
  - In general, there are three different types of schedulers which may co-exist in a complex operating system.
    - Long term scheduler
    - Medium term scheduler
    - Short term scheduler.

- **Below are different times with respect to a process.**
  - **Arrival Time :** Time at which the process arrives in the ready queue.
  - **Completion Time :** Time at which process completes its execution.
  - **Burst Time** : Time required by a process for CPU execution.
  - **Turn Around Time :** Time Difference between completion time and arrival time.
    - Turn Around Time = Completion Time - Arrival Time
  - **Waiting Time(W.T) :** Time Difference between turn around time and burst time.
    - Waiting Time = Turn Around Time - Burst Time

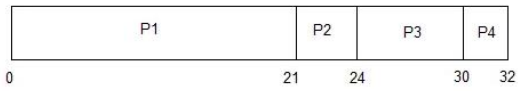
- **Different Scheduling Algorithms:**
  1. **First Come First Serve (FCFS)**
  2. **Shortest Job First (SJF)**
  3. **Round Robin Scheduling**
  4. **Priority Based scheduling**

- **FCFS Scheduling**

  Simplest scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is a non – preemptive scheduling algorithm.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

  - Jobs are executed on FCFS (First Come, First Serve) basis.
  - It is a non-preemptive, preemptive scheduling algorithm.
  - Easy to understand and implement.
  - Its implementation is based on FIFO (First In First Out) queue.
  - Poor in performance as average wait time is high

- **SJF Scheduling**

  Process which has the shortest burst time is scheduled first. If two processes have the same bust time then FCFS is used to break the tie. It is a non-preemptive scheduling algorithm

  - This is also known as shortest job first, or SJF
  - This is a non-preemptive, pre-emptive scheduling algorithm.
  - Best approach to minimize waiting time.
  - Easy to implement in Batch systems where required CPU time is known in advance.
  - Impossible to implement in interactive systems where required CPU time is not known.
  - The processer should know in advance how much time process will take.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :

Now, the average waiting time will be = (0 + 2 + 5 + 11)/ 4 = 4.5 ms

- **Round Robin Scheduling**

  Each process is assigned a fixed time (Time Quantum/Time Slice) in cyclic way.

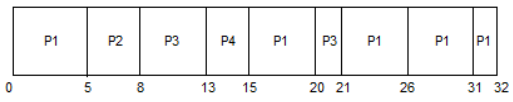
  It is designed especially for the time-sharing system.

  The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1-time quantum. To implement Round Robin scheduling, we keep the ready queue as a FIFO queue o£ processes. New processes are added to the tail of the ready queue.

  The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1-time quantum, and dispatches the process.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

One of two things will then happen. The process may have a CPU burst of less than 1-time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1-time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail o£ the ready queue. The CPU scheduler will then select the next process in the ready queue.

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a quantum.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

- **Priority Scheduling**

  In this scheduling, processes are scheduled according to their priorities, i.e., highest priority process is scheduled first. If priorities of two processes match, then schedule according to arrival time.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,

The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

  - Priority scheduling is a non-preemptive algorithm and one of' the most common scheduling algorithms in batch systems.
  - Each process is assigned a priority. Process with highest priority is to be executed first and so on.
  - Processes with same priority are executed on first come first served basis.
  - Priority can be decided based on memory requirements, time requirements or any other resource requirement.

- **Algorithms**

FCFS :

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Set the waiting of the first process as '0' and its burst time as its turnaround time
Step 5: for each process in the Ready Q calculate
(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
Step 6: Calculate
(a) Average waiting time = Total waiting Time / Number of process
(b) Average Turnaround time = Total Turnaround Time / Number of process
Step 7: Stop the process

SJF :

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.
Step 6: For each process in the ready queue, calculate
(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

RR :

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Calculate the no. of time slices for each process where No. of time slice for process(n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

(a) Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

(b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process.

Priority Scheduling:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time, priority

Step 4: Start the Ready Q according the priority by sorting according to lowest to highest burst time and process.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(b) Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 7: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

Conclusion: Thus, we have studied and implemented FCFS , SJF, Priority and Round Robin scheduling algorithm.

Program code:

FCFS

```
import java.util.*;
class Fcfs
{
	public static void main(String[] args)
	{
		int id[]=new int[20];
		int etime[]=new int[20];
		int stime[]=new int[20];
		int wtime[] = new int[20];
		int tat[]=new int[20];
		int total=0,totall=0;
		float avg,avgl;

		Scanner sn = new Scanner(System.in);
		System.out.print("\nEnter the number of processes: ");
		int n = sn.nextInt();

		for (int i=0;i<n;i++)
		{
			System.out.println();
			System.out.print("Enter the process ID of process"+(i+1)+":");
			id[i]=sn.nextInt();
			System.out.print("Enter the execution time of process"+(i+1)+": ");
			etime[i]=sn.nextInt();
		}
		stime[0]=0;

		for (int i=1;i<n;i++)
		{
			stime[i]=stime[i-1]+etime[i-1];
		}
		wtime[0]=0;

		for (int i=0;i<n;i++)
		{
			wtime[i]=stime[i]-id[i];
			total=total+wtime[i];
		}

		for(int i=0;i<n;i++)
		{
			tat[i]=wtime[i]+etime[i];
			totall=totall+tat[i];
		}
		avg= (float)total/n;
		avgl=(float)totall/n;

		System.out.println("\nArrival_time\tExecution_time\tService_time\t  Wait_time\tturn_around
		time");
```

```
            for(int i=0;i<n;i++)
            {
                System.out.println(id[i]+"\t\t"+etime[i]+"\t\t"+stime[i] +"\t\t"+wtime[i]+"\t\t"+tat[i]);
            }

            System.out.println("\nAverage turn around time: "+avgl+"\nAverage wait time: "+avg);
        }
}
```

**********OUTPUT*********

```
C:\Users\R.R.B\Desktop>javac Fcfs.java

C:\Users\R.R.B\Desktop>java Fcfs

Enter the number of processes: 4

Enter the process ID of process1:0
Enter the execution time of process1: 8

Enter the process ID of process2:1
Enter the execution time of process2: 4

Enter the process ID of process3:2
Enter the execution time of process3: 9

Enter the process ID of process4:3
Enter the execution time of process4: 5

Arrival_time    Execution_time  Service_time    Wait_time       turn_around time
0                       8               0               0               8
1                       4               8               7               11
2                       9               12              10              19

3                       5               21              18              23

Average turn around time: 15.25
Average wait time: 8.75

C:\Users\R.R.B\Desktop>
```

Program code:
SJF

```
import java.util.*;
class Sjf
{
	public static void main(String[] args)
	{
		int id[]=new int[20];
		int etime[] = new int[20]; int stime]=new int[20];
		int wtime[] = new int[20];
		int btime[] = new int[20];
		int ctime[] = new int[20];
		int flag[]=new int[20];
		//check process is completed or not.
		int ta[]=new int[20];
		int total=0;
		int st=0;
		int temp,i;
		float avgwt=0,avgta=0;
		Scanner sn = new Scanner(System.in);
		System.out.print("\nEnter the number of processes: ");
		int n = sn.nextInt();
		for (i=0;i<n;i++)
		{
			System.out.println();
			System.out.print("Enter the process ID of process"+(i+1)+": "); id[i]=sn.nextInt();
			System.out.print("Enter the execution time of process "+(i+1)+": ");
			etime[i]=sn.nextInt(); btime[i]=etime[i];
			flag[i]=0;
		}
		while(true)
		{
			int min=9999,c=n;
			if(total==n) break;
			for(i=0;i<n;i++)
			{
				if((id[i]<=st) &&(etime[i]<min))(flag[i]==0)&&
				{
					min=etime[i];
					c=i;
				}
			}
			if(c==n)
				st++;
			else
			{
				etime[c]--;
				st++;
```

```
                        if(etime[c]==0)
                        {
                                ctime[c]=st;
                                flag[c]=1;
                                total++;
                        }
                }
        }
        for(i=0;i<n;i++)
        {
                ta[i]=ctime[i]-id[i];
                wtime[i]=ta[i]-btime[i];
                avgwt+=wtime[i];
                avgta+=ta[i];
        }
        System.out.println("\nArrival_time\tExecution_time\tcom    pletion_time\t    Wait_time\tturn
around time");
        for(i=0;i<n;i++)
        {
                System.out.println(id[i]+"\t\t"+btime[i]+"\t\t"+ctime[i] +"\t\t"+wtime[i]+"\t\t"+ta[i]);
        }
        System.out.println("\nAverage wait time:"+(float) (avgwt/n));
        System.out.println("\nAverage turn around time:"+(float) (avgta/n));
    }
}
```

**********OUTPUT*********

```
C:\Users\R.R.B\Desktop>javac Sjf.java

C:\Users\R.R.B\Desktop>java Sjf

Enter the number of processes: 4
Enter the process ID of process1: 0
Enter the execution time of process 1: 7
Enter the process ID of process2: 2
Enter the execution time of process 2: 4
Enter the process ID of process3: 4
Enter the execution time of process 3: 1
Enter the process ID of process4: 5
Enter the execution time of process 4: 4
Arrival_time    Execution_time          com pletion_time        Wait_time       turn around time
0                       7                       16                      9               16
2                       4                       7                       1               5
4                       1                       5                       0               1
5                       4                       11                      2               6
Average wait time:3.0
Average turn around time:7.0

C:\Users\R.R.B\Desktop>
```

Program code:
Priority

```
// Java program for implementation of FCFS
// scheduling
import java.util.*;

class Process {
	int pid; // Process ID
	int bt; // CPU Burst time required
	int priority; // Priority of this process
	Process(int pid, int bt, int priority)
	{
	  this.pid = pid;
	  this.bt = bt;
	  this.priority = priority;
	}
	public int prior()
	{
		return priority;
	}
}

public class Priority{

// Function to find the waiting time for all
// processes
public void findWaitingTime(Process proc[], int n,int wt[])
{
	// waiting time for first process is 0
	wt[0] = 0;

	// calculating waiting time
	for (int i = 1; i < n; i++)
		wt[i] = proc[i - 1].bt + wt[i - 1];
}

// Function to calculate turn around time
public void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
{
	// calculating turnaround time by adding
	// bt[i] + wt[i]
	for (int i = 0; i < n; i++)
		tat[i] = proc[i].bt + wt[i];
}

// Function to calculate average time
public void findavgTime(Process proc[], int n)
{
	int wt[] = new int[n], tat[] = new int[n],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.print( "\nProcesses 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.print(" " + proc[i].pid + "\t\t"+ proc[i].bt + "\t " + wt[i]+ "\t\t " + tat[i] + "\n");
        }
        System.out.print("\nAverage waiting time = "+ (float)total_wt / (float)n);
        System.out.print("\nAverage turn around time = "+ (float)total_tat / (float)n);
}
public void priorityScheduling(Process proc[], int n)
{
        // Sort processes by priority
        Arrays.sort(proc, new Comparator<Process>()
        {
                @Override
                public int compare(Process a, Process b)
                {
                        return b.prior() - a.prior();
                }
        });
        System.out.print( "Order in which processes gets executed \n");
        for (int i = 0; i < n; i++)
                System.out.print(proc[i].pid + " ");
        findavgTime(proc, n);
}
// Driver code
public static void main(String[] args)
{
        Priority ob = new Priority();
        int n = 3;
        Process proc[] = new Process[n];
        proc[0] = new Process(1, 10, 2);
        proc[1] = new Process(2, 5, 0);
        proc[2] = new Process(3, 8, 1);
        ob.priorityScheduling(proc, n);
}
}
```

**********OUTPUT*********

C:\Users\R.R.B\Desktop>java Priority
Order in which processes gets executed
1 3 2
Processes Burst time Waiting time Turn around time
1 10 0 10
3 8 10 18
2 5 18 23

Average waiting time = 9.333333
Average turn around time = 17.0
C:\Users\R.R.B\Desktop>

Program code:
Round Robin

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

class RR
{
public static void main(String args[]) throws IOException
{
	BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
	System.out.println("Enter the Time Quantum: ");
	int q = Integer.parseInt(br.readLine());
	System.out.println("Please enter the number of Processes:");
	int n = Integer.parseInt(br.readLine());
	int proc[][] = new int[n + 1][4];  //proc[][0] is the AT array,][1]-RT,[][2] - WT,[[3] - TT
	for(int i = 1; i <= n; i++)
	{
		System.out.println("Please enter the Burst Time for Process " + i + ": ");
		proc[i][1] = Integer.parseInt(br.readLine());
	}
		System.out.println();
		//Calculation of Total Time and Initialization of Time Chart array
		int total_time = 0;
		for(int i = 1; i <= n; i++)
		{
			total_time += proc[i][1];
		}
		int time_chart[] = new int[total_time];
		int sel_proc = 1;
		int current_q = 0;
		for(int i = 0; i < total_time; i++)
		{
			//Assign selected process to current time in the Chart
```

```
                time_chart[i] = sel_proc;
            //Decrement Remaining Time of selected process by 1 since it has been assigned the CPU for 1 unit of time
            proc[sel_proc][1]--;
            //WT and TT Calculation
            for(int j=1;j<= n; j++)
            {
            if(proc[j][1] != 0)
            {
                proc[j][3]++;//If process has not completed execution its TT is incremented by 1
                if(j != sel_proc)//If the process has not been currently assigned the CPU its WT is incremented by 1
                proc[j][2]++;
            }
                else if(j == sel_proc)//This is a special case in which the process has been assigned CPU and has completed its execution
                proc[j][3]++;
            }
            //Printing the Time Chart
            if(i!= 0)

            {

                if(sel_proc != time_chart[i - 1])

                    //If the CPU has been assigned to a different Process we need to print the current value of time and the name of the new Process
                {
                    System.out.print("--"+i+"--P" + sel_proc);
                }

            } else          //If the current time is 0 i.e the printing has just started we need to print the name of the First selected Process
                System.out.print(i + "--P" + sel_proc);
            if(i == total_time - 1)//All the process names have been printed now we have to print the time at which execution ends
                    System.out.print("--" + (i + 1));
                //Updating value of sel_proc for next iteration
            current_q++;
            if(current_q == q || proc[sel_proc][1]== 0)//If Time slice has expired or the current process has completed execution
            {
                current_q=0;

            //This will select the next valid value for sel_proc

            for(int j = 1; j <= n; j++)
            {
                sel_proc++;
            if(sel_proc == (n + 1))
                sel_proc = 1;
```

```
                if(proc[sel_proc][1] != 0)
                        break;
                }
        }
}
System.out.println();

System.out.println();

        //Printing the WT and TT for each Process
        System.out.println("P\t WT \t TT ");
        for(int i = 1; i <= n; i++)
        {
                System.out.printf("%d\t%3dms\t%3dms",i,proc[i][2],proc[i][3]);
                System.out.println();
        }
        System.out.println();

//Printing the average WT & TT
        float WT=0,TT = 0;
        for(int i = 1; i <= n; i++)
        {
                WT+= proc[i][2];
                TT += proc[i][3];
        }
        WT/= n;
        TT/= n;
        System.out.println("The Average WT is: " + WT + "ms");
        System.out.println("The Average TT is: " + TT + "ms");

        }
}
```

**********OUTPUT*********

```
C:\Users\R.R.B\Desktop>javac RR.java

C:\Users\R.R.B\Desktop>java RR
Enter the Time Quantum:
3
Please enter the number of Processes:
5
Please enter the Burst Time for Process 1:
5
Please enter the Burst Time for Process 2:
3
Please enter the Burst Time for Process 3:
8
Please enter the Burst Time for Process 4:
6
Please enter the Burst Time for Process 5:
9
```

0--P1--3--P2--6--P3--9--P4--12--P5--15--P1--17--P3--20--P4--23--P5--26--P3--28--P5--31

P	WT	TT
1	12ms	17ms
2	3ms	6ms
3	20ms	28ms
4	17ms	23ms
5	22ms	31ms

The Average WT is: 14.8ms
The Average TT is: 21.0ms
C:\Users\R.R.B\Desktop>

GROUP - B
EXPERIMENT NO: 02

Aim: Write a program to simulate Page replacement algorithm.

Objectives:
- To study Page replacement algorithm.
- To implement FIFO, LRU and Optimal Page Replacement Algorithm

Problem Statement: Write a program to simulate Page replacement algorithms

Outcomes: After completion of this assignment students will be able to:
- Understand the Page replacement algorithm

Software Requirements: Windows OS, JDK1.7

Theory Concepts:

What are PAGE REPLACMENT Algorithms?

- As studied in Demand Paging, only certain pages of a process are loaded initially into the memory.
- This allows us to get more processes into memory at the same time.
- But what happens when a process requests for more pages and no free memory is available to bring them in.
- Following steps can be taken to deal with this problem:
  1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
  2. Remove some other process completely from the memory to free frames.
  3. Find some pages that are not being used right now, move them to the disk to get free frames. This technique is called Page replacement and is most commonly used.
- In this case, if a process requests a new page and supposes there are no free frames, then the Operating system needs to decide which page to replace.
- The operating system must use any page replacement algorithm in order to select the victim frame.
- The Operating system must then write the victim frame to the disk then read the desired page into the frame and then update the page tables. And all these require double the disk access time.

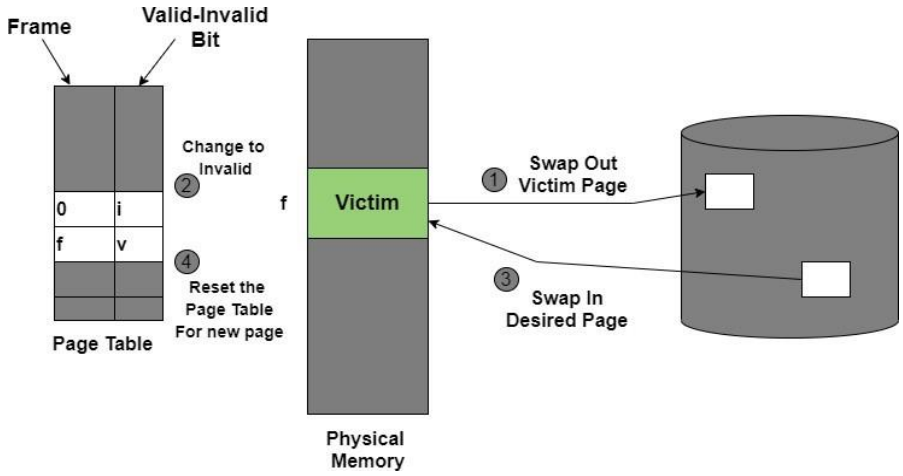
- **Page Replacement in OS**

In Virtual Memory Management, Page Replacement Algorithms play an important role. The main objective of all the Page replacement policies is to decrease the maximum number of page faults.

- **Page Fault –** It is basically a memory error, and it occurs when the current programs attempt to access the memory page for mapping into virtual address space, but it is unable to load into the physical memory then this is referred to as Page fault.

- **Basic Page Replacement Algorithm in OS**

Page Replacement technique uses the following approach. If there is no free frame, then we will find the one that is not currently being used and then free it. A-frame can be freed by writing its content to swap space and then change the page table in order to indicate that the page is no longer in the memory.



1. First of all, find the location of the desired page on the disk.
2. Find a free Frame: a) If there is a free frame, then use it. b) If there is no free frame then make use of the page-replacement algorithm in order to select the victim frame. c) Then after that write the victim frame to the disk and then make the changes in the page table and frame table accordingly

3. After that read the desired page into the newly freed frame and then change the page and frame tables.
4. Restart the process.

- **Page Replacement Algorithms in OS**

1. FIFO Page Replacement Algorithm

- It is a very simple way of Page replacement and is referred to as First in First Out.
- This algorithm mainly replaces the oldest page that has been present in the main memory for the longest time.
- This algorithm is implemented by keeping the track of all the pages in the queue.
- As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim.
- This is not an effective way of page replacement but it can be used for small systems.

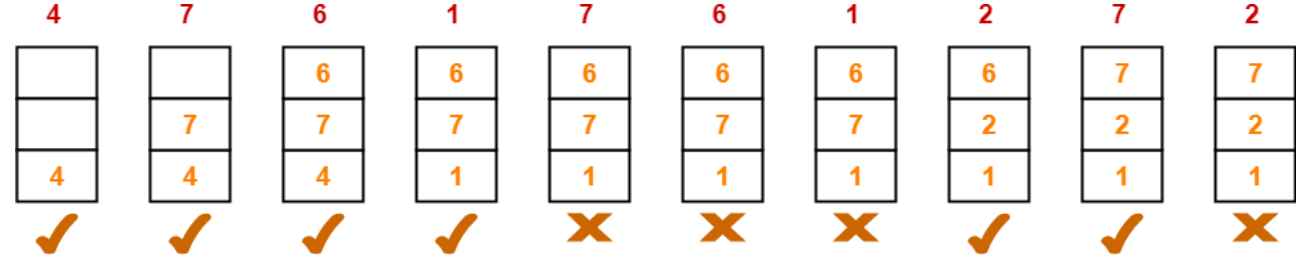
Example:

- A system uses 3 page frames for storing process pages in main memory.
- It uses the First in First out (FIFO) page replacement policy.
- Assume that all the page frames are initially empty.
- What is the total number of page faults that will occur while processing the page reference string given below-

  4 , 7, 6, 1, 7, 6, 1, 2, 7, 2
- Also calculate the hit ratio and miss ratio.

Solution-

Total number of references = 10



From here,

Total number of page faults occurred=6

Calculating Hit ratio-

Total number of page hits=Total number of references-Total number of page misses or page faults

Total number of page hits = 10 – 6 = 4

Thus,

Hit ratio = Total number of page hits / Total number of references

= 4 /10 = 0.4 or 40%

Calculating Miss ratio-

Total number of page misses or page faults = 6

Thus,

Miss ratio = Total number of page misses / Total number of references

= 6 /10 = 0.6 or 60%

Alternatively, Miss ratio

= 1 – Hit ratio

= 1 – 0.4

= 0.6 or 60%

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the necessary variables. Step

Step 3: Enter the number of frames.

Step 4: Enter the reference string end with zero.

Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen.

Step 6: When a page is brought into memory, it is inserted at the tail of the queue. Step 7: Initially all the three frames are empty.

Step 8: The page fault range increases as the no of allocated frames also increases. Step 9: Print the total number of page faults.

Step 10: Stop the program.

2. LRU Page Replacement Algorithm in OS

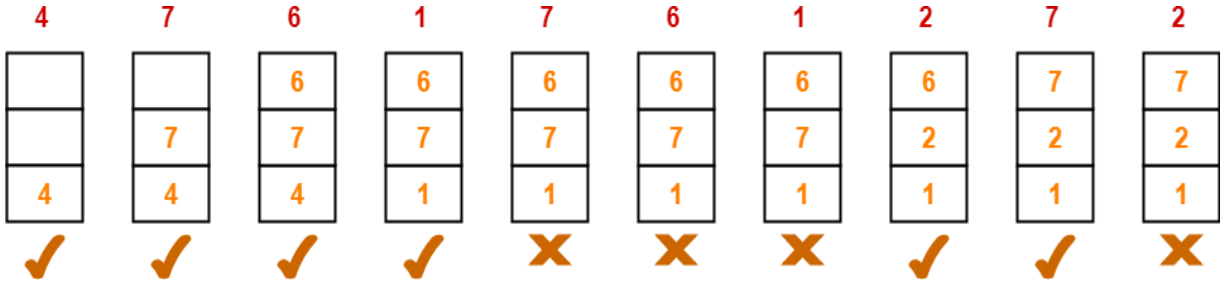
- This algorithm stands for "Least recent used" and this algorithm helps the Operating system to search those pages that are used over a short duration of time frame.
- The page that has not been used for the longest time in the main memory will be selected for replacement.
- This algorithm is easy to implement.
- This algorithm makes use of the counter along with the even-page.

Example

- A system uses 3 page frames for storing process pages in main memory.
- It uses the Least Recently Used (LRU) page replacement policy.
- Assume that all the page frames are initially empty.
- What is the total number of page faults that will occur while processing the page reference string given below-

  4 , 7, 6, 1, 7, 6, 1, 2, 7, 2
- Also calculate the hit ratio and miss ratio.

Solution-



Total number of references = 10

Total number of page faults occurred = 6
In the similar manner as above- Hit ratio = 0.4 or 40% Miss ratio = 0.6 or 60%

ALGORITHM:

Step 1: Start the process
Step 2: Declare the size
Step 3: Get the number of pages to be inserted
Step 4: Get the value
Step 5: Declare counter and stack
Step 6: Select the least recently used page by counter value
Step 7: Stack them according the selection.
Step 8: Display the values
Step 9: Stop the process

3. Optimal Page Replacement Algorithm

- This algorithm mainly replaces the page that will not be used for the longest time in the future.
- The practical implementation of this algorithm is not possible.
- Practical implementation is not possible because we cannot predict in advance those pages that will not be used for the longest time in the future.
- This algorithm leads to less number of page faults and thus is the best- known algorithm
- Also, this algorithm can be used to measure the performance of other algorithms.

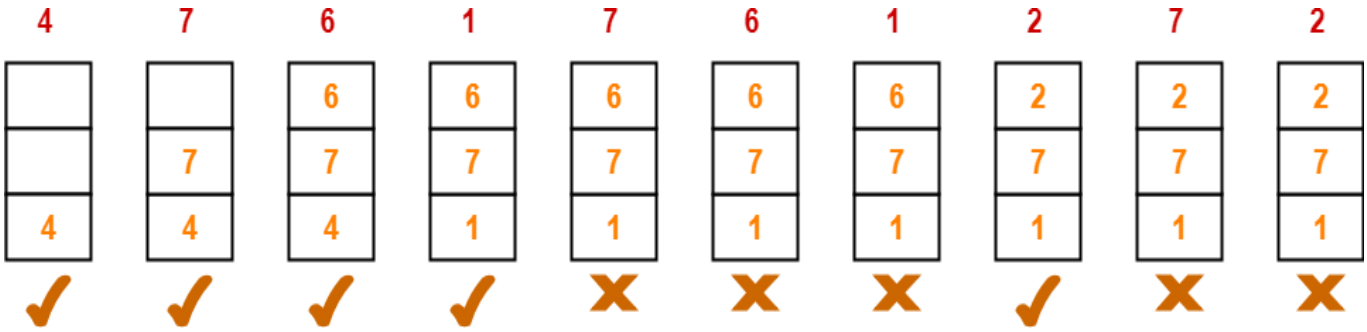
Example:

- A system uses 3 page frames for storing process pages in main memory.
- It uses the Optimal page replacement policy.
- Assume that all the page frames are initially empty.
- What is the total number of page faults that will occur while processing the page reference string given below-
  4 , 7, 6, 1, 7, 6, 1, 2, 7, 2
- Also calculate the hit ratio and miss ratio.

Solution-

Total number of references = 10
Total number of page faults occurred = 5 In the similar manner as above-

Hit ratio = 0.5 or 50%
Miss ratio = 0.5 or 50%



ALGORTHIM:

Step 1. Start Program
Step 2. Read Number Of Pages And Frames
Step 3. Read Each Page Value
Step 4. Search For Page In The Frames
Step 5. If Not Available Allocate Free Frame
Step 6. If No Frames Is Free Replace The Page With The Page That Is Leastly Used
Step 7. Print Page Number Of Page Faults
Step 8. Stop process.

Conclusion: Thus, we have studied and implemented FIFO, LRU and Optimal Page Replacement Algorithm

Program Code

FIFO

```
import java.io.*;
public class FIFO
{
   public static void main(String[] args) throws IOException
   {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      int frames, pointer = 0, hit = 0, fault = 0,ref_len;
      int buffer[];
      int reference[];
      int mem_layout[][];

      System.out.println("Please enter the number of Frames: ");
      frames = Integer.parseInt(br.readLine());
      System.out.println("Please enter the length of the Reference string: ");
      ref_len = Integer.parseInt(br.readLine());

      reference = new int[ref_len];
      mem_layout = new int[ref_len][frames];
      buffer = new int[frames];
      for(int j = 0; j < frames; j++)
            buffer[j] = -1;

      System.out.println("Please enter the reference string: ");
      for(int i = 0; i < ref_len; i++)
      {
         reference[i] = Integer.parseInt(br.readLine());
      }
      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++;
         break;
        }
       }
       if(search == -1)
       {
        buffer[pointer] = reference[i];
```

```
          fault++;
          pointer++;
          if(pointer == frames)
           pointer = 0;
         }
            for(int j = 0; j < frames; j++)
                mem_layout[i][j] = buffer[j];
        }
            for(int i = 0; i < frames; i++)
        {
            for(int j = 0; j < ref_len; j++)
                System.out.printf("%3d ",mem_layout[j][i]);
            System.out.println();
        }
                System.out.println("The number of Hits: " + hit);
        System.out.println("Hit Ratio: " + (float)((float)hit/ref_len));
        System.out.println("The number of Faults: " + fault);
    }
}
```

**********OUTPUT**********

```
C:\Users\R.R.B\Desktop>javac FIFO.java
C:\Users\R.R.B\Desktop>java FIFO
Please enter the number of Frames:
3
Please enter the length of the Reference string:
10
Please enter the reference string:
4
7
6
1
7
6
1
2
7
2

  4   4   4   1   1   1   1   1   1   1
 -1   7   7   7   7   7   7   2   2   2
 -1  -1   6   6   6   6   6   6   7   7
The number of Hits: 4
Hit Ratio: 0.4
The number of Faults: 6
C:\Users\R.R.B\Desktop>
```

Program Code

LRU

```
import java.io.*;
import java.util.*;
public class LRU {
   public static void main(String[] args) throws IOException
   {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      int frames,pointer = 0, hit = 0, fault = 0,ref_len;
      Boolean isFull = false;
      int buffer[];
      ArrayList<Integer> stack = new ArrayList<Integer>();
      int reference[];
      int mem_layout[][];

      System.out.println("Please enter the number of Frames: ");
      frames = Integer.parseInt(br.readLine());
      System.out.println("Please enter the length of the Reference string: ");
      ref_len = Integer.parseInt(br.readLine());

      reference = new int[ref_len];
      mem_layout = new int[ref_len][frames];
      buffer = new int[frames];
      for(int j = 0; j < frames; j++)
            buffer[j] = -1;

      System.out.println("Please enter the reference string: ");
      for(int i = 0; i < ref_len; i++)
      {
         reference[i] = Integer.parseInt(br.readLine());
      }
      System.out.println();
      for(int i = 0; i < ref_len; i++)
      {
         if(stack.contains(reference[i]))
         {
          stack.remove(stack.indexOf(reference[i]));
         }
         stack.add(reference[i]);
         int search = -1;
         for(int j = 0; j < frames; j++)
         {
            if(buffer[j] == reference[i])
            {
               search = j;
```

```
                hit++;
                break;
            }
        }
        if(search == -1)
        {
         if(isFull)
         {
          int min_loc = ref_len;
                for(int j = 0; j < frames; j++)
                {
                 if(stack.contains(buffer[j]))
                    {
                        int temp = stack.indexOf(buffer[j]);
                        if(temp < min_loc)
                        {
                            min_loc = temp;
                            pointer = j;
                        }
                    }
                }
         }
            buffer[pointer] = reference[i];
            fault++;
            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 < frames; i++)
    {
        for(int j = 0; j < ref_len; j++)
            System.out.printf("%3d ",mem_layout[j][i]);
        System.out.println();
    }

    System.out.println("The number of Hits: " + hit);
    System.out.println("Hit Ratio: " + (float)((float)hit/ref_len));
    System.out.println("The number of Faults: " + fault);
}
```

}
**********OUTPUT**********
C:\Users\R.R.B\Desktop>javac LRU.java

C:\Users\R.R.B\Desktop>java LRU
Please enter the number of Frames:
3
Please enter the length of the Reference string:
10
Please enter the reference string:
4
7
6
1
7
6
1
2
7
2

```
 4   4   4   1   1   1   1   1   1   1
-1   7   7   7   7   7   7   2   2   2
-1  -1   6   6   6   6   6   6   7   7
```

The number of Hits: 4
Hit Ratio: 0.4
The number of Faults: 6

C:\Users\R.R.B\Desktop>

Program Code

Optimal

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class OptimalReplacement {

   public static void main(String[] args) throws IOException
   {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      int frames, pointer = 0, hit = 0, fault = 0,ref_len;
      boolean isFull = false;
      int buffer[];
      int reference[];
      int mem_layout[][];

      System.out.println("Please enter the number of Frames: ");
      frames = Integer.parseInt(br.readLine());

      System.out.println("Please enter the length of the Reference string: ");
      ref_len = Integer.parseInt(br.readLine());

      reference = new int[ref_len];
      mem_layout = new int[ref_len][frames];
      buffer = new int[frames];
      for(int j = 0; j < frames; j++)
            buffer[j] = -1;

      System.out.println("Please enter the reference string: ");
      for(int i = 0; i < ref_len; i++)
      {
         reference[i] = Integer.parseInt(br.readLine());
      }
      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++;
         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];
 fault++;
 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 < frames; i++)
        {
            for(int j = 0; j < ref_len; j++)
                System.out.printf("%3d ",mem_layout[j][i]);
            System.out.println();
        }

        System.out.println("The number of Hits: " + hit);
        System.out.println("Hit Ratio: " + (float)((float)hit/ref_len));
        System.out.println("The number of Faults: " + fault);
    }

}
```

**********OUTPUT**********

```
C:\Users\R.R.B\Desktop>javac OptimalReplacement.java
C:\Users\R.R.B\Desktop>java OptimalReplacement
Please enter the number of Frames:
3
Please enter the length of the Reference string:
10
Please enter the reference string:
4
7
6
1
7
6
1
2
7
2

  4   4   4   1   1   1   1   2   2   2
 -1   7   7   7   7   7   7   7   7   7
 -1  -1   6   6   6   6   6   6   6   6
The number of Hits: 5
Hit Ratio: 0.5
The number of Faults: 5

C:\Users\R.R.B\Desktop>
```