SHARADCHANDRA PAWAR COLLEGE OF ENGINEERING, PUNE

DEPARTMENT OF COMPUTER ENGINEERING

SYSTEM PROGRAMMING AND OPERATING SYSTEM

LABORATORY MANUAL

A.Y. 2024 - 25

SEMESTER-I

Subject Code: -310243

TEACHING SCHEME

EXAMINATION SCHEME

Practical:4Hrs/Week

Practical Assessment: 25 Marks

Term Work: 25 Marks

Name of Faculty: - Prof. Phapale K. S.

GROUP – A

EXPERIMENT NO : 01

Title:

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..

Prerequisite:

- Basic Data Structure in Java.
- Concepts of Assembler.

Software Requirements:

- Eclipse SDK

Tools/Framework/Language Used:

- Java

Hardware Requirement:

- PIV, 2GB RAM, 500 GB HDD.

Learning Objectives:

To interpret the data structures required in pass-I and pass-II and implementation of a two-pass assembler.

Outcomes:

After completion of this assignment students can :

- Understand various data structures used in Two pass Assembler
- Implement two pass assembler for pseudo-machine.

Theory Concepts:

Assembler is a System program which is used to translate program written in Assembly Language into machine language (Fig1.1). The translated program is called as object program.

Fig. 1.1 Assembler

Two Pass Assembler:

It handles forward reference problem easily (Fig 1.2). Pass-I: (Analysis)

- Symbols are entered in the Symbol table Mnemonics and the corresponding opcodes are stored in table called Mnemonic table
- Perform LC Processing
- Generate Intermediate code

Pass-II: (Synthesis)

- Synthesis the target form using the address information found in Symbol table.
- First pass constructs an Intermediated Representation (IR) of the source program for use by the second pass.

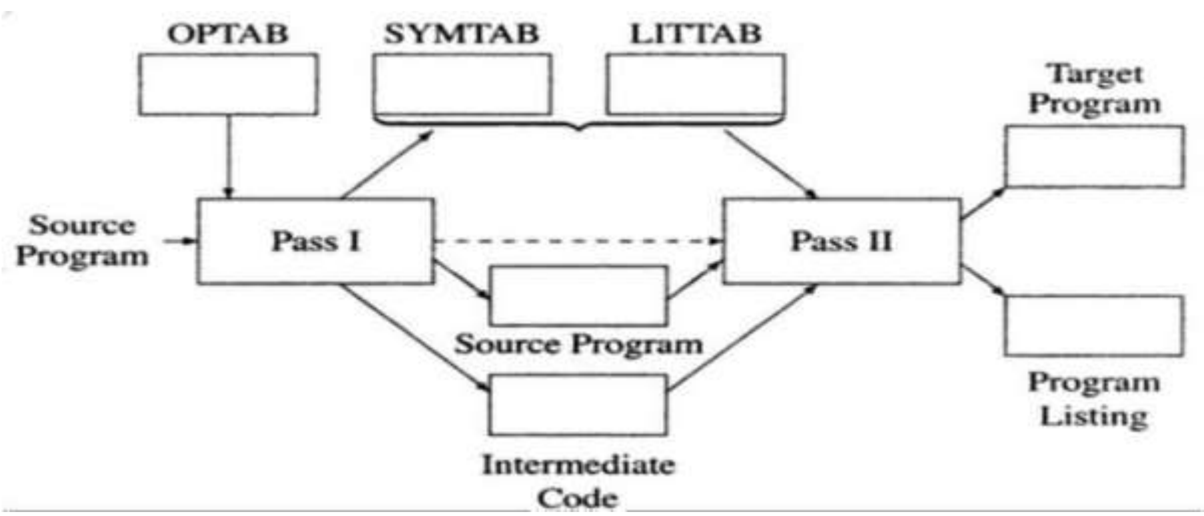


Fig 1.2: Pass-I and Pass-II Assembler

Pass-I uses the following data structures:

1. OPTAB: - A table of mnemonic opcodes and related info.

2. SYMTAB: -Symbol Table.
3. LITTAB: -A table of literals used in the program
4. POOLTAB: - contains the literal no. of starting literal of each literal pool.

OPTAB: It contains the fields mnemonic, class and mnemonic opcode. The _class' field indicates whether the Opcode corresponds to an imperative statement (IS), a declaration statement (DL) or an assembler directive (AD).If an imperative statement is present, then the mnemonic info field contains the pair (machine opcode, instruction length) else it contains the pair id of a routine to handle the declaration or directive statement.

Mnemonic	TYPE	OP-Code
STOP	IS	00
ADD	IS	01
SUB	IS	02
MUL	IS	03
MOVER	IS	04
MOVEM	IS	05
COMP	IS	06
BC	IS	07
DIV	IS	08
Mnemonic	TYPE	OP-Code
READ	IS	09
PRINT	IS	10
DC	DL	01
DS	DL	02
START	AD	01
END	AD	02
ORIGIN	AD	03
EQU	AD	04
LTORG	AD	05

SYMTAB: It contains the fields address and length. The processing of an assembly statement begins with the processing of its label field. If it contains a symbol, the symbol and the value in LC is copied into a new entry of SYMTAB. If it is an imperative statement, then length of the machine instruction is simply added to the LC. The length is also entered into the symbol table.

LITTAB and POOLTAB: Literal table stores the literals used in the program and POOLTAB stores the pointers to the literals in the current literal pool.

Algorithm for Pass-I:

- loc_cntr := 0; (default value) pooltab_ptr :=1; POOLTAB[1]:=1; littab_ptr:=1;
- While next statement is not an END statement
- If label is present then

  {

  this_label:= symbol in label field; Enter(this_label, loc_cntr) in SYMTAB.

  }
- If an LTORG statement then

  {
- Process literals LITTAB[POOLTAB[pooltab_ptr]…LITTAB[lit_tab_ptr-1] to allocate memory and put the address in the address field. Update location counter accordingly.
- pooltab_ptr := pooltab_ptr +1;
- POOLTAB[pooltab_ptr]:=littab_ptr;

  }
- If START or ORIGIN statement then

  {

  loc_cntr := value specified in the operand field;

  }
- If an EQU statement then

  {
- this_addr := value of <address_spec>;
- Correct the symbtab entry for this_label to (this_label,this_addr).

  }
- If a declaration statement then

  {
- code:= code of the declaration statement;
- size := size of memory are required by DC/DS
- loc_cntr := loc_cntr + size;
- Generate IC _(DL, code)…‘

  }

- If an imperative statement then

  {
- code:= machine opcode from OPTAB;
- loc_cntr := loc_cntr + instruction length from OPTAB;
- If operand is a literal then

  {
- this_literal := literal in operand field; LITTAB[littab_ptr]:= this_literal; littab_ptr= littab_ptr +1;

  }

  else (i.e. operand is a symbol)

  {

  this_entry := SYMTAB entry number of operandGenerate IC

  ‗(IS,code)(S,this_entry)‘;

  }

3.

- Perform step 2(b).
- Generate IC‘(AD, 02)‘.
- Go to Pass II.

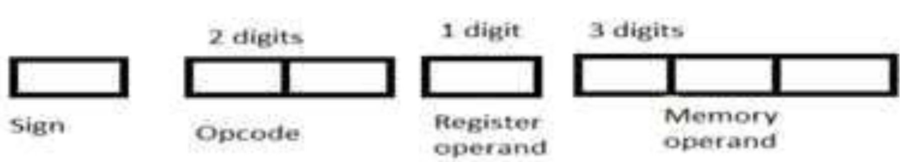
Pass-II: (Synthesis)

PASS-II takes intermediate code as an input from PASS-I and converts it into equivalent machine code. Pass-II uses the following data structures:

- SYMTAB: -Symbol Table.
- LITTAB: -A table of literals used in the program
- Intermediate Code

Pass-II reads each instruction in intermediate code and converts it into machine language instruction.

Machine code format is:



Ex : + 09 0 113

Algorithm of Pass II Assembler

It has been assumed that the target code is to be assembled in the area named code_area.

- Code_area_adress= address of code_area; Pooltab_ptr=1; Loc_cntr=0;
- While next statement is not an END statement
- Clear machine_code_buffer;
- If an LTORG statement
- Process literals in LITTAB and assemble the literals in machine_code_buffer.
- Size= size of memory area required for literals
- Pooltab_ptr=pooltab_ptr +1;
- If a START or ORIGIN statement
- Loc_cntr=value specified in operand field;
- Size=0;
- If a declaration statement

  If a DC statement then assembles the constant in machine_code_buffer;

ii) Size= size of memory area required by DC/DS;

- If an imperative statement
- Get operand address from SYMTAB or LITTAB
- Assemble instruction in machine_code_buffer;
- Size=size of instruction;

  If size≠ 0 then

- Move contents of machine_code_buffer to the addresscode_area_address + loc_cntr;
- Loc_cntr=loc_cntr+size;
- Processing end statement
- Perform steps 2(b) and 2(f)
- Write code_area into output file.

Testing:

SAMPLE PROGRAM Input

Source Code

START 200 MOVER

AREG =‘5’ MOVEM AREG A

LOOP MOVER AREG

AMOVER CREG B
ADD

CREG =‘1’LTORG NEXT1

SUB

AREG =‘1’ ORIGIN
LOOP

+6

MUL

CREG BA DS 2

B DC ‘3’ NEXT2

QU LOOP END

EXPECTED OUTPUT:POOLTA

Index	PoolNo
1	1
2	3

SymNo	SYMBOL	ADDRESS	SIZE
01	A	209	2
02	LOOP	202	1
03	B	211	1
04	NEXT1	207	1
05	NEXT2	202	1

LitNo	LITERAL	ADDRESS
01	=‘5’	205
02	=‘1’	206
03	=‘1’	212

PROGRAM:-

**************************Pass 1 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+"\t"+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\Lalita Kshirsagar\OneDrive\Desktop\BE COMP>javac pass1.java

C:\Users\Lalita Kshirsagar\OneDrive\Desktop\BE COMP>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\Lalita Kshirsagar\OneDrive\Desktop\BE COMP>
```

********************************** Pass 2 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

Conclusion :-

Questions:

1. **What is an assembler along with basic functions of it?**

2. **What is cross assembler?**

3. **What are various advanced assembler directives?**

4. **What is Forward Referencing? How to solve it in two pass assembler??**

EXPERIMENT NO : 02

Title:

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

Prerequisite:

- Basic Data Structure in Java.
- Concepts of macro-processor.

SOFTWARE REQUIREMENTS:

- Eclipse SDK

Tools/Framework/Language Used:

- Java

HARDWARE REQUIREMENTS:

- PIV, 2GB RAM, 500 GB HDD.

Learning Objectives:

To interpret the data structures and implement pass-I and pass-II two- pass macro- processor.

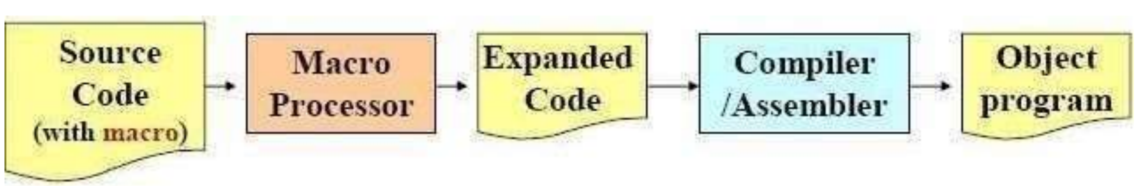
Outcomes:

Students will be able to:

- Understand various data structures used in Two pass macro-processor
- Implement two pass macro-processor for pseudo-machine.

THEORY:

Macro-Processor is a system program used to identify the macro call and performing macro expansion.



Features of macro processor:

1. Recognize the macro definition
2. Save macro definition
3. Recognize the macro call
4. Perform macro expansion

Typically, MACRO is defined at start of program or at end of program.

Macro Definition Syntax: -

1) Macro header: - It contains keyword ‗MACRO‘.

2) Macro prototype statement syntax :-< Macro Name > [& < Formal Parameters >]

3) Model Statements: - It contains 1 or more simple assembly statements,which will replace MACRO CALL, while macro expansion.

4) MACRO END MARKER: - It contains keyword ‗MEND‘.

MACRO CALL :-

< MACRO NAME > [<ACTUAL Parameters >]

Example of

```
MACRO
  INCR & MEM, & VAL, &
   RMOVER &R, & MEM ADD &
              R, & VAL
   MOVEM & R, & MEM
     MEND START 300
  INCR  A,  B,BREGSTOP
  A      DS     1
  B      DS     1
  END
```

Macro Expansion:-

Macro expansion is the task of replacing macro call by statements from macro body.

Example of MACRO Expansion for above program :-

```
START 300
   MOVER BREG, A ADD
 BREG, B MOVEM BREG, A
  STOP
```

```
A       DS    1
B       DS    1
END
```

Forward reference Problem

2) The assembler specifies that the macro definition should occur anywhere in the program. So there can be chances of macro call before it‘s definition witch gives rise to the forwards reference problem of macro.

Due to which macro is divided into two passes:

1. **PASS 1**-Recoganize macro definition save macro definition
2. **PASS 2**-Recoganize macro call perform macro expansion

Pass 1 data bases:

1. The input macro source deck.
2. The output macro source deck copy for use by pass 2.
3. The macro definition table (MDT), used to store the body of the macrodefinitions.
4. The macro name table (MNT), used to store the names of defined macros.
5. The macro definition table counter (MDTC), used to indicate the next availableentry in the MDT.
6. The macro name table counter (MNTC), used to indicate the next availableentry in the MNT.
7. The argument list array (ALA).

2. **Algorithm for Pass 1 of Macro Processor**

Processing Macro Definitions

1. Initialize MDTC and MNTC.
2. Read the next source statement of the program.
3. If the statement contains MACRO pseudo-op. go to Step 6.
4. Output the instruction of the program.
5. If the statement contains END pseudo-op, go to Pass 2, else go to Step 2.
6. Read the next source statement of the program.

7. Make an entry of the macro name and MTDC into MNT at locationMNTC andincrement the MNTC by 1.
8. Prepare the parameter (arguments) list array.
9. Enter the macro name into the MDT and increment the MTDC by 1.
10. Read the next card and substitute index for the parameters (arguments).\
11. Enter the statement into the MDT and increment the MDT by 1. If
12. MEND pseudo-op found, go to Step 2, else go to Step 10.

Pass 2 data bases:

a. The copy of the input macro source deck.
b. The output expanded source deck to be used as input to the assembler.
c. The macro definition table (MDT), created in pass1.
d. The macro name table (MNT), created in pass1.
e. The macro definition table pointer (MDTP), used to indicate the next line of text to be used during macro expansion
f. The argument list array (ALA), used to substitute macro call argumentsfor the index markers in the stored macro definition.

Algorithm for Pass 2 of Macro Processor – Processing for Calls and Expansion of Macro

1. Read the next source statement copied bypass 1.
2. Search into the MNT for a record and evaluate the operation code.
3. If the operation code has a macro name, go to Step 5.
4. Write the statement to the expanded source file.
5. If END pseudo-op found, pass the entire expanded code to theassembler for assembling and stop. Else go to Step 1.
6. Update the MDTP to the MDT index from the MNT entry.
7. Prepare the parameter (argument) list array.

8. Increment the MDTP by 1.
9. Read the statement from the current MDT and substitute actual parameters(arguments) from the macro call.
10. If the statement contains MEND pseudo-op, go to Step 1, elsewrite the expanded source code and go to Step 8.

SAMPLE PROGRAM Input

MACRO INCR &M,&NMOVEM
&N, &M ADD &N, &M
MEND
MACRO ADDITION &X, &Y,&AREG
ADD &AREG, &XSUB
&AREG, &Y
MEND
START
MOVER BREG, A
ADD BREG, B

ADDITION A, B, CREG
MOVEM BREG, A
INCR B, CREG STOP
A DS 1
B DS 1
END

Expected Output:

START
MOVER BREG, A

ADD BREG, B
ADD CREG,
A SUB CREG, B
MOVEM BREG,
A **MOVEM**
CREG, BADD
CREG, B STOP

A DS 1
B DS 1
END

MDT

Index	Card
0	**MACRO INCR &M,&N**
1	MOVEM #1, #0
2	ADD #1, #0
3	**MEND**
4	**MACRO ADDITION &X,&Y,&AREG**
5	ADD #2, #0
6	SUB #2, #1
7	**MEND**

MNT

S. No.	Name	DT index
0	INCR	0
1	ADDITION	4

Index	Argument
0	B
1	CRFG

Index	Argument
0	A
1	B
2	CRFG

Program:

```
#Static Table
OPCODE_TABLE = {
"HALT":'00',
"ADD":'01',
"SUB":'02',
"MULT":'03',
"MOVER":'04',
"MOVEM":'05',
"COMP":'06',
"BC":'07', #JUMP
"DIV":'08',
"READ":'09',
"PRINT":'10'
}
REGISTER_TABLE = {
"AREG":'1',
"BREG":'2',
"CREG":'3',
"DREG":'4'
}
CONDITIONALS =
```

```
{ "LT" :'1',
"LE" :'2',
"GT" :'3',

"GE" :'4',
"EQ" :'5',
"ANY":'6'
}
ASSEMBLER_DIR = {
"START" : 'NULL',
"END": 'NULL',
}
DECLARATIVES = {
"DS" : 'NULL',
"DC" : 'NULL'
}
#Dynamic Tables
SYMBOL_TABLE = [[],[]]
LITERAL_TABLE = {}
def CHECK(word):
'''CHECKS IF THE WORD IS A REGISTER/CONDITIONAL/SYMBOL.'''
if word in REGISTER_TABLE:
return REGISTER_TABLE[word]
elif word in CONDITIONALS:
return CONDITIONALS[word]
elif word[0] == '=' :
if word in LITERAL_TABLE:
return LITERAL_TABLE[word]
else:

LITERAL_TABLE[word] = "L"+str((len(LITERAL_TABLE)+1))
return LITERAL_TABLE[word]
else:
#If present return
if word in SYMBOL_TABLE[0]:
idx = SYMBOL_TABLE[0].index(word)
return SYMBOL_TABLE[1][idx]
else: SYMBOL_TABLE[0].append(word)
SYMBOL_TABLE[1].append("S"+str((len(SYMBOL_TABLE[0])+1)))
return SYMBOL_TABLE[1][-1]
LC = 000
with open(r"code.txt") as f, open(".output1.txt", "w+") as out: for
line in f:
line = line.strip('\n').split(' ')
IC = ["" for _ in range(len(line))]
# if line[0][1]== ':' :
# print()
# print(*line, sep='\t') #
else:
# print("\n ",*line, sep='\t') #If
first word is a LABEL
if line[0][-1] == ':' :
```

```
SYMBOL_TABLE[0].append(line[0][:-1])
SYMBOL_TABLE[1].append(LC)
line.pop(0)
#If first word is an opcode
if line[0] in OPCODE_TABLE:
LC+=1

IC[0] = OPCODE_TABLE[line[0]]
#To check HALT opcode as length is 1 if
len(line) > 1:
IC[1] = CHECK(line[1])
if len(line) == 3:
IC[2] = CHECK(line[2])
IC.insert(0,LC)
#print(*IC, sep='\t')
print(*IC, sep='\t', file = out) #Else
if Assembler Directive elif line[0]
in ASSEMBLER_DIR: if line[0]
== 'START':
if len(line) == 1:
LC = 0
else:
LC = int(line[1]) - 1
#To avoid index out of range. if
len(line) == 3:

#For declartive Statements if
line[1] in DECLARATIVES:
LC+=1
if line[0] in SYMBOL_TABLE[0]:
idx = SYMBOL_TABLE[0].index(line[0])
SYMBOL_TABLE[1][idx] = LC
else:
SYMBOL_TABLE[0].append(line[0])
SYMBOL_TABLE[1].append(LC)
if line[0] == 'ORIGIN':
LC = int(line[1]) - 1
print("\n\nSYMBOL_TABLE = ", SYMBOL_TABLE)
print('------------------------------------------------------------
-----------------------------------')
print("LITERAL_TABLE = ", LITERAL_TABLE)
print('------------------------------------------------------------
------------------------------------')
print("LC = ", LC)
print('------------------------------------------------------------
------------------------------------')
```

CODE.TXT

```
START 100
SYM1 DS 1
ADD AREG SYM1
SUB BREG ='10'
LABEL1: DIV CREG N
```

READ SYM1
PRINT N
PRINT A
PRINT B
PRINT C
BC ANY LABEL1
SUB BREG ='10'
SUB BREG ='20'
SUB BREG ='30'
SUB BREG ='40'
ORIGIN 300
PRINT SYM1
N DS 1
A DS 1
B DS 1
C DS 1
READ N
HALT
END
OUTPUT
SYMBOL_TABLE = [[], []]

LITERAL_TABLE = {}

-------------- LC
= 99

SYMBOL_TABLE = [['SYM1'], [100]]

LITERAL_TABL={}
-------------- LC
= 100

SYMBOL_TABLE = [['SYM1'], [100]]

LITERAL_TABLE = {}

-------------- LC
= 101

SYMBOL_TABLE = [['SYM1'], [100]]

LITERAL_TABLE = {"='10'": 'L1'}

-------------- LC
= 102

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N'], [100, 102, 'S4']

LITERAL_TABLE = {"='10'": 'L1'}

-------------- LC
= 103

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N'], [100, 102, 'S4']]

LITERAL_TABLE = {"='10'": 'L1'}

- LC = 104

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N'], [100, 102, 'S4']]

LITERAL_TABLE = {"='10'": 'L1'}

-------------- LC
= 105

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A'], [100, 102, 'S4', 'S5']]

LITERAL_TABLE = {"='10'": 'L1'}

-------------- LC
= 106

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B'], [100, 102, 'S4', 'S5', '
S6']]

LITERAL_TABLE = {"='10'": 'L1'}

-------------- LC
= 107

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1'}

LC= 108

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1'}

-------------- LC
= 109

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1'}

-------------- LC
= 110

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2'}

-------------- LC
= 111

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3'}

-------------- LC
= 112

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}

-------------- LC
= 113

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}

-------------- LC
= 299

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 'S4', 'S
5', 'S6', 'S7']]

LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}

-------------- LC
= 300

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 301, 'S5
', 'S6', 'S7']]


```
LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}
----------------------------------------------------------------------------
-------------- LC
= 301
----------------------------------------------------------------------------
--------------

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 301, 302
, 'S6', 'S7']]
----------------------------------------------------------------------------
-------------
LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}
----------------------------------------------------------------------------
-------------- LC
= 302
----------------------------------------------------------------------------


SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 301, 302
, 303, 'S7']]
----------------------------------------------------------------------------
-------------
LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}
----------------------------------------------------------------------------
-------------- LC
= 303
----------------------------------------------------------------------------
--------------

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 301, 302
, 303, 304]]
----------------------------------------------------------------------------
-------------
LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}
----------------------------------------------------------------------------
-------------- LC
= 304
----------------------------------------------------------------------------
--------------

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 301, 302
, 303, 304]]
----------------------------------------------------------------------------
-------------
LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}
----------------------------------------------------------------------------
-------------- LC
```

= 305

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 301, 302
, 303, 304]]

LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}

-------------- LC
= 306

SYMBOL_TABLE = [['SYM1', 'LABEL1', 'N', 'A', 'B', 'C'], [100, 102, 301, 302
, 303, 304]]

LITERAL_TABLE = {"='10'": 'L1', "='20'": 'L2', "='30'": 'L3', "='40'": 'L4'
}

-------------- LC
= 306

Conclusion:-

Questions:-

1. **Distinguish between macro and a subroutine?**

2. **Define and Distinguish between parameters that can be used in macros?**

3. **What is advantage of Macro definition?**

GROUP - B

EXPERIMENT NO: 03

Title:

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

Prerequisite:

- Basic Data Structure in Java.
- Concepts of scheduling.

Software Requirements:

- Eclipse SDK

Tools/Framework/Language Used:

- Java.

Hardware Requirement:

- PIV, 2GB RAM, 500 GB HDD.

Learning Objectives:

Understand the concept of scheduling algorithm.

Theory Concepts:

CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will becarried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution. Some process scheduling algorithms are –

1. First-Come, First-Served (FCFS) Scheduling
2. Shortest-Job- First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling

These algorithms are either **non-preemptive or preemptive.**

Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

CPU Scheduling Criteria

A CPU scheduling algorithm tries to maximize and minimize the following:

Maximize:

CPU utilization: CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low- level and 90 percent for the high-level system.

Throughput: The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

Minimize:

Waiting time: Waiting time is an amount that specific process needs to wait in the ready queue.

Waiting Time=Turnaround Time **-** CPU Time

Response time: It is an amount to time in which the request was submitted until the first response is produced.

Turnaround Time: Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

Turnaround Time= Finish Time - Arrival Time

First Come First Serve (FCFS)

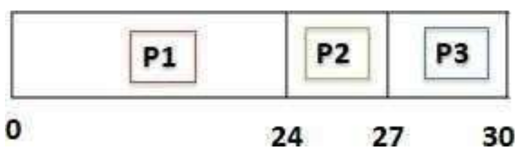
Jobs are executed on first come, first serve basis. It is a non-preemptive scheduling algorithm. Easy to understand and implement. Its implementation is based on FIFO queue. Poor in performance as average wait time is high.

Example of First Come First Serve Scheduling Algorithm

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3

The **Gantt Chart** for the schedule is:



Average Turnaround Time: 81/3=17
Average waiting time: 51/3 =17
Shortest Job First (SJF)

SJF 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.

Example of Shortest Job First Scheduling Algorithm

Process	CPU Time	Arrival Time	End Time	Turnaround time	Waiting time
P1	7	0	16	16	9
P2	4	2	7	5	1
P3	1	4	5	1	0
P4	4	5	11	6	2
			Total	**28**	**12**

Priority Based Scheduling

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.

Shortest Remaining Time

Shortest remaining time (SRT) is the preemptive version of the SJN algorithm. The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion. Impossible to implement in interactive systems where required CPU time is not known. It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

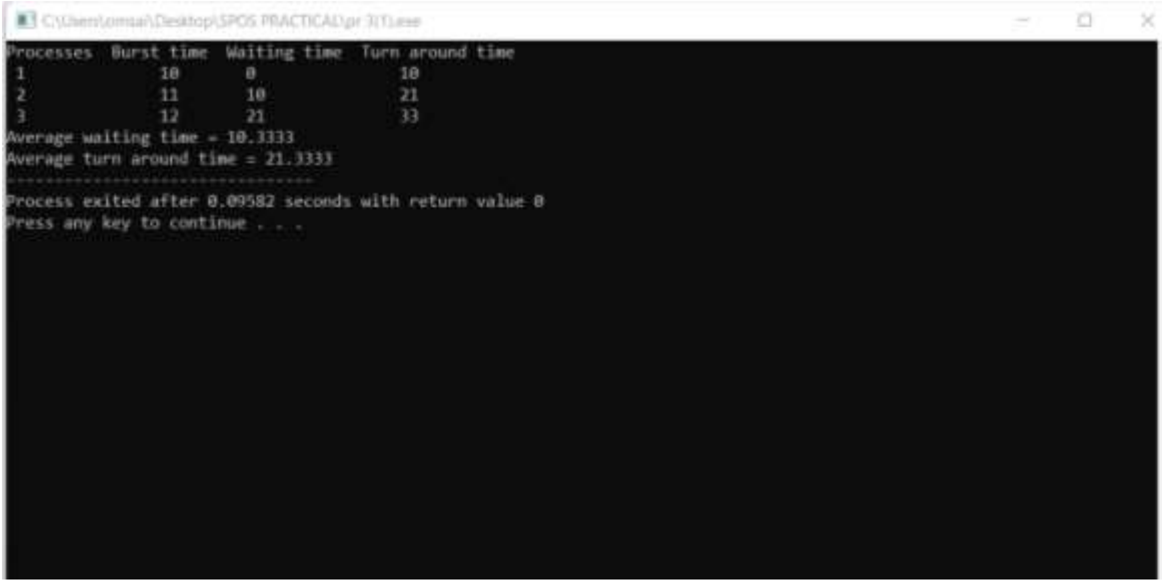
Round Robin is the preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called a quantum. Once a process is executed for a given time period, it is preempted and other process executes for a given time period. Context switching is used to save states of preempted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

Program FCFS:

```
// C++ program for implementation of FCFS
// scheduling
#include<iostream>
using namespace std;

// Function to find the waiting time for all
// processes
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
void findTurnAroundTime( int processes[], int n,
                    int bt[], int wt[], int tat[])
{
     // calculating turnaround time by adding
     // bt[i] + wt[i]
     for (int i = 0; i < n ; i++)
          tat[i] = bt[i] + wt[i];
}
//Function to calculate average time
void findavgTime( int processes[], int n, int bt[]){
     int wt[n], tat[n], 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
     cout << "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];
          cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
               << wt[i] <<"\t\t " << tat[i] <<endl;}
     cout << "Average waiting time = "
          << (float)total_wt / (float)n;
     cout << "\nAverage turn around time = "
          << (float)total_tat / (float)n;}
// Driver code
int main(){
     //process id's
  int processes[] = { 4, 5, 6};
```

```
 int n = sizeof processes / sizeof processes[0];
 //Burst time of all processes
 int burst_time[] = {10, 11,12} ;
findavgTime(processes, n, burst_time);
return 0;
}
```

Output:

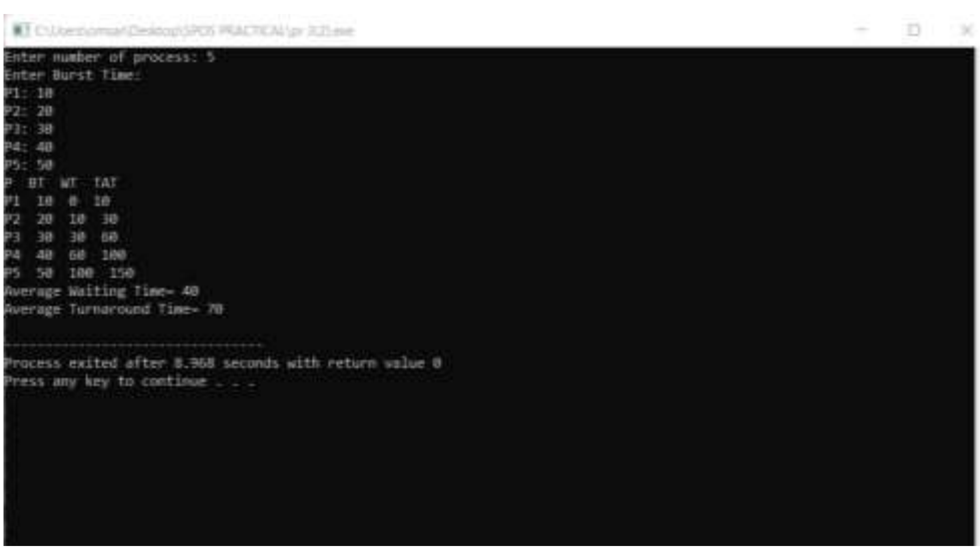


SJF(Preemptive)

```
#include <iostream>
using namespace std;
int main() {
	// Matrix for storing Process Id, Burst
	  // Time, Average Waiting Time & Average
	  // Turn Around Time.
	  int A[100][4];
	  int i, j, n, total = 0, index, temp;
	  float avg_wt, avg_tat;
	   cout << "Enter number of process: ";
	  cin >> n;
	  cout << "Enter Burst Time:" << endl;
	  // User Input Burst Time and alloting Process Id.
	  for (i = 0; i < n; i++) {
		cout << "P" << i + 1 << ": ";
		cin >> A[i][1];
		A[i][0] = i + 1;}
	// Sorting process according to their Burst Time.
	  for (i = 0; i < n; i++) {
		index = i;
		for (j = i + 1; j < n; j++)
```

```
                    if (A[j][1] < A[index][1])
                            index = j;
            temp = A[i][1];
            A[i][1] =
            A[index][1];
            A[index][1] =
            temp; temp =
            A[i][0]; A[i][0] =
            A[index][0];
            A[index][0] =
            temp;}
    A[0][2] = 0;
    // Calculation of Waiting
    Times for (i = 1; i < n; i++) {
            A[i][2] = 0;
            for (j = 0; j < i; j++)
                    A[i][2] += A[j][1];
            total       +=
   A[i][2];}    avg_wt    =
   (float)total / n; total =
   0;
    cout << "P         BT        WT       TAT" << endl;
    // Calculation of Turn Around Time and printing the
    // data.
    for (i = 0; i < n; i++) {
            A[i][3] = A[i][1] + A[i][2];
            total += A[i][3];
            cout << "P" << A[i][0] << " " << A[i][1] << "  " << A[i][2] << "  " << A[i][3] <<endl;}
  avg_tat = (float)total / n;
    cout << "Average Waiting Time= " << avg_wt << endl;
    cout << "Average Turnaround Time= " << avg_tat << endl;}
```

Output



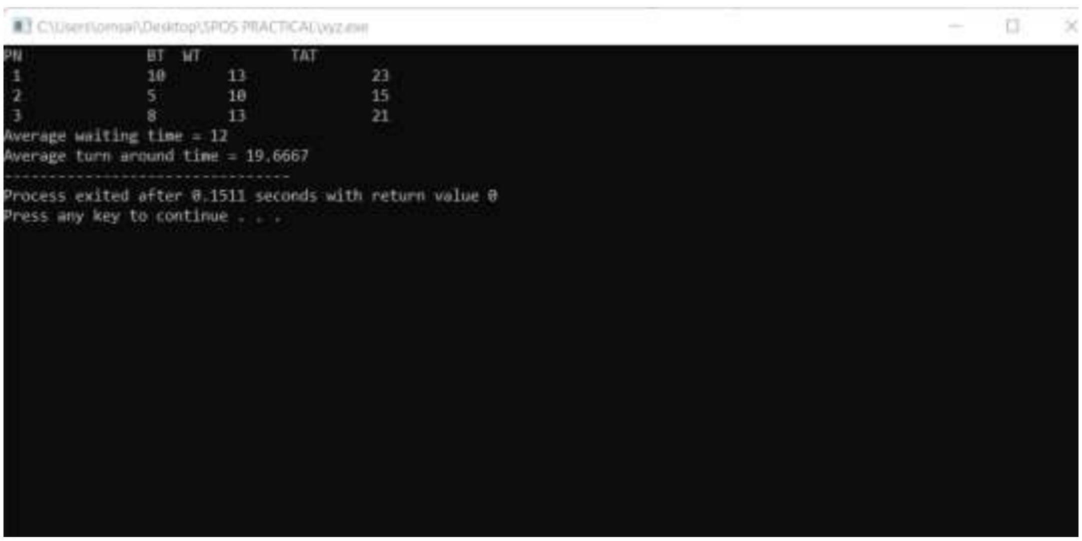
Round Robin(Preemptive)

```
// C++ program for implementation of RR scheduling
#include<iostream>
using namespace std;
// Function to find the waiting time for all
// processes

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[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 (1)
 { bool 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;  } }
// Function to calculate turn around time

void findTurnAroundTime(int processes[], int n,
      int bt[], int wt[], int tat[])
{
 // calculating turnaround time by adding
 // bt[i] + wt[i]
 for (int i = 0; i < n ; i++)
  tat[i] = bt[i] + wt[i];
}

// Function to calculate average time
void findavgTime(int processes[], int n, int bt[],
         int quantum)
{
 int wt[n], tat[n], 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
 cout << "PN\t "<< " \tBT "
  << " WT " << " \tTAT\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];
  cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
   << wt[i] <<"\t\t " << tat[i] <<endl; }
 cout << "Average waiting time = "
  << (float)total_wt / (float)n;
 cout << "\nAverage turn around time = "
  << (float)total_tat / (float)n; }
// Driver code
int main()
{
 // process id's
```

```
 int processes[] = { 1, 2, 3};
 int n = sizeof processes / sizeof processes[0];

 // Burst time of all processes
 int burst_time[] = {10, 5, 8};

 // Time quantum
 int quantum = 2;
 findavgTime(processes, n, burst_time, quantum);
 return 0;
}
```

Output:



Priority (Non-Preemptive)

```
#include <iostream>
void swap(int *a,int *b){
int temp=*a;
*a=*b;
*b=temp;}
int main(){
int n;
printf(&quot;Enter Number of Processes: &quot;);
scanf(&quot;%d&quot;,&amp;n);
int burst[n],priority[n],index[n];
for(int i=0;i&lt;n;i++){
printf(&quot;Enter Burst Time and Priority Value for Process %d: &quot;,i+1);
scanf(&quot;%d %d&quot;,&amp;burst[i],&amp;priority[i]);
index[i]=i+1;}
for(int i=0;i&lt;n;i++){
int temp=priority[i],m=i;
for(int j=i;j&lt;n;j++){
```

```
if(priority[j] &gt; temp){
temp=priority[j];
m=j;}}
swap(&amp;priority[i], &amp;priority[m]);
swap(&amp;burst[i], &amp;burst[m]);
swap(&amp;index[i],&amp;index[m]);}
int t=0;
printf(&quot;Order of process Execution is\n&quot;);
for(int i=0;i&lt;n;i++){
printf(&quot;P%d is executed from %d to %d\n&quot;,index[i],t,t+burst[i]);
t+=burst[i];}
printf(&quot;\n&quot;);
printf(&quot;Process Id\tBurst Time\tWait Time\n&quot;);

int wait_time=0;
int total_wait_time = 0;
for(int i=0;i&lt;n;i++){
printf(&quot;P%d\t\t%d\t\t%d\n&quot;,index[i],burst[i],wait_time);
total_wait_time += wait_time;
wait_time += burst[i];}
float avg_wait_time = (float) total_wait_time / n;
printf(&quot;Average waiting time is %f\n&quot;, avg_wait_time);
int total_Turn_Around = 0;
for(int i=0; i &lt; n; i++){
total_Turn_Around += burst[i];
}
float avg_Turn_Around = (float) total_Turn_Around / n;
printf(&quot;Average TurnAround Time is %f&quot;,avg_Turn_Around);
return 0;
}
```

Output:

```
Enter Number of Processes: 4
Enter Burst Time and Priority Value for Process 1: 8
8
Enter Burst Time and Priority Value for Process 2: 6
8
Enter Burst Time and Priority Value for Process 3: 5
7
Enter Burst Time and Priority Value for Process 4: 4
5
Order of process Execution is
P1 is executed from 0 to 8
P2 is executed from 8 to 14
P3 is executed from 14 to 19
P4 is executed from 19 to 23

Process Id      Burst Time      Wait Time
P1              8               0
P2              6               8
P3              5               14
P4              4               19
Average waiting time is 10.250000
Average TurnAround Time is 5.750000
--------------------------------
Process exited after 19.94 seconds with return value 0
Press any key to continue . . .
```

Conclusion:-

Questions:-

1. **What is Scheduling?**

2. What are different types of scheduling?

3. What is a preemptive and non-preemptive scheduling?

4. Different preemptive and non preemptive scheduling algorithm?

EXPERIMENT NO : 04

Title:

Write a program to simulate Memory placement strategies – best fit, first fit, next fit and worst fit.

Prerequisite:

- Basic Data Structure in Java.
- Concepts of scheduling.

Software Requirements:

- Eclipse SDK

Tools/Framework/Language Used:

- Java.

Hardware Requirement:

- PIV, 2GB RAM, 500 GB HDD.

Learning Objectives:

- Understand the concept of scheduling algorithm.

Outcomes:

- After completion of this assignment students can perform scheduling of process by preemptive and non-preemptive methods in Java.

Theory:

In the first fit, the partition is allocated which is first sufficient from the top of Main Memory.

Example :

Input : blockSize[] = {100, 500, 200, 300, 600};

processSize[] = {212, 417, 112, 426};

Output:

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

Its advantage is that it is the fastest search as it searches only the first block i.e. enough to assign a process.

It may have problems of not allowing processes to take space even if it was possible to allocate. Consider the above example, process number 4 (of size 426) does not get memory.

However it was possible to allocate memory if we had allocated using best fit allocation [block number 4 (of size 300) to process 1, block number 2 to process 2, block number 3 to

process 3 and block number 5 to process 4].

Implementation:

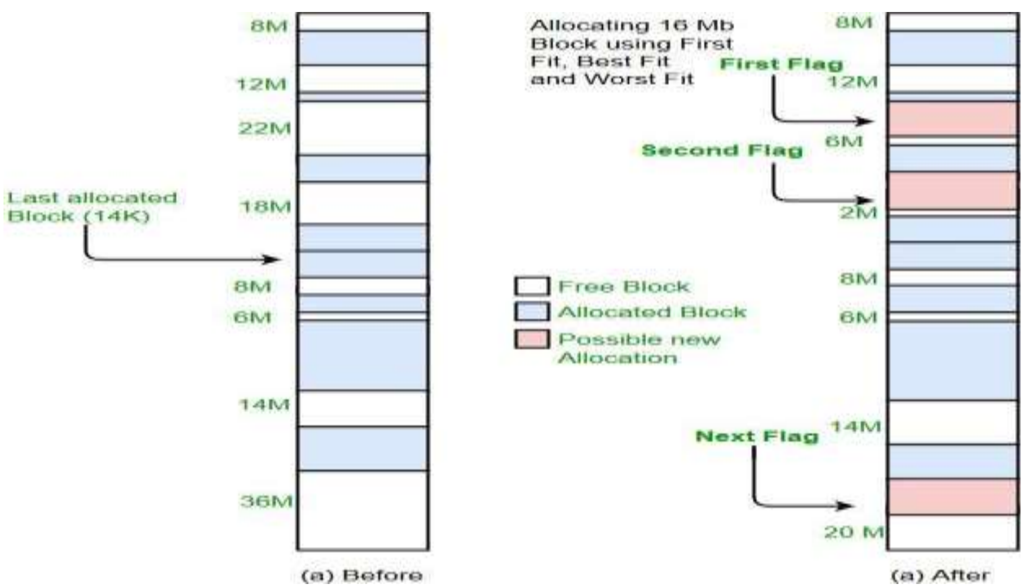
1- Input memory blocks with size and processes with size.

2- Initialize all memory blocks as free.

3- Start by picking each process and check if it can be assigned to current block.

4- If size-of-process <= size-of-block if yes then assign and check for next process.

5- If not then keep checking the further blocks.



Advantages:

It is fast in processing. As the processor allocates the nearest available memory partition to the job, it is very fast in execution.

Disadvantages:

It wastes a lot of memory. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory is wasted and many jobs may not get space in the memory, and would have to wait for another job to complete.

B. Next Fit Memory Allocation

Next fit is a modified version of ‘first fit’. It begins as the first fit to find a free partition but when called next time it starts searching from where it left off, not from the beginning. This policy makes use of a roving pointer. The pointer moves along the memory chain to search for a next fit. This helps in, to avoid the usage of memory always from the head (beginning) of the free block chain.

Program

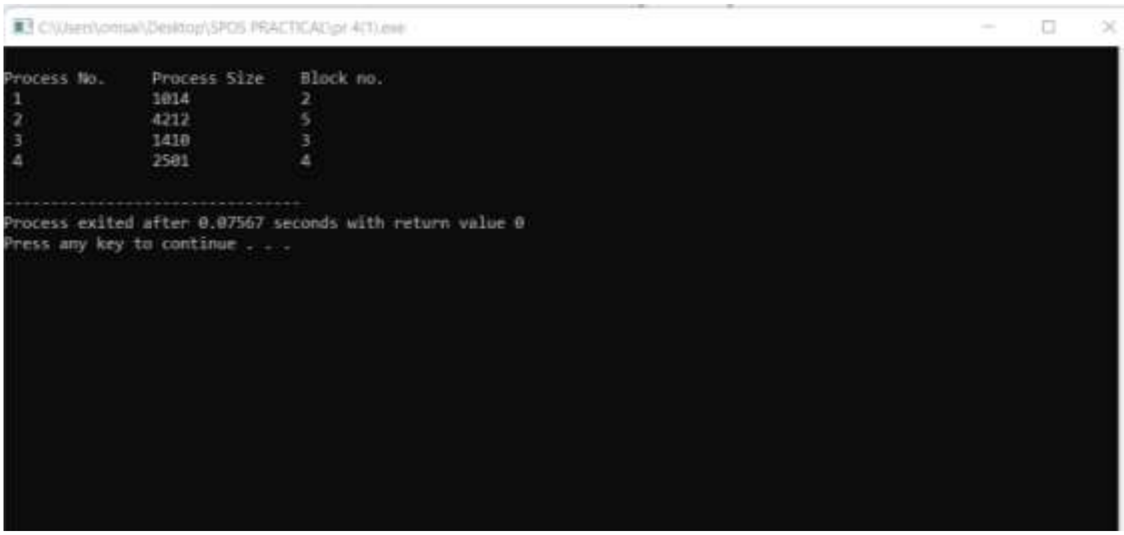
Best - Fit algorithm

```
// C++ implementation of Best - Fit algorithm

#include<iostream>

using namespace std;

// Method to allocate memory to blocks as per Best fit algorithm

void bestFit(int blockSize[], int m, int processSize[], int n)

{

// Stores block id of the block allocated to a process

int allocation[n];

// Initially no block is assigned to any process

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

allocation[i] = -1;

// pick each process and find suitable blocks
```

```
// according to its size ad assign to it
for (int i = 0; i < n; i++)
{
// Find the best fit block for current process
int bestIdx = -1;
for (int j = 0; j < m; j++)
{
if (blockSize[j] >= processSize[i])
{
if (bestIdx == -1)
bestIdx = j;
else if (blockSize[bestIdx] > blockSize[j])
bestIdx = j;
}
}
// If we could find a block for current process
if (bestIdx != -1)
{
// allocate block j to p[i] process
allocation[i] = bestIdx;
// Reduce available memory in this block.
blockSize[bestIdx] -= processSize[i];
}
```

```
}
cout << "\nProcess No.\tProcess Size\tBlock no.\n";
for (int i = 0; i < n; i++)
{
cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
if (allocation[i] != -1)
cout << allocation[i] + 1;
else
cout << "Not Allocated";
cout << endl;
}
}
// Driver Method
int main()
{
int blockSize[] = {1000, 2000, 3000, 4000, 5000};
int processSize[] = {1014, 4212, 1410,2501};
int m = sizeof(blockSize) / sizeof(blockSize[0]);
int n = sizeof(processSize) / sizeof(processSize[0]);
bestFit(blockSize, m, processSize, n);
return 0;
}
```

Output:

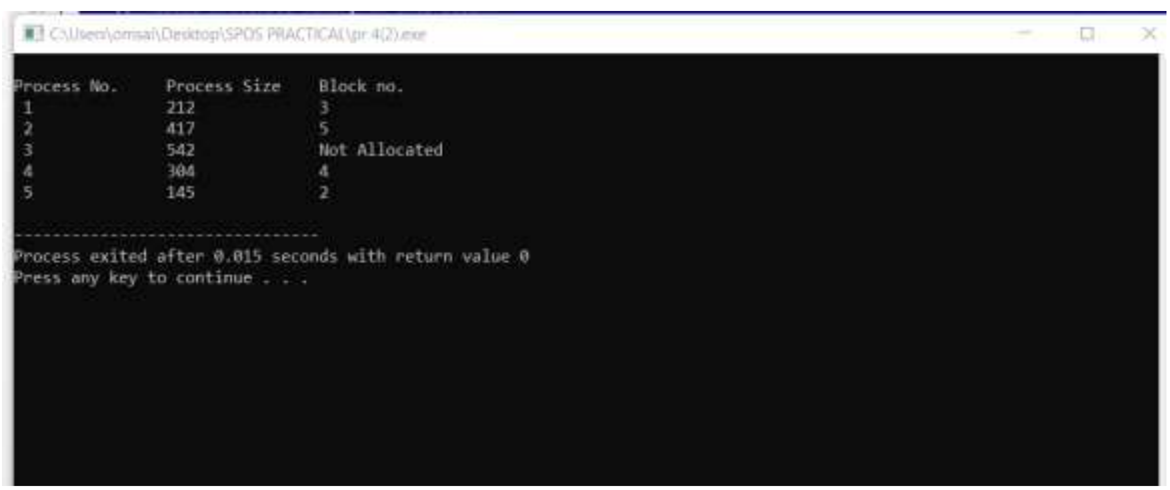


First - Fit algorithm

```
// C++ implementation of First - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

// Function to allocate memory to
// blocks as per First fit algorithm
void firstFit(int blockSize[], int m,
int processSize[], int n)
{
 // Stores block id of the
 // block allocated to a process
 int allocation[n];

 // Initially no block is assigned to any process
 memset(allocation, -1, sizeof(allocation));

 // pick each process and find suitable blocks
 // according to its size ad assign to it
 for (int i = 0; i < n; i++)
 {
  for (int j = 0; j < m; j++)
  {
   if (blockSize[j] >= processSize[i])
   {
    // allocate block j to p[i] process
    allocation[i] = j;

    // Reduce available memory in this block.
```

```
    blockSize[j] -= processSize[i];
    break;

   }
  }
 }

 cout << "\nProcess No.\tProcess Size\tBlock no.\n";
 for (int i = 0; i < n; i++)
 {
  cout << " " << i+1 << "\t\t"
<< processSize[i] << "\t\t";
  if (allocation[i] != -1)
   cout << allocation[i] + 1;
  else
   cout << "Not Allocated";
  cout << endl;
 }
}

// Driver code
int main()
{
 int blockSize[] = {100, 200, 300, 400, 500};
 int processSize[] = {212, 417, 542, 304, 145};
 int m = sizeof(blockSize) / sizeof(blockSize[0]);
 int n = sizeof(processSize) / sizeof(processSize[0]);
 firstFit(blockSize, m, processSize, n);
 return 0 ;
}
```

Output:



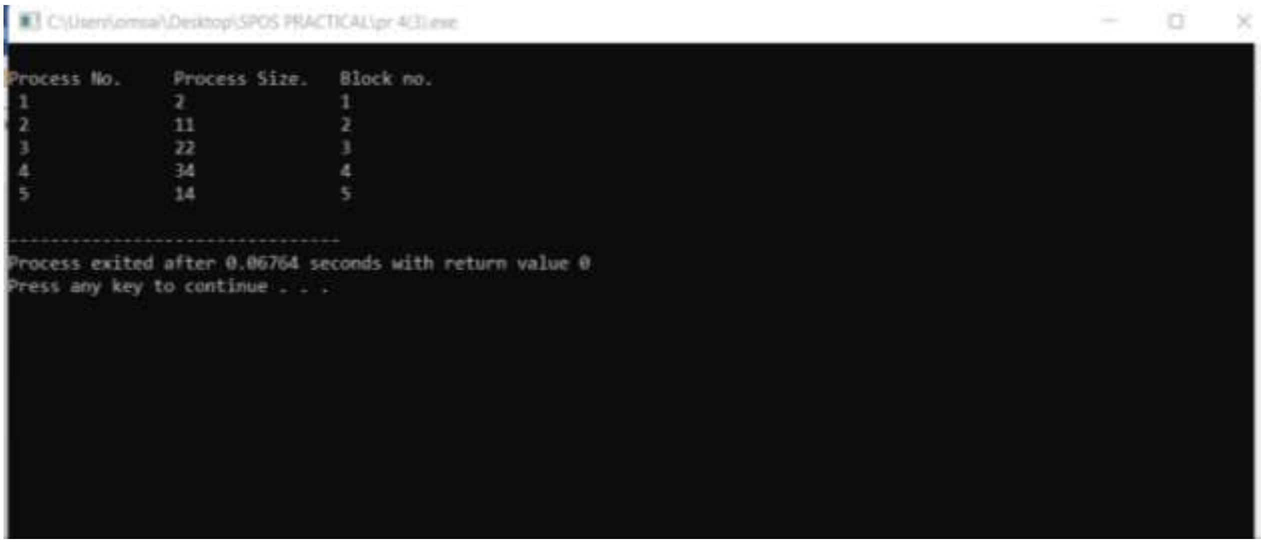
Next fit

```
// C/C++ program for next fit
// memory management algorithm
#include <bits/stdc++.h>
using namespace std;

// Function to allocate memory to blocks as per Next fit
// algorithm
void NextFit(int blockSize[], int m, int processSize[], int n)
{
 // Stores block id of the block allocated to a
 // process
 int allocation[n], j = 0, t = m - 1;

 // Initially no block is assigned to any process
 memset(allocation, -1, sizeof(allocation));

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

  // Do not start from beginning
  while (j < m){
   if(blockSize[j] >= processSize[i]){

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

    // Reduce available memory in this block.
    blockSize[j] -= processSize[i];
// sets a new end point
    t = (j - 1) % m;
    break;
```

```
    }
    if (t == j){
     // sets a new end point
     t = (j - 1) % m;
     // breaks the loop after going through all memory block
     break;
    }
    // mod m will help in traversing the
    // blocks from starting block after
    // we reach the end.
    j = (j + 1) % m;
   }
  }
  cout << "\nProcess No.\tProcess Size.\tBlock no.\n";
  for (int i = 0; i < n; i++)
   {
   cout << " " << i + 1 << "\t\t"
   << processSize[i] << "\t\t";
   if (allocation[i] != -1)
    cout << allocation[i] + 1;
   else
    cout << "Not Allocated";
   cout << endl;
  }
 }
// Driver program
int main()
{
 int blockSize[] = {10, 20, 30, 40, 50};
 int processSize[] = {2, 11, 22, 34, 14};
 int m = sizeof(blockSize) / sizeof(blockSize[0]);
 int n = sizeof(processSize) / sizeof(processSize[0]);
 NextFit(blockSize, m, processSize, n);
```

```
 return 0 ;
}
```

Output:



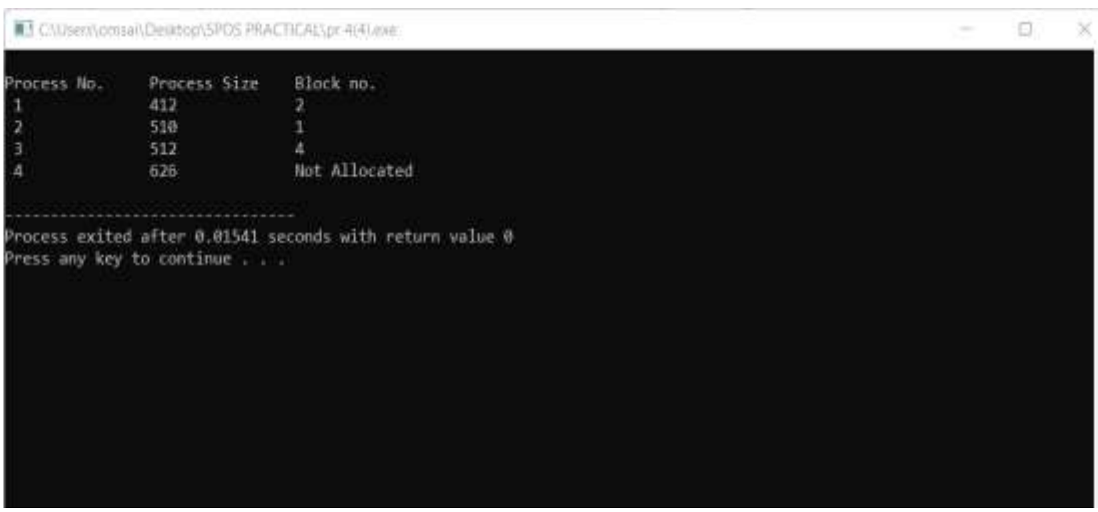
worst - Fit algorithm

```
// C++ implementation of worst - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

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

 // Initially no block is assigned to any process
 memset(allocation, -1, sizeof(allocation));

 // pick each process and find suitable blocks
```

```
// according to its size ad assign to it
for (int i=0; i<n; i++)
{
 // Find the best fit block for current process
 int wstIdx = -1;
 for (int j=0; j<m; j++)
 {
  if (blockSize[j] >= processSize[i])
  {
   if (wstIdx == -1)
    wstIdx = j;
   else if (blockSize[wstIdx] < blockSize[j])
    wstIdx = j;
  }
 }
 // If we could find a block for current process
 if (wstIdx != -1)
{
 // allocate block j to p[i] process
  allocation[i] = wstIdx;
  // Reduce available memory in this block.
  blockSize[wstIdx] -= processSize[i];
 }
}
cout << "\nProcess No.\tProcess Size\tBlock no.\n";
for (int i = 0; i < n; i++)
{
 cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
 if (allocation[i] != -1)
  cout << allocation[i] + 1;
 else
  cout << "Not Allocated";
 cout << endl;
```

```
 }
}
// Driver code
int main()
{
 int blockSize[] = {700, 900, 500, 600, 400};
 int processSize[] = {412, 510, 512, 626};
 int m = sizeof(blockSize)/sizeof(blockSize[0]);
 int n = sizeof(processSize)/sizeof(processSize[0]);
 worstFit(blockSize, m, processSize, n);
 return 0 ;
}
```

Output:



```
C:\Users\omsai\Desktop\SPOS PRACTICAL\pr-4(4).exe

Process No.     Process Size    Block no.
 1              412             2
 2              510             1
 3              512             4
 4              626             Not Allocated

--------------------------------
Process exited after 0.01541 seconds with return value 0
Press any key to continue . . .
```

Conclusion: