Experiment 1

Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Page No. 1
Date

Aim :- To implement Pass I Assembler

Problem statement :- Design suitable data structures and implement pass-I of a two-pass assembler for pseudo-machine in java using object oriented feature. Implementation should consist of few instructions from each category and few assembler directives.

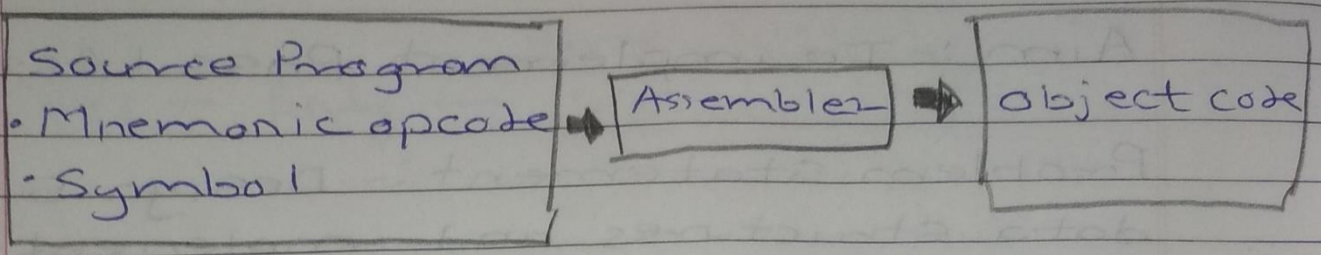
Theory :-

Assembly Language :

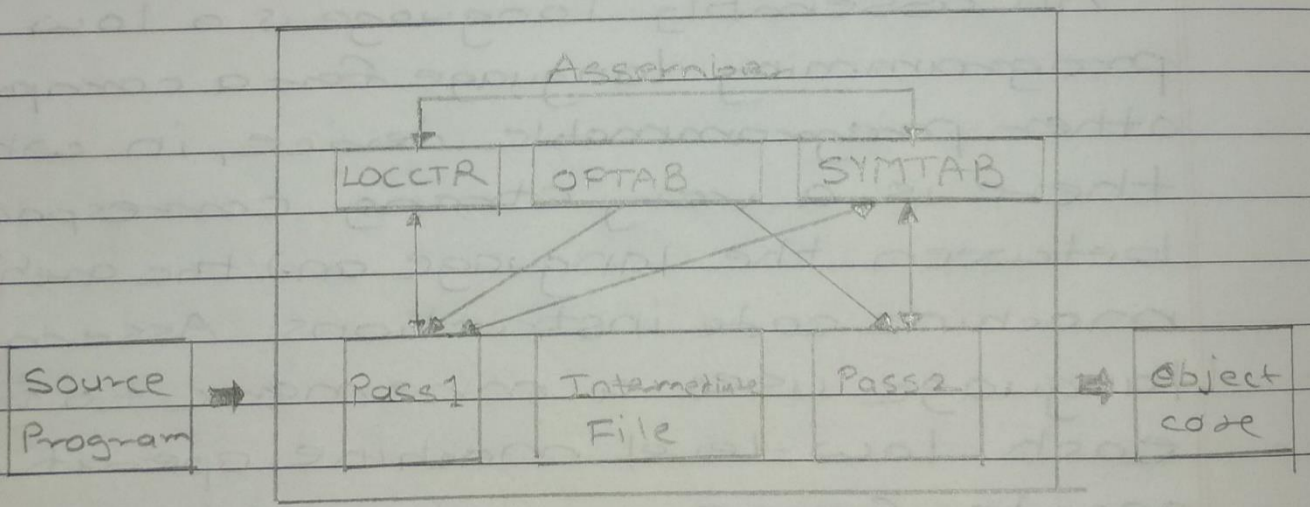
An assembly language is a low level programming language for a computer, or other programmable device, in which there is a very strong correspondence between the language and the architecture machine code instructions. Assembly language uses a mnemonic to represent each low-level machine operation or opcode. Some op-codes require one or more operand as a part of the instruction

Assembler :-

Assembly language is converted into executable machine code by a utility program referred to as an assembler the conversion process is referred to as assembly, or assembly the code.



An assembler, is a translate that translate an assembler program into a conventional machine language program. Basically, the assembler goes through the program one line at a time, and generates machine code for that instructions.



Translate assembly program to object program or machine code is called an Assembler.

One-pass assemblers are used when it is necessary or desirable to avoid a second pass over the source program. An external storage for the intermediate file between two passes is slow or is inconvenient to use

Assembler Directive:-

- Assembler directives are pseudo instructi
  - They will not be translated into machine instructions.
  - They will only provide instruction/ information to the assembler.

- Basic assemble directives:
  - START: Specify name and starting address fe the program.
  - END: Indicate the end of source program.
  - EQU: The EQU ~~ter~~ directive is used to replace number by a Symbol.

- Three Main Data Structures
  - Operation code Table (OPTAB)
  - LOCATION counter (LOCCTR)
  - Symbol Table (SYMTAB)

Instruction Formats

- Addressing modes - Direct addressin
- Register addressing. Register indirect addressing. Immediate addressing
- Implicit addressing. Implicit addressing

- Page relocation - It is desirable

to load and run several program and resources at the same time. The System must be able to load program into memory wherever there is a room. The exact stating address of the program is not known until load time. The assembler can identify those parts of the program that need modification

- Literal

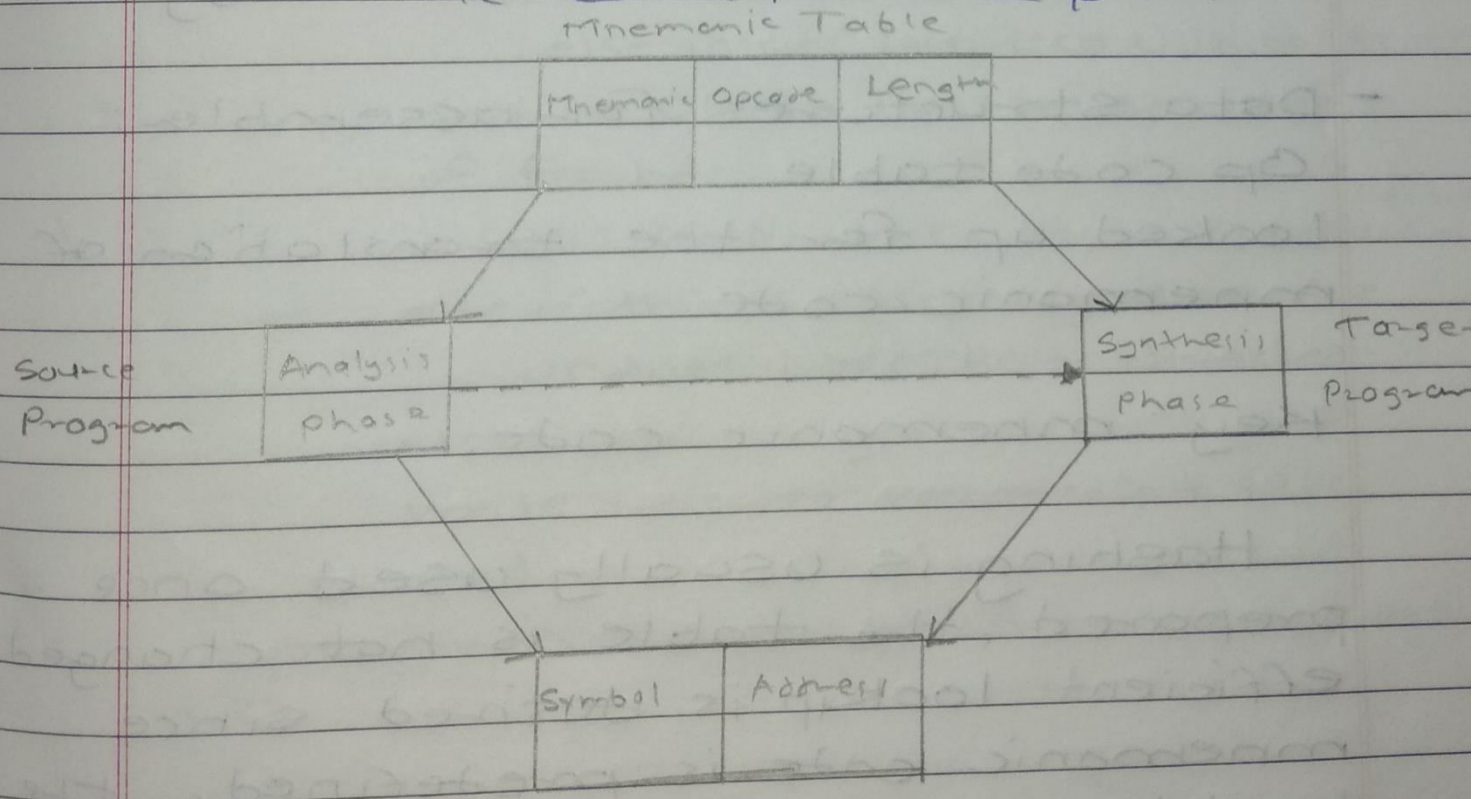
It is convenint fo the programmer to be able to write the value of a constant operand as a part of the instruction that uses it. Such an operand is called a literal. In this assembler language notation, a literal is identified with prefix '=', which is followed by a specification of the literal value.

The difference between literal operands and immediate operands

- For operand we use '=' as prefix, and with immediate operand we can use '#' as prefix
- During immediate addressing, the operand value is assembled as part of the machine instruction,

and there is no money reference.
- With a literal, the assembler generates the specified value as a constant at some other memory location.

- One-pass assemblers
A one pass assembler passes over the source file exactly once, in the same pass collecting the labels, resolving future references and doing the actual assembly. The difficult part is to resolve future lable references and assemble code in one pass.



- Forward reference in one pass assembler
  - Omits the operand address if the symbol has not yet been defined.
  - Enters this undefined symbol into SYMTAB and indicates that it is undefined.
  - Adds the address of this operand address to a list forward reference associated with the SYMTAB entry.
  - When the definition for the symbol is encountered, scan the reference list and insets the address.
  - At the end of the program, report the error if there are still SYMTAB entries indicated undefined symbols.

- Data structures for assembler:
  Op code table
  Looked up for the translation of mnemonic code.

  Key: mnemonic code.

  Hashing is usually used once prepared, the table is not changed efficient lookup is desired since mnemonic code is predefined, the hashing function can be turned a priori. The table may have the instruction format and length

to decide where to put op code bits, operand bits, offset bits

Algorithm for Pass 1 assembler:

```
begin
    if starting address is given
        LOCCTR = starting address;
    else
        LOCCTR = 0;
    while OPCODE != END do ;; or EOF
        begin
        read a line from the code
        if there is a label
            if this label is in SYMTAB, then error
            else insert (label, LOCCTR) into SYMTAB
        search OPTAB for the op code
        if found
            LOCCTR += N ;; N is the length of instruction (4 for MIPS)
        else if this is assembly directive
            update LOCCTR as directed
        else error
        write line to intermediate file
        end
    program size = LOCCTR - starting address;
end
```

Algorithm (Assembler First Pass)

1. loc_cntr := 0 (default value)
   pooltab_ptr := 1; POOLTAB[1] := 1;
   littab_ptr := 1;
2. While next statement is not an END statement
(a) If label is present then
   this_label := Symbol in label field;
   Enter (this_label, loc_cntr) in SYMTAB
(b) If an LTORG statement then,
(i) Process literals LITTAB[POOLTAB]
... LITTAB[littab_ptr-1] to allocate memory and put the address in the address field. Update loc_cntr accordingly
(ii) pooltab_ptr := pooltab_ptr + 1;
(iii) POOLTAB(pooltab_ptr) := littab_ptr;
(c) If a stat a ORIGIN statement then
   loc_cntr := value specified in operand field
(d) If an EQU statement then
(i) this_addr := value of <address spec>;
(ii) Correct the symtab entry for this label to (this_label, this_addr)
3. (Processing of END statement)
(a) Perform step 2(b).
(b) Generate IC '(AD, 02)';
(c) Go to pass II.

Conclusion:- Thus we have implemented Pass-1 Assembler using object oriented features.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 01 [Pass 1 Assembler]

Problem Statement: Design suitable data structures and implement pass-I of a twopass assembler for pseudo-machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives

---

1. Pass 1 Program:

```
Import java.io.BufferedReader;
import java.io.*;
 import java.io.IOException;
 import java.util.*;

public class Pass1 { public static void main(String[] args) {

        BufferedReader br = null; FileReader fr = null;

        FileWriter fw = null; BufferedWriter bw = null;

        try {

                String inputfilename = "/home/sagar-ravan/Desktop/Input.txt"; fr = new
                FileReader(inputfilename); br = new BufferedReader(fr);

                String OUTPUTFILENAME = "/home/sagar-ravan/Desktop/IC.txt";

                fw = new FileWriter(OUTPUTFILENAME); bw = new BufferedWriter(fw);

                Hashtable<String, String> is = new Hashtable<String, String>(); is.put("STOP", "00");
                is.put("ADD", "01"); is.put("SUB", "02"); is.put("MULT", "03"); is.put("MOVER", "04");
                is.put("MOVEM", "05"); is.put("COMP", "06"); is.put("BC", "07"); is.put("DIV", "08");
                is.put("READ", "09"); is.put("PRINT", "10");

                Hashtable<String, String> dl = new Hashtable<String, String>(); dl.put("DC", "01");
                dl.put("DS", "02");

                Hashtable<String, String> ad = new Hashtable<String, String>();

                ad.put("START", "01"); ad.put("END", "02"); ad.put("ORIGIN",
                "03"); ad.put("EQU", "04"); ad.put("LTORG", "05");

                Hashtable<String, String> symtab = new Hashtable<String, String>();

                Hashtable<String, String> littab = new Hashtable<String, String>();

                ArrayList<Integer> pooltab = new ArrayList<Integer>();

                String sCurrentLine; int locptr = 0; int litptr = 1; int
                symptr = 1; int pooltabptr = 1; sCurrentLine =
                br.readLine();

                String s1 = sCurrentLine.split(" ")[1];

                if (s1.equals("START")) { bw.write("AD \t 01 \t");

                        String s2 = sCurrentLine.split(" ")[2]; bw.write("C \t" + s2 + "\n");
```

```
                locptr = Integer.parseInt(s2);

            }

            while ((sCurrentLine = br.readLine()) != null) { int mind_the_LC = 0; String type = null; int flag2 = 0; //
                checks whether addr is assigned to current symbol

                String s = sCurrentLine.split(" |\\,")[0]; // consider the first word in theline

                for (Map.Entry m : symtab.entrySet()) { // allocating addr to arrived symbols

                if (s.equals(m.getKey())) {

                        m.setValue(locptr); flag2 = 1;

                }

                }

                if (s.length() != 0 && flag2 == 0) { // if current string is not " " or addr is not assigned,
// then the current string must be a new symbol.

                        symtab.put(s, String.valueOf(locptr));
                symptr++;

                }

                int isOpcode = 0; // checks whether current word is an opcode or not

                s = sCurrentLine.split(" |\\,")[1]; // consider the second word in the line

                for (Map.Entry m : is.entrySet()) { if (s.equals(m.getKey())) { bw.write("IS\t" + m.getValue() + "\t");
                    // if match found in imperative stmt

                        type = "is"; isOpcode = 1;

                }

                }

                for (Map.Entry m : ad.entrySet()) { if (s.equals(m.getKey())) { bw.write("AD\t" + m.getValue()
                + "\t");
                    // if match found in Assembler Directive type = "ad"; isOpcode = 1;

                }

                }

                for (Map.Entry m : dl.entrySet()) { if (s.equals(m.getKey())) { bw.write("DL\t" + m.getValue()
                + "\t");
                // if match found in declarative stmt type = "dl"; isOpcode = 1;

                }

                }

                if (s.equals("LTORG")) { pooltab.add(pooltabptr);

                for (Map.Entry m : littab.entrySet()) { if (m.getValue() == "") {
                    // if addr is not assigned to theliteral

                        m.setValue(locptr); locptr++; pooltabptr++;
                        mind_the_LC = 1; isOpcode = 1;

                            }
                }

                }
```

```
if (s.equals("END")) { pooltab.add(pooltabptr);

for (Map.Entry m : littab.entrySet()) { if (m.getValue() == "") {

m.setValue(locptr); locptr++; mind_the_LC = 1;

}

}

}

if (s.equals("EQU")) { symtab.put("equ", String.valueOf(locptr));

}

if (sCurrentLine.split(" |\\,").length > 2) { // if there are 3 words s = sCurrentLine.split("
|\\,")[2]; // consider the 3rd word

// this is our first operand.
// it must be either a Register/Declaration/Symbol if (s.equals("AREG")) {
bw.write("1\t"); isOpcode = 1;

} else if (s.equals("BREG")) { bw.write("2\t"); isOpcode =
1;

} else if (s.equals("CREG")) { bw.write("3\t"); isOpcode =
1;

} else if (s.equals("DREG")) { bw.write("4\t"); isOpcode =
1;

} else if (type == "dl") { bw.write("C\t" + s + "\t");

} else { symtab.put(s, ""); // forward referenced symbol }

}

if (sCurrentLine.split(" |\\,").length > 3) { // if there are 4 words s = sCurrentLine.split("

|\\,")[3]; // consider 4th word.

// this is our 2nd operand

// it is either a literal, or a symbol if (s.contains("=")) { littab.put(s, "");

bw.write("L\t" + litptr + "\t");
isOpcode = 1; litptr++;

} else { symtab.put(s, "");
// Doubt : what if the current symbol is already present in SYMTAB?

// Overwrite?
bw.write("S\t" + symptr + "\t"); symptr++;

}
}

bw.write("\n"); // done with a line.

if (mind_the_LC == 0) locptr++;

}

String f1 = "/home/sagar-ravan/Desktop/SYMTAB.txt";

FileWriter fw1 = new FileWriter(f1);
```

```
BufferedWriter bw1 = new BufferedWriter(fw1); for (Map.Entry m : symtab.entrySet())
{ bw1.write(m.getKey() + "\t" + m.getValue() + "\n"); System.out.println(m.getKey() + "
" + m.getValue());

}

String f2 = "/home/sagar-ravan/Desktop/LITTAB.txt";

        FileWriter fw2 = new FileWriter(f2);
BufferedWriter bw2 = new BufferedWriter(fw2); for (Map.Entry m : littab.entrySet()) {
bw2.write(m.getKey() + "\t" + m.getValue() + "\n"); System.out.println(m.getKey() + " "
+ m.getValue());

}

String f3 = "/home/sagar-ravan/Desktop/POOLTAB.txt";

        FileWriter fw3 = new FileWriter(f3);
BufferedWriter bw3 = new BufferedWriter(fw3);

for (Integer item : pooltab) { bw3.write(item + "\n");
        System.out.println(item);

}

bw.close(); bw1.close(); bw2.close();
bw3.close();

} catch (IOException e) {

        e.printStackTrace();

}

}

}
```

PASS 1 - ASSEMBLER OUTPUT:

IC.txt

IC.txt

```
1  IS      04      1       L       1
2  IS      05      1       S       1
3  IS      04      2       L       2
4  IS      04      3       S       3
5  AD      05
6  IS      01      3       L       3
7  IS      00
8  DL      02      C       1
9  DL      02      C       1
10 AD      02
```

SYMTAB.txt

SYMTAB.txt

```
1 A       8
2 LOOP    3
3 B       9
```

LITTAB.txt

LITTAB.txt

```
1 ='4'    4
2 ='6'    10
3 ='1'    5
```

POOLTAB.txt

POOLTAB.txt

```
1 1
2 3
```

Experiment 2

Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Page No. 1
Date

Aim :- To design data structure for Pass-2 Assembler

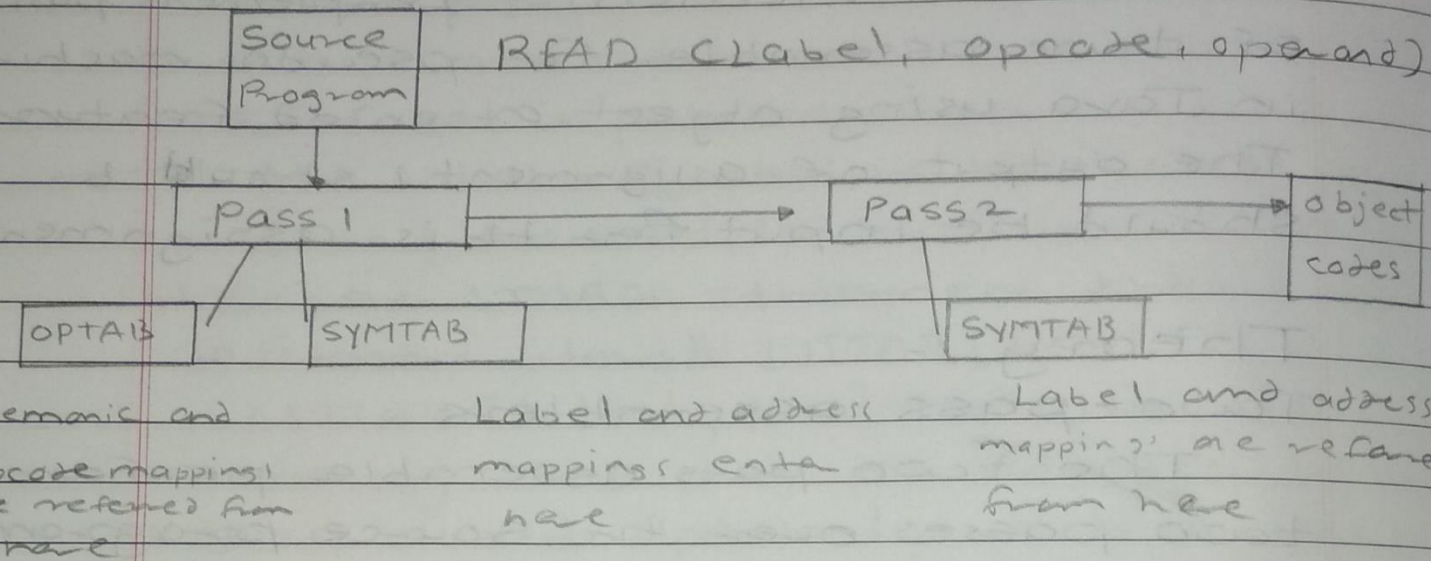
Problem statement :- Implement pass II of two assembler for pseudo-machine in Java using object oriented features. The output of assignment 1 should be input for this assignment

Theory :-

Two-pass assemblers

The two pass assemble performs two passes over the source program. In the first pass, it reads the entire source program, looking only for label definitions. All the labels are collected, assigned addresses, and placed in the Symbol table in this pass, no instructions as assembled and at the end the symbol table should contain all the labels assigned in the program. To assign address to label, the assembles maintains a Location counter (LC). In the second pass the instructions are again read and are assembled using the symbol table. Basically the assembler goes through the program one line at a time, and generates machine code for that instruction. Then assembler proceeds to the next instruction

A Simple Two Pass Assembler Implementation



- Difference between one pass and Two pass assemblers.

• A one pass assemblers passes over the source file exactly once, in the same pass collecting the labels, resolving future references and doing the actual assembly. The difficult part is to resolve future label references and assemble code in one pass. The one pass assembler prepared an intermediate file, which is used as input by the two pass assembler.

• A two pass assembler does two passes over the source file. In the first pass all it does is looks for label definitions and introduces

them in a symbol table. In the second pass, after the symbol table is complete, it does the actual assembly by translating the operations into machine code and so on.

A two-pass assembler perform two sequential scans over the source code.
Pass 1: Symbols and literals are defined
Pass 2: object program is generated

Data structures:-
- Location counter (LC) points to the next location where the code will be placed
- Op-code translation table: contain Symbolic instructions, their lengths and their opcodes.
- Symbol Table (ST): contains labels and their values.
- String storage buffer (SSB); contain ASCII characters for the strings.
- Forward reference table (FRT); contain pointer to the String in SSB and offset where its value will be inserted in the object code.

Algorithm

begin
if starting address is given
LOCCTR = starting address;
else
LOCCTR = 0;
while OPCODE != END do ;; or EOF
begin
read a line from the code
if there is a label
if this label is in SYMTAB, then error
else insert (label, LOCCTR) into SYMTAB
search OPTAB for the op code
if found
LOCCTR += N ;; N is the length of the
instruction (4 for MIPS)
else if this is an assembly directive
update LOCCTR as directed
else error
write line to intermediate file
end
program size = LOCCTR - starting address
end

Conclusion :-

Thus we have generated machine code for the Source Program.

Name:Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 02 [Pass 2 Assembler]

Problem Satement: Implement Pass-II of two pass assembler for pseudo-machine in Java using object oriented features. The output of assignment-1 (intermediate file and symbol table) should be input for this assignment.

---

1. Pass 2 Program:

```
import java.io.BufferedReader;
 import java.io.BufferedWriter;
 import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
 import java.lang.reflect.Array;
import java.util.ArrayList;
 import java.util.Hashtable;
 import java.util.Map;
 public class Pass2 { public static void main(String[] args) {

 try {

                        //1. Read Intermediate code file

                          String f ="/home/Exp2/Desktop/IC_new.txt";

                          FileReader fw =new FileReader(f);

                          BufferedReader IC_file=new BufferedReader(fw);


                          //2.Read Symbol table file

                          String f1="/home/aaExp2/Desktop/SYMTAB.txt";

                          FileReader fs=new FileReader(f1);

                          BufferedReader symtab_file=new BufferedReader(fs);

                           symtab_file.mark(500);


                          //3.Read Literal table file

                          String f2="/home/Exp3/Desktop/LITTAB.txt";

                          FileReader fl=new FileReader(f2);

                          BufferedReader littab_file=new BufferedReader(fl);

                         littab_file.mark(500);



                          //4.create  littab array and hashtable for symbol table


                        String littab[][]=new String[10][2] ;
```

```
Hashtable<String, String> symtab = new Hashtable<String, String>();
String str; int z=0;
//5.Read LITTAB.txt
while ((str = littab_file.readLine()) != null) {
        littab[z][0]=str.split("\\s+")[0]; //first word littab[z][1]=str.split("\\s+")[1];
         //second word z++;
}
//6.Read SYMTAB.txt
while ((str = symtab_file.readLine()) != null) {
symtab.put(str.split("\\s+")[0], str.split("\\s+")[1]); }
//7.Read POOLTAB.txt
String f3 = "/home/Exp2/Desktop/POOLTAB.txt";
FileReader fw3 = new FileReader(f3);
BufferedReader pooltab_file = new BufferedReader(fw3);

ArrayList<Integer> pooltab = new ArrayList<Integer>();
 String t;
while ((t = pooltab_file.readLine()) != null) {
pooltab.add(Integer.parseInt(t));
}
int pooltabptr = 1;
int temp1 = pooltab.get(0);        //dry run
int temp2 = pooltab.get(1);
//7.Read IC.txt String sCurrentLine;
sCurrentLine = IC_file.readLine();
int locptr=0;
//locptr=Integer.parseInt(sCurrentLine.split("\\s+")[3]);
locptr=Integer.parseInt(sCurrentLine.split("\t")[3]);

while ((sCurrentLine = IC_file.readLine()) != null) {

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

String s0 = sCurrentLine.split("\t")[0]; //contains statement type

String s1 = sCurrentLine.split("\t")[1]; //contains statement code

        if (s0.equals("IS")) {
```

```
System.out.print(s1+"\t");

 if (sCurrentLine.split("\t").length == 5) {

System.out.print(sCurrentLine.split("\t")[2]+ "\t");

//7.2 if third character is L

if (sCurrentLine.split("\t")[3].equals("L"))

{
int add = Integer.parseInt(sCurrentLine.split("\t")[4]);

//machine_code

_file.write(littab[add-1][1]);
System.out.print(littab[add-1][1]);
}
//7.3 or if third character is S
if (sCurrentLine.split("\t")[3].equals("S"))
{ int add1 = Integer.parseInt(sCurrentLine.split("\t")[4]);
        //search for the 4th word in symbol
table int i = 1;
String l1;
for (Map.Entry m : symtab.entrySet())
{
if (i == add1) {
System.out.print((String) m.getValue());
}
i++;
                                    }
                            }
} else {
System.out.print("0\t000");
}
}

//DRY RUN is a must
if (s0.equals("AD")) { littab_file.reset();
        if (s1.equals("05")) {                          //if it is
```

```
                                        LTORG int j = 1; while (j < temp1) { littab_file.readLine();
                                                }
                                                    while (temp1 < temp2) {
                                                System.out.print("00\t0\t00" + littab_file.readLine().split("")[1]);
                                                if(temp1<(temp2-1)){ locptr++;
                                                        System.out.println();
                                                        System.out.print(locptr+"\t");
                                                } temp1++;
                                            } temp1 = temp2; pooltabptr++;
                                            if (pooltabptr < pooltab.size()) { temp2 = pooltab.get(pooltabptr); }
                                    } int j = 1;
                                        if (s1.equals("02")) {                          //if it is "END" stmt
                                                    String s;
                                         while ((s = littab_file.readLine()) != null)
                                            {
                                                if (j >= temp1)
                                                        System.out.print("00\t0\t00" + s.split("")[1]);
                                                         j++;
                                                }
                                        }
                                }


                                    if(s0.equals("DL")&&s1.equals("01")){                   //if it is DC stmt
                                     System.out.print("00\t0\t00"+sCurrentLine.split("")[1]); }

                                locptr++;

                                System.out.println();
                        }
                        IC_file.close();
                        symtab_file.close();
                         littab_file.close();
                         pooltab_file.close();

                } catch (IOException e) {
                        e.printStackTrace();
                }

        }
}
```

PASS 2 - ASSEMBLER OUTPUT:

PASS- 2 OUTPUT:

Problems Javadoc Declaration Console Navigator (Deprecated)

```
<terminated> Pass2 [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (31-May-2021, 9:38:20 pm)
200     04      1       4
201     05      1       8
202     04      2       10
203     04      3       9
204     00      0       004
205     00      0       006
206     01      3       5
207     00      0       000
208
209
210     00      0       001
```

IC_New.txt

IC_new.txt

```
1  AD       01      C       200
2  IS       04      1       L       1
3  IS       05      1       S       1
4  IS       04      2       L       2
5  IS       04      3       S       3
6  AD       05
7  IS       01      3       L       3
8  IS       00
9  DL       02      C       1
10 DL       02      C       1
11 AD       02
```

Input.txt

Input.txt

```
1  START 200
2   MOVER AREG,='4'
3   MOVEM AREG,A
4   MOVER BREG,='1'
5  LOOP MOVER CREG,B
6   LTORG
7   ADD CREG,='6'
8   STOP
9  A DS 1
10 B DS 1
11  END
```

LITTAB.txt

LITTAB.txt

```
1  ='4'    4
2  ='6'    10
3  ='1'    5
```

POOLTAB.txt

POOLTAB.txt

```
1
3
```

SYMTAB.txt

SYMTAB.txt

```
A       8
LOOP    3
B       9
```

Assignment 3

Name - Bhushan Nikumbhe
Roll No 22
Div B
Page No. 1
Date

Aim :- To design Data structure for Macroprocessor.

Problem Statement :- Design suitable data structures and implement pass-I of a two-pass-macro-processor using OOP features in java.

Theory :-

1: Macro processor

A macro processor is a program that reads a files and scans them for certain keywords. When a keywords is found, it is replaced by some text. The keyword/text combination is called a Macro.

2. Basic tasks performed by Macro-processor

a) Recognize macro definition
b) Save the definition
c) Recognize call
d) Expanded calls and substitute argument.

3. Macro call & Expansion

The operation define by macro can be used by writing macro name in the mnemonic field. And its operand field. Appearance of macro name in the mnemonic fields leads to macro call.

4. Macro definition part :-
It consists of
1. Macro prototype statement :- This declares the name of macro and the types of parameters.
2. Preprocessor Statement :- It is used to perform auxiliary function during macro expansion.

5. Implementation Logic
I. Definition Processing :- Scan all macro definitions and for each macro defition enter the macro name in macro name table (MNT). Store entire macro definition in macro definition table (MDT) and add auxiliary information in MNT such as no positional parameters (#PP), macro definition table position (MDTP) etc.
II. Macro Expansion :- Examine all statement in assembly source program to detect the macro calls. For each macro call locate the macro in MNT, retrieve MDTP and actual parameters and expand the macro.

6. Data Structure required for macro definition processing
I. Macro Name Table (MNT) :- Fields Name of macro, MDTP (Macro definition Table Pointer), KPDTPC (Keywords

Parameters Default Table Position)
II. Parameter Name Table (PNTAB) :-
Fields parameter name
III. Keywords Parameter Default Table (KPDTAB) :- Field parameter name, default value.

Algorithm :

```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE ≠ 'END' do
        begin
            GETLINE
            PROCESSLINE
            end {while}
        end {macro processor}
procedure PROCESSLINE
    begin
        search NAMTAB for OPCODE
        if found then
            EXPAND
        else if OPCODE = 'MACRO' then
            DEFINE
        else write source line to expanded file
        end {PROCESSLINE}
```

Algorithm :-

```
procedure EXPAND
    begin
      EXPANDING := TRUE
      get first line of macro definition from
      DEFTAB set up arrangements from macro
      invocation in ARGTAB write macro
      while not end of macro definition do
          begin
            GETLINE
            PROCESSLINE
            end {while}
          Expanding := FALSE
        end {EXPAND}

procedure GETLINE
    begin
      if EXPANDING then
          begin get next line of macro
          definition from DEFTAB
          end {if}
      else
          read next line from input file
    end {GETLINE}
```

Conclusion :- Thus pass I of Macro processor is implemented and .MINT, MDT & ALA file is generated.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 03 [PASS-1 Macroprocessor]

Problem Satement: Design suitable data structures and implement pass-I of a two-pass macro-processor using OOP features in Java.

1. Pass 1 Macro Code:

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

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

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

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

```
                                        while(!(s.charAt(i)==' '||s.charAt(i)==',')){
                                                temp+=s.charAt(i++);
                                                if(i==s.length())
                                                        break;

                                        }
                                        i--;
                                        f3.write("#"+pntab.get(temp));
                                }
                                else
                                        f3.write(s.charAt(i));
                        }
                        f3.write("\n");
                        mdtp++;
                }
                else{
                        f1.write(s+'\n');
                }
        }
        b1.close();
        f1.close();
        f2.close();
        f3.close();
        f4.close();
    }
}

/*
OUTPUT:

bhushan@bhushan:~/SPOSL$ javac macroPass1.java
bhushan@bhushan:~/SPOSL$  java macroPass1
bhushan@bhushan:~/SPOSL$  cat intermediate.txt
M1 10,20,&b=CREG
M2 100,200,&u=AREG,&v=BREG

bhushan@bhushan:~/SPOSL$  cat mnt.txt
M1      2       2       1       1
M2      2       2       7       3
M3      2       0       13      4

bhushan@bhushan:~/SPOSL$  cat mdt.txt
MOVE #3,#1
ADD #3,='1'
MOVER #3,#2
M2 69,169
ADD #3,='5'
MEND
MOVER #3,#1
MOVER #4,#2
M3 73,173
ADD #3,='15'
ADD #4,='10'
MEND
ADD #1,#2
MEND

bhushan@bhushan:~/SPOSL$  cat kpdt.txt
a       AREG
b       -
u       CREG
v       DREG

*/
```

Assignment 4

Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Page No. 1
Date

Aim :- Design of MACRO PASS-2

Problem Statement :- Write a java program for pass-II of two-pass macro-processor. The output of assignment 3 should be input for this assignment.

Theory :-

1. MACRO Processor

A macro processor is a program that reads a files and scans them for certain keywords, when a keyword is found, it is replaced by some text. The keyword / text combination is called a macro

2. Basic tasks performed by Macro processor

a) recognize macro definition
b) Save the definition
c) Recognize call
d) Expanded calls and substitute arguments

In two-pass macro-preprocessor, you have two algorithms to implement, first pass and second pass. Both the algorithms examine the line by line over the input data available. Two algorithms to implement two-pass-macro-processor etc.

Pass I Macro Definition

1. Initialize MDTC and MNTC with value one, so that previous value of MDTC and MNTC is set to value one.
2. Read the first input data.
3. If this data contain MACRO pseudo opcode then
   A. Read the next data input.
   B. Enter the name of the macro and current value of MDTC in MNT.
   C. Increase the counter value of MNT by value one.
   D. Prepare that argument list array respective to the macro found.
   E. Enter the macro definition into MDT
   Increase the counter of MDT by value one
   F. Read next line of the input data.
   G. Substitute true index notations for dummy arguments passed in MACRO.
   H. Increase the counter of the MDT by value one.
   I. If mend pseudo opcode is encountered then next source of input data is read.
   J. Else expands data input.
4. If macro pseudo opcode is not encountered in data input then
   A. A copy of input data is created
   B. If end pseudo opcode is found then go to Pass 2
   C. otherwise read next source of input dat

Pass 2 Macro calls and Expansion.

1. Read the input data received from Pass I.
2. Examine each operation code for finding respective entry in MNT.
3. If the name of the macro is encountered then
   A. A pointer is set to the MNT entry where name of the macro is found. This pointer is called Macro definition Table Pointer (MDTP).
   B. Prepare argument list array containing a table of dummy arguments.
   C. Increase the value of MDTP by value one.
   D. Read next line from MDT.
   E. Substitute the value from the arguments list of the macro for Dummy arguments.
   F. If mend pseudo Opcode is found then next source of input data is read.
   G. Else expands data input.
4. When macro name is not found then create expanded data file.
5. If end pseudo code is encountered then feed the expanded source file to Assembler for processing.
6. Else read next source of datainput.

Conclusion :- Thus Pass II of Macro processor is implemented and ALA file is generated.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 04 [PASS-2 Macroprocessor]

Problem Satement: Write a Java program for pass-II of a two-pass macro-processor. The output of assignment-3 (MNT, MDT and file without any macro definitions) should be input for this assignment.

---

1. Pass 2 Macro Code:

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

public class macroPass2 {
	public static void main(String[] Args) throws IOException{
		BufferedReader b1 = new BufferedReader(new FileReader("intermediate.txt"));
		BufferedReader b2 = new BufferedReader(new FileReader("mnt.txt"));
		BufferedReader b3 = new BufferedReader(new FileReader("mdt.txt"));
		BufferedReader b4 = new BufferedReader(new FileReader("kpdt.txt"));
		FileWriter f1 = new FileWriter("Pass2.txt");
		HashMap<Integer,String> aptab=new HashMap<Integer,String>();
		HashMap<String,Integer> aptabInverse=new HashMap<String,Integer>();
		HashMap<String,Integer> mdtpHash=new HashMap<String,Integer>();
		HashMap<String,Integer> kpdtpHash=new HashMap<String,Integer>();
		HashMap<String,Integer> kpHash=new HashMap<String,Integer>();
		HashMap<String,Integer> macroNameHash=new HashMap<String,Integer>();
		Vector<String>mdt=new Vector<String>();
		Vector<String>kpdt=new Vector<String>();
		String s,s1;
		int i,pp,kp,kpdtp,mdtp,paramNo;
		while((s=b3.readLine())!=null)
			mdt.addElement(s);
		while((s=b4.readLine())!=null)
			kpdt.addElement(s);
		while((s=b2.readLine())!=null){
			String word[]=s.split("\t");
			s1=word[0]+word[1];
			macroNameHash.put(word[0],1);
			kpHash.put(s1,Integer.parseInt(word[2]));
			mdtpHash.put(s1,Integer.parseInt(word[3]));
			kpdtpHash.put(s1,Integer.parseInt(word[4]));
		}
		while((s=b1.readLine())!=null){
			String b1Split[]=s.split("\\s");
			if(macroNameHash.containsKey(b1Split[0])){
				pp= b1Split[1].split(",").length-b1Split[1].split("=").length+1;
				kp=kpHash.get(b1Split[0]+Integer.toString(pp));
				mdtp=mdtpHash.get(b1Split[0]+Integer.toString(pp));
				kpdtp=kpdtpHash.get(b1Split[0]+Integer.toString(pp));
				String actualParams[]=b1Split[1].split(",");
				paramNo=1;
				for(int j=0;j<pp;j++){
					aptab.put(paramNo, actualParams[paramNo-1]);
					aptabInverse.put(actualParams[paramNo-1],paramNo);
					paramNo++;
				}
				i=kpdtp-1;
				for(int j=0;j<kp;j++){
					String temp[]=kpdt.get(i).split("\t");
					aptab.put(paramNo,temp[1]);
					aptabInverse.put(temp[0],paramNo);
					i++;
					paramNo++;
				}
				i=pp+1;
				while(i<=actualParams.length){
					String initializedParams[]=actualParams[i-1].split("=");
```

```
		aptab.put(aptabInverse.get(initializedParams[0].substring(1,initializedParams[0].length())),initializedParams[1].substring(0,in
itializedParams[1].length()));
					i++;
				}
				i=mdtp-1;
				while(mdt.get(i).compareToIgnoreCase("MEND")!=0){
					f1.write("+ ");
					for(int j=0;j<mdt.get(i).length();j++){
						if(mdt.get(i).charAt(j)=='#')
							f1.write(aptab.get(Integer.parseInt("" + mdt.get(i).charAt(++j))));
						else
							f1.write(mdt.get(i).charAt(j));
					}
					f1.write("\n");
					i++;
				}
				aptab.clear();
				aptabInverse.clear();
			}
			else
				f1.write("+ "+s+"\n");
		}
		b1.close();
		b2.close();
		b3.close();
		b4.close();
		f1.close();
	}
}


/*
OUTPUT:

OUTPUT:
bhushan@bhushan:~/SPOSL$ javac macroPass2.java
bhushan@bhushan:~/SPOSL$ java macroPass2
bhushan@bhushan :~/SPOSL$ cat Pass2.txt

Intermediate - -
M1 10,20,&b=CREG
M2 100,200,&u=&AREG,&v=&BREG



Kpdt--
a	AREG
b	-
u	CREG
v	DREG

pass2 --
+ MOVE AREG,10
+ ADD AREG,='1'
+ MOVER AREG,20
+ ADD AREG,='5'
+ MOVER &AREG,100
+ MOVER &BREG,200
+ ADD &AREG,='15'
+ ADD &BREG,='10'


MNT --
M1	2	2	1	1
M2	2	2	6	3


MDT --
MOVE #3,#1
```

```
ADD #3,='1'
MOVER #3,#2
ADD #3,='5'
MEND
MOVER #3,#1
MOVER #4,#2
ADD #3,='15'
ADD #4,='10'
MEND
```

Assignment 5

Name - Bhushan Nikumbhe
Roll No. 22
Div. B

Page No. 1
Date

Aim :- Design Lex program for to generate token of given input file

Problem Statement :- Write a program using Lex specifications to implement lexical analysis Phase of compiler to generates tokens of subset of java programs

Pre-requisites :- LEX 110, LEX 120, LEX 130, LEX 140, LEX 160, 250

Software Requirement :-
O/S : Ubuntu
S/w name : Lex Tool (flex)

Objective :-
1. To understand Lex concept,
2. To implement Lex program
3. To study about Lex & Java.
4. To know important about Lexical analyzer.

Theory :-
Lex stands for Lexical Analyzer. Lex is a tool for generating Scanners. Scanners are programs that recognize lexical patterns in text. These lexical patterns are defined in a particular syntax A matched regular expression

may have an associated action. This action may also include returning a token. When lex receives input in the form of a file a text, it takes input one character at a time and continues until a pattern is matched then lex performs the associated action. If on the other hand, no regular expression can be matched, further processing stops and Lex displays an error message. Lex and C are tightly coupled. A .lex file is passed through the lex utility, and produces outputs in files in C. These files are coupled to produce an executable version of the lexical analyzer.

Regular Expression in Lex:-

A regular expression is a pattern description using a Meta language. An expression is made up of symbols. Normal Symbols are characters and numbers, but there are other symbols that have special meaning in Lex. The following two tables define some of the symbols used in LEX and give a few typical examples.

Programming in Lex :-
Programming in Lex can be divided into three steps
1. Specify the pattern associated actions in form that lex can understand.
2. Run Lex over this file to generate C code for the scanner.
3. compile and link the C code to produce the executable scanner.

Regular expression are used for pattern matching. A character class define a single character and normal operators lose their meaning. In case both matches are the same length, then the first pattern listed is used.

```
.... definitions ....
%%
.... rules ....
%%
.... subroutines ....
```

Input to Lex is divided into three sections with %% dividing the sections. However if we don't

Specify any rules then the default action is to match everything and copy it to output. Defaults for input and output are stdin and stdout, respectively.

Conclusion :-

Thus, we have studied lexical analyzer and implemented an application for lexical analyzer to perform scan the program and generates token of subset of Java.

Name:Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 05 [LEX Program]

Problem Satement: Write a program using Lex specifications to implement lexical analysis Phase of compiler to generate tokens of subset of Java program.

---

1. Code b2.l:

```
%{
   FILE* yyin; %}
DATATYPE "int"|"char"|"float"|"double"
KEYWORDS "class"|"static"
DIGIT [0-9]
NUMBER {DIGIT}+
TEXT [a-zA-Z]
IDENTIFIER {TEXT}({DIGIT}|{TEXT}|"_")*
ACCESS "public"|"private"|"protected"
CONDITIONAL "if"|"else"|"else if"|"switch"
LOOP "for"|"while"|"do"
FUNCTION {ACCESS}{DATATYPE}{IDENTIFIER}"("({DATATYPE}{IDENTIFIER})*")"
%%
[ \n\t]+ ;
{DATATYPE} {printf("%s == DATATYPE\n",yytext);}
{KEYWORDS} {printf("%s == KEYWORDS\n",yytext);}
{NUMBER} {printf("%s == NUMBER\n",yytext);}
{IDENTIFIER} {printf("%s == IDENTIFIER\n",yytext);}
{CONDITIONAL} {printf("%s == CONDITIONAL\n",yytext);}
{FUNCTION} {printf("%s == FUNCTION\n",yytext);}
. ;
%%
int yywrap(){
}

int main(int argc,char* argv[]){    yyin= fopen(argv[1],"r");
   yylex();    fclose(yyin);
   return 0;
```

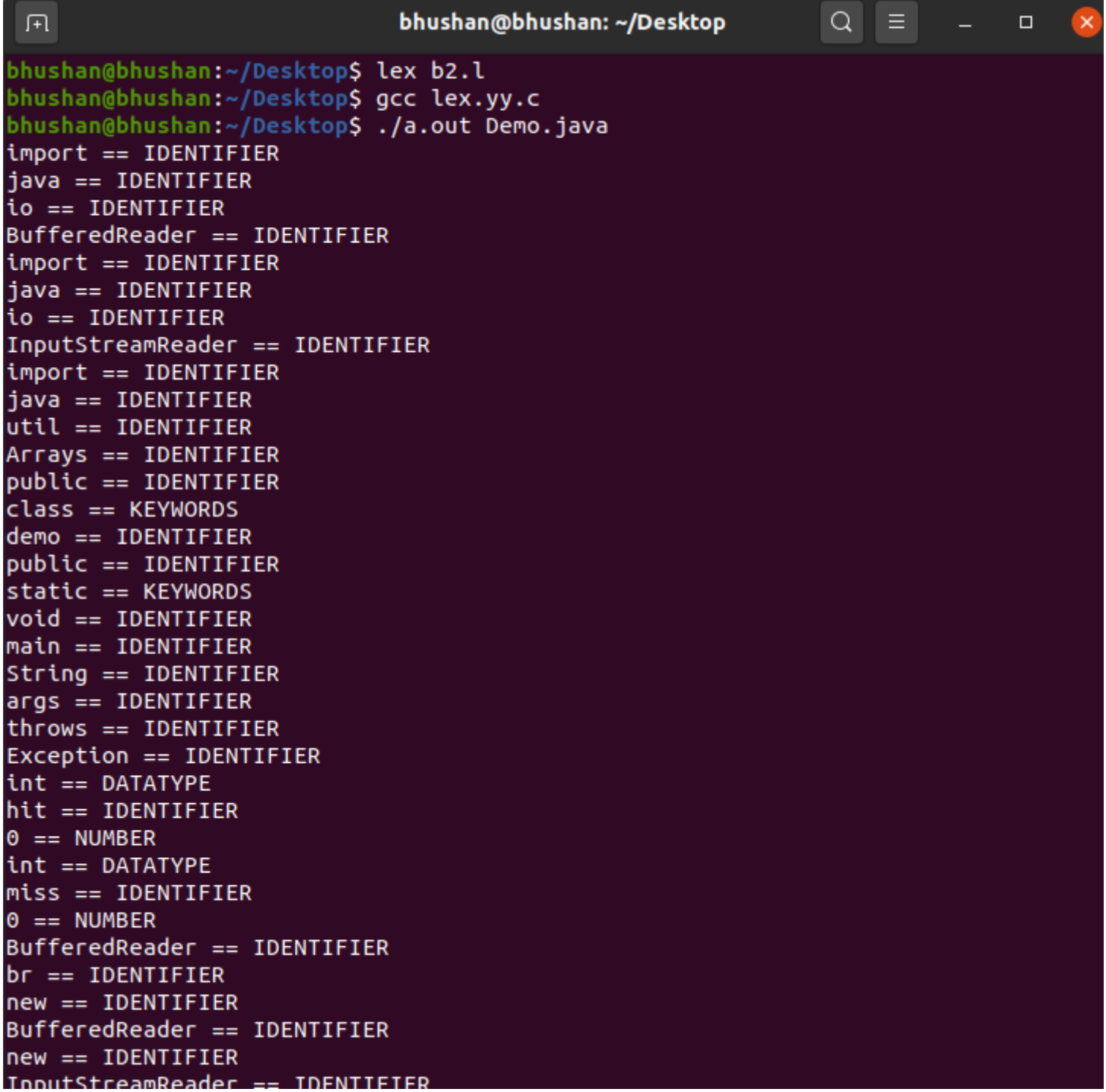
}

2. Demo.java Code:

```
import java.io.BufferedReader; import
java.io.InputStreamReader; import java.util.Arrays;

public class demo

{


        public static void main(String[] args)throws Exception { int hit=0; int miss=0;

            BufferedReader br=new BufferedReader(new

InputStreamReader(System.in));

            System.out.println("Enter total no of frames"); int
            noFrames=Integer.parseInt(br.readLine());

            int[] frames=new int[noFrames]; int[] lruTime=new
            int[noFrames];

            System.out.println("Enter total no of pages"); int totalPages =
            Integer.parseInt(br.readLine());

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

                System.out.println("Enter page value"); int page=

                Integer.parseInt(br.readLine()); int searchIndex=isPresent(frames,

                page );

                    if(searchIndex!=-1){
//                  page fonud

                        hit++; lruTime[searchIndex]=i; System.out.println("Page
                        Hit");

                } else{

                    System.out.println("Page Miss"); miss++;

//                  page not found

                    int emptyindex=isEmpty(frames); if(emptyindex!=-1){

//                      if frame is empty

                                frames[emptyindex]=page;

                                lruTime[emptyindex]=i;
                    } else{

//user lru algo to find replace location

                                int minLocationIndex=lru(lruTime);

                                System.out.println("Replace "+
frames[minLocationIndex]);

                        frames[minLocationIndex]=page; lruTime[minLocationIndex]=i;

                    }
```

```
                }

        }


                System.out.println("Total page hit" + hit);
                System.out.println("Total Page miss " + miss);
                System.out.println(Arrays.toString(frames));

    }

    public static int lru(int[] lruTime){ int min = 9999; int index = -1; for(int
                i=0;i<lruTime.length;i++){

                    if(min>lruTime[i]){ min=lruTime[i]; index=i;

                        }

                }

                    return index;
    }

    public static int isEmpty(int[] frames){

                    for(int i=0;i<frames.length;i++)
            { if(frames[i]==0){ return i;

                }

            }

            return -1;

    }

    public static int isPresent(int[] frames, int search){

        for(int i=0;i<frames.length;i++){ if(frames[i]==search)

                    return i;
        }

        return -1; }


}
```

OUTPUT:



OUTPUT:

bhushan@bhushan:~/SPOSL/LexProgram$ lex b2.l

bhushan@bhushan:~/SPOSL/LexProgram$ gcc lex.yy.c

bhushan@bhushan :~/SPOSL/LexProgram$ ./a.out demo.java

import == IDENTIFIER
java == IDENTIFIER
io == IDENTIFIER
BufferedReader == IDENTIFIER
import == IDENTIFIER
java == IDENTIFIER
io == IDENTIFIER
InputStreamReader == IDENTIFIER
import == IDENTIFIER
java == IDENTIFIER
util == IDENTIFIER

Arrays == IDENTIFIER
 public == IDENTIFIER
class == KEYWORDS
demo == IDENTIFIER
public == IDENTIFIER
 static == KEYWORDS
 void == IDENTIFIER
 main == IDENTIFIER
String == IDENTIFIER
 args == IDENTIFIER
throws == IDENTIFIER
 Exception == IDENTIFIER
 int == DATATYPE
hit == IDENTIFIER
0 == NUMBER
 int == DATATYPE
 miss == IDENTIFIER
 0 == NUMBER

BufferedReader == IDENTIFIER
 br == IDENTIFIER
 new == IDENTIFIER
BufferedReader == IDENTIFIER
new == IDENTIFIER
InputStreamReader == IDENTIFIER
 System == IDENTIFIER
 in == IDENTIFIER
 System == IDENTIFIER
out == IDENTIFIER
 println == IDENTIFIER
Enter == IDENTIFIER
total == IDENTIFIER
no == IDENTIFIER
of == IDENTIFIER
 frames == IDENTIFIER
 int == DATATYPE
noFrames == IDENTIFIER
 Integer == IDENTIFIER
 parseInt == IDENTIFIER
 br == IDENTIFIER
 readLine == IDENTIFIER
 int == DATATYPE
 frames == IDENTIFIER
new == IDENTIFIER
int == DATATYPE
 noFrames == IDENTIFIER
int == DATATYPE
lruTime == IDENTIFIER
 new == IDENTIFIER
int == DATATYPE
 noFrames == IDENTIFIER
 System == IDENTIFIER
out == IDENTIFIER
println == IDENTIFIER
Enter == IDENTIFIER
total == IDENTIFIER
 of == IDENTIFIER
pages == IDENTIFIER

int == DATATYPE
 totalPages == IDENTIFIER
Integer == IDENTIFIER
parseInt == IDENTIFIER
 br == IDENTIFIER
readLine == IDENTIFIER
 for == IDENTIFIER
int == DATATYPE
 i == IDENTIFIER
 0 == NUMBER
 i == IDENTIFIER
totalPages == IDENTIFIER
 i == IDENTIFIER
System == IDENTIFIER
out == IDENTIFIER
println == IDENTIFIER
Enter == IDENTIFIER
 page == IDENTIFIER
value == IDENTIFIER
 int == DATATYPE
 page == IDENTIFIER
 Integer == IDENTIFIER
 parseInt == IDENTIFIER
br == IDENTIFIER
readLine == IDENTIFIER
int == DATATYPE
searchIndex == IDENTIFIER
isPresent == IDENTIFIER
 frames == IDENTIFIER
page == IDENTIFIER
if == IDENTIFIER
searchIndex == IDENTIFIER
1 == NUMBER
 page == IDENTIFIER
fonud == IDENTIFIER
hit == IDENTIFIER
 lruTime == IDENTIFIER
 searchIndex == IDENTIFIER
 i == IDENTIFIER
System == IDENTIFIER
 out == IDENTIFIER
println == IDENTIFIER
Page == IDENTIFIER
Hit == IDENTIFIER
 else == IDENTIFIER
 System == IDENTIFIER
 out == IDENTIFIER
println == IDENTIFIER
Page == IDENTIFIER
 Miss == IDENTIFIER
 miss == IDENTIFIER
 page == IDENTIFIER
 not == IDENTIFIER
found == IDENTIFIER
 int == DATATYPE
 emptyindex == IDENTIFIER
isEmpty == IDENTIFIER
frames == IDENTIFIER

if == IDENTIFIER
emptyindex == IDENTIFIER
1 == NUMBER
if == IDENTIFIER
frame == IDENTIFIER
 is == IDENTIFIER
empty == IDENTIFIER
frames == IDENTIFIER
 emptyindex == IDENTIFIER
page == IDENTIFIER
 lruTime == IDENTIFIER
emptyindex == IDENTIFIER
i == IDENTIFIER
 else == IDENTIFIER
 user == IDENTIFIER
lru == IDENTIFIER
algo == IDENTIFIER
 to == IDENTIFIER
find == IDENTIFIER
 replace == IDENTIFIER
 location == IDENTIFIER
int == DATATYPE
minLocationIndex == IDENTIFIER
 lru == IDENTIFIER
 lruTime == IDENTIFIER
System == IDENTIFIER
out == IDENTIFIER
println == IDENTIFIER
 Replace == IDENTIFIER
 frames == IDENTIFIER
 minLocationIndex == IDENTIFIER
 frames == IDENTIFIER
minLocationIndex == IDENTIFIER
page == IDENTIFIER
 lruTime == IDENTIFIER
minLocationIndex == IDENTIFIER
 i == IDENTIFIER
 System == IDENTIFIER
 out == IDENTIFIER
 println == IDENTIFIER
Total == IDENTIFIER
 page == IDENTIFIERhit == IDENTIFIER
 hit == IDENTIFIER
System == IDENTIFIER
out == IDENTIFIER
println == IDENTIFIER
 Total == IDENTIFIER
Page == IDENTIFIER
miss == IDENTIFIER
miss == IDENTIFIER
System == IDENTIFIER
out == IDENTIFIER
 println == IDENTIFIER
Arrays == IDENTIFIER
frames == IDENTIFIER
public == IDENTIFIER
 static == KEYWORDS
 int == DATATYPE

lru == IDENTIFIER
int == DATATYPE
lruTime == IDENTIFIER
int == DATATYPE
min == IDENTIFIER
9999 == NUMBER
int == DATATYPE
index == IDENTIFIER
1 == NUMBER
for == IDENTIFIER
int == DATATYPE
i == IDENTIFIER
0 == NUMBER
i == IDENTIFIER
lruTime == IDENTIFIER
length == IDENTIFIER
i == IDENTIFIER
if == IDENTIFIER
min == IDENTIFIER
lruTime == IDENTIFIER
i == IDENTIFIER
min == IDENTIFIER
lruTime == IDENTIFIER
i == IDENTIFIER
index == IDENTIFIER
i == IDENTIFIER
return == IDENTIFIER
index == IDENTIFIER
public == IDENTIFIER
static == KEYWORDS
int == DATATYPE
isEmpty == IDENTIFIER
int == DATATYPE
frames == IDENTIFIER
for == IDENTIFIER
int == DATATYPE
i == IDENTIFIER
0 == NUMBER
i == IDENTIFIER
frames == IDENTIFIER
length == IDENTIFIER
i == IDENTIFIER
if == IDENTIFIER
frames == IDENTIFIER
i == IDENTIFIER 0 == NUMBER
return == IDENTIFIER
i == IDENTIFIER
return == IDENTIFIER
1 == NUMBER
public == IDENTIFIER
static == KEYWORDS
int == DATATYPE
isPresent == IDENTIFIER
int == DATATYPE
frames == IDENTIFIER
int == DATATYPE
search == IDENTIFIER
for == IDENTIFIER

```
int == DATATYPE
 i == IDENTIFIER
 0 == NUMBER
 i == IDENTIFIER
 frames == IDENTIFIER
 length == IDENTIFIER
 i == IDENTIFIER
if == IDENTIFIER
frames == IDENTIFIER
 i == IDENTIFIER
search == IDENTIFIER
return == IDENTIFIER
 i == IDENTIFIER
return == IDENTIFIER
 1 == NUMBER

bhushan@bhushan:~/SPOSL/LexProgram$
```

Assignment 6

Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Page No. 1
Date

Aim :- Design Lex program to count no. of words, lines and characters of given input file.

Problem Statement :- Write program using Lex Specifications to implement lexical analysis Phase of compiler to count no. of words, lines and characters of given input file.

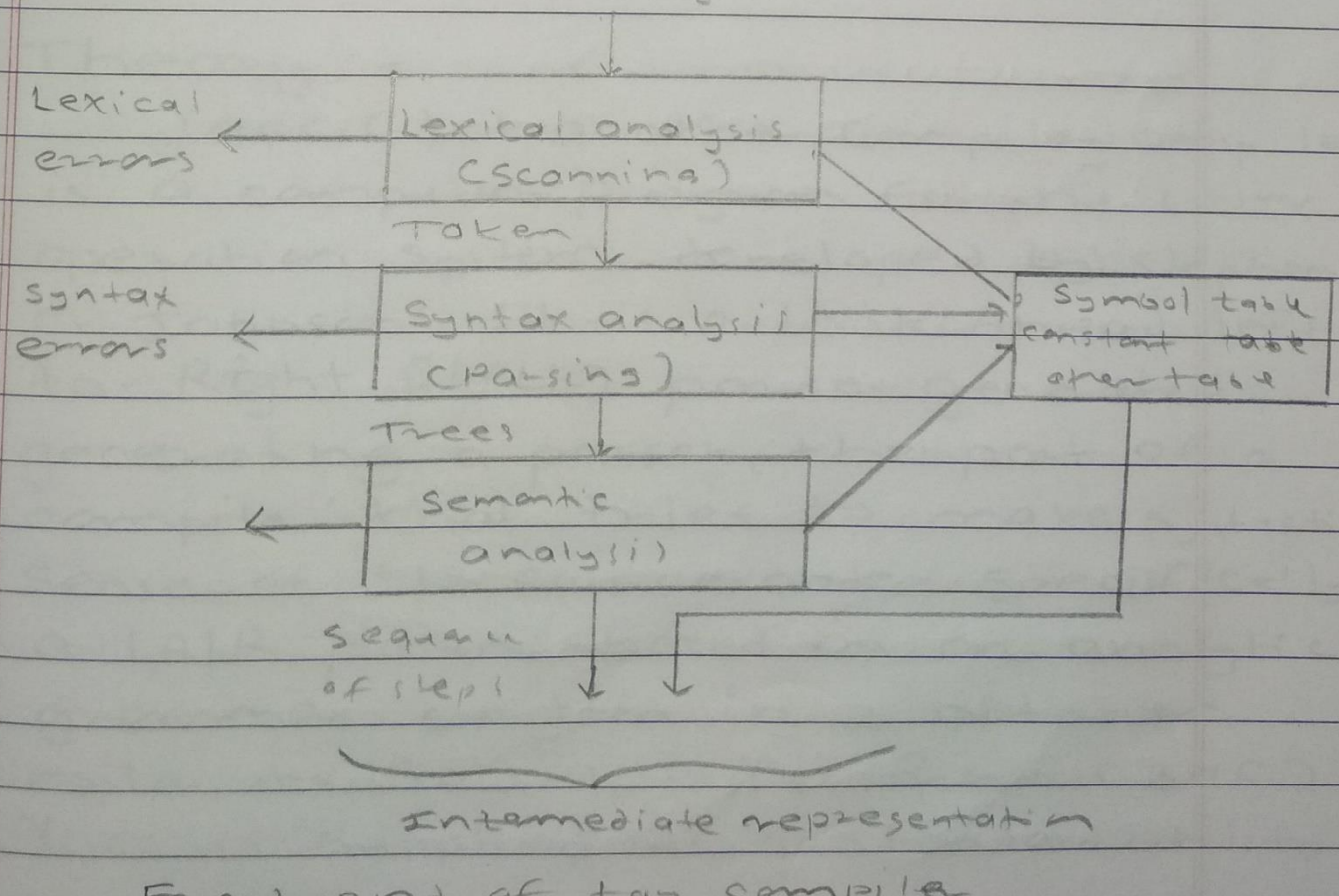
Pre-requisites :- Lex Basics

Software Requirements :-
O/S : Ubuntu Kylin
S/W name : Lex Tool (flex)

Objectives :-
1. To understand Lex concepts
2. To implement Lex Program for no's of count.
3. To Study about Lex & Java.
4. To know important about lexical analyzer.

HOW THE INPUT IS MATCHED -
when the generated scanner is run, it analyzes its input looking for strings, which match any of its patterns. If it finds more than

one match, it takes the one matching the most text. If it finds two or more matches of the same length, the rule listed first in the flex input file is chosen. Once the match is determined, the text corresponding to the match is made available in the global character pointer "yytext", and its length in the global integer "yyleng". The action corresponding to the matched pattern is then executed and then the remaining input is scanned for another match. If no match is found, then the default rule is executed. the next character in the input is considered matched and copied to the standard output



Conclusion :-

Thus, we have studied lexical analyzer and implemented an application ~~fo~~ lexical analyzer to count total number of words, chard and line etc.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 06 [LEX Program]

Problem Satement: Write a program using Lex specifications to implement lexical analysis Phase of compiler to count no. of words, lines and characters of given Input file.

---

1. Code b3.l:

```
%{

int no_line=0;
 int no_space=0;
 int no_char=0;
 int no_words=0;
#include<string.h>

%}

%%

([a-zA-Z])+ {no_words++; no_char+=strlen(yytext);}

[" "] {no_space++;}

["\n"] {no_line++;}

. ;

%%


int yywrap(){

}

int main(int argc,char* argv[]){
  yyin=fopen("test.txt","r");

   yylex();

 printf("Total Spaces %d\n",no_space);
 printf("Total Words %d\n",no_words);
printf("Total Line %d\n",no_line);
  no_char+=no_space;

   printf("Total Char %d\n",no_char);

   fclose(yyin);

}
```

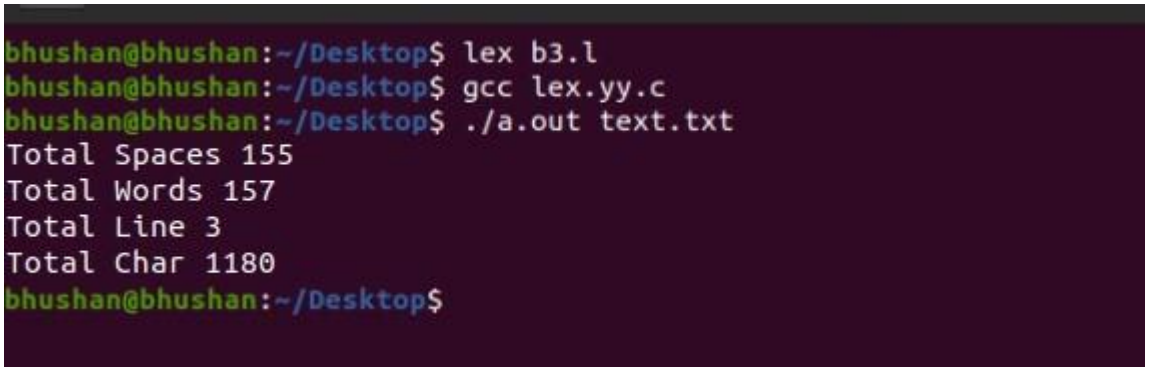
2. text.txt File:

// Content of text.txt File

The earliest foundations of what would become computer science predate the invention of the modern digital computer. Machines for calculating fixed numerical tasks such as the abacus have existed since antiquity, aiding in computations such as multiplication and division. Algorithms for performing computations have existed since antiquity, even before the development of sophisticated computing equipment.

Computer science, the study of computers and computing, including their theoretical and algorithmic foundations, hardware and software, and their uses for processing information. The discipline of computer science includes the study of algorithms and data structures, computer and network design, modeling data and information processes, and artificial intelligence. Computer science draws some of its foundations from mathematics and engineering and therefore incorporates techniques from areas such as queueing theory, probability and statistics, and electronic circuit design. Computer science also makes heavy use of hypothesis testing and experimentation during the conceptualization, design, measurement, and refinement of new algorithms, information structures, and computer architectures.

OUTPUT:



Assignment 7

Name - Bhushan Nikumbhe
Roll No. 22
Div - B
Page No. 1
Date

Aim :- Design Lex & Yacc program to validate type and syntax of variable declaration in java.

Problem Statement :- Write a program using Yacc specification to implement lexical analysis Phase of compiler to validate type and syntax of variable declaration in java.

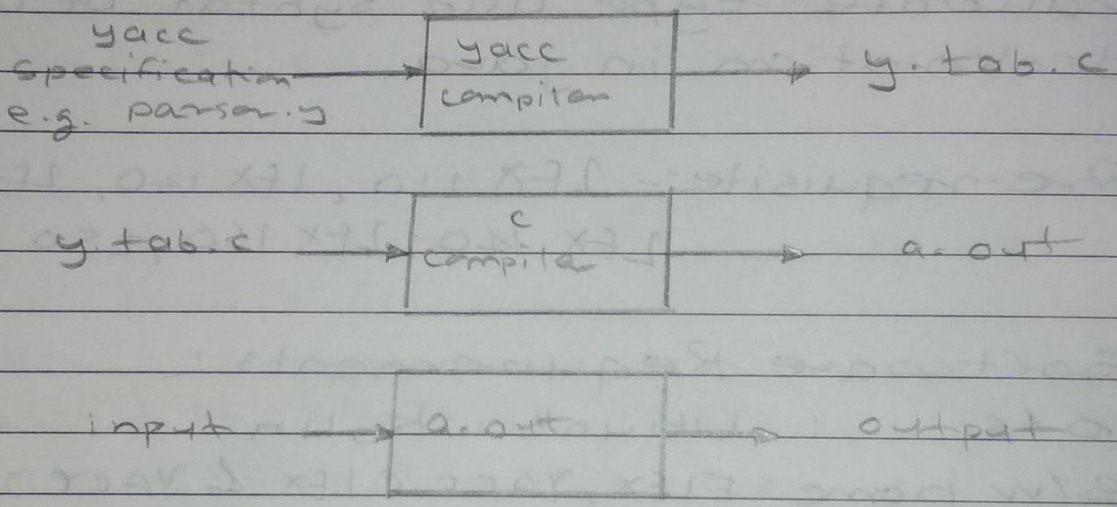
Pre-requisite :- LFX 110, LFX 120, LFX 130, LFX 140, LFX 160, 250

Software Requirements :-
O/S : Ubuntu Kylin
S/W name : FLEX, YACC CLEX & YACC)

Theory :-

Yacc (Yet another compiler-compiler) is a computer program for the UNIX operation system developed by Stephen C. Johnson. It is a Look Ahead left-to-Right (LALR) parser generated, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to Backus-Naur form (BNF). Yacc is supplied as a standard utility

on BSD and AT&T UNIX GNU based Linux distributions include Bison, a forward-compatible Yacc replacement. Yacc is one of the automatic tools for generating the parser program. Basically Yacc is a LALR parser generator.



Structure of a Yacc file:
A yacc file looks much like a lex file:

```
... definitions ...
%%
... rules ...
%%
... code ...
```

Definitions: As with lex, all code between %{ and %} is copied to the beginning of the resulting C file. Rules as with lex, a number of combinations of pattern

and actions. The patterns are now those of a context-free grammar, rather than of a regular grammar as was the case with lex code.

Translating, compiling and executing A Yacc Program:

The Lex program file consists of Lex specifications and should be named .l and the Yacc program consist of Yacc specifications and should be named .y following may be issued to generate the parser.

```
Lex <filename>.l
Yacc -d <filename>.y
cc lex.yy.c y.tab.c -ll
./a.out
```

Lexical Analyzer for YACC:

The user must supply a lexical analyzer to read the input stream and communicate tokens to the parser. The lexical analyzer is an integer-valued function called yylex. The function returns an integer, the token number, representing the kind of token read. If there is

a value associated with that token, it should be assigned to the external variable yylval.

```
yylex() {
  extern int yylval;
  int c;
  ...
  c = getchar();
  ... switch(c) {
    ... case '0':
    ...
    case '9':
  yylval = c - '0';
  return(DIGIT);
  ... }
...
```

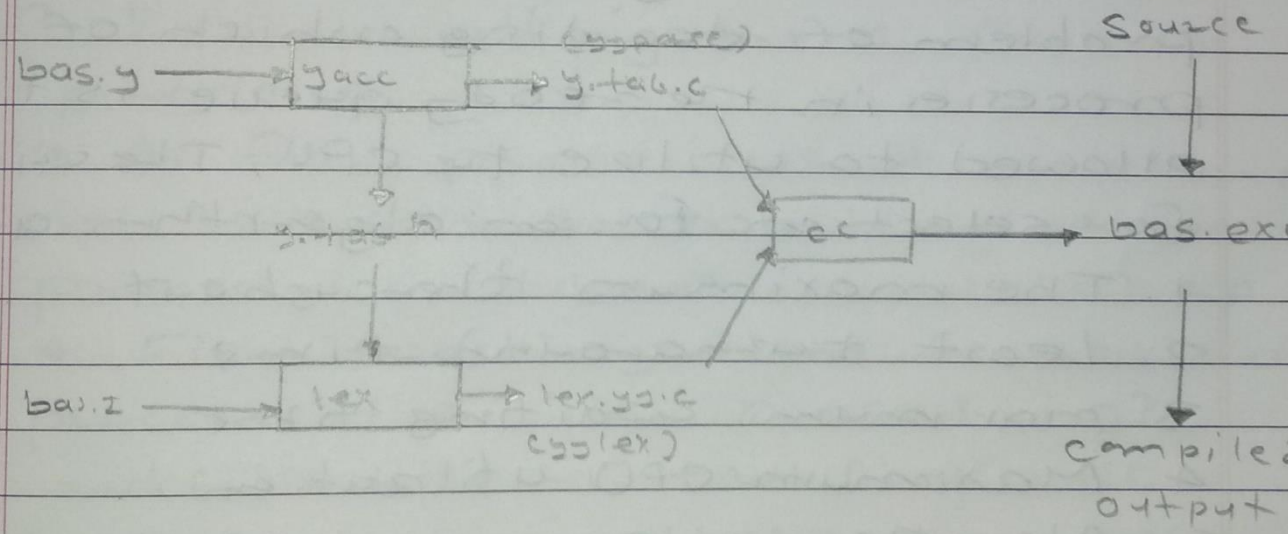
The intent is to return a token number of DIGIT, and a value equal to the numerical value of the digit. Provided that the lexical analyze code is placed in the program section of the specification file, the intentifier DIGIT will be defined as the token number associated with the token DIGIT.

Comparing Sentence Types :-

1. The simple sentence is an independent clause with one subject and one verb. For example : we are the indian.
2. The compound sentence is two or more independent clause, joined with comma semicolon & and conjunctions

Application :-

Yacc is used to generate parsers, which are integral part of compiler.



conclusion :-

Thus, we have studied lexical analyzer, Syntax analysis and implemented Lex & Yacc application fa Syntax analyzer to validate te given infix expression.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 07 [LEX Program]

Problem Satement: Write a program using YACC specifications to implement syntax analysis phase of compiler to recognize simple and compound sentences given in input file

---

1. Code b5.l:

```
%{
   #include<stdio.h>
   int simple=0;
%}
%%
[ \t\n][aA][nN][dD][ \t\n] {simple=1;}
[ \t\n][bB][uU][tT][ \t\n] {simple=1;}
[ \t\n][oO][rR][ \t\n] {simple=1;}
. ;
%%
int yywrap(){
}
int main(){
   printf("Enter sentence:  \n");     yylex();
if(simple==1){
      printf("compound\n\n");
   }    else{
      printf("simple\n\n");
   }    return 0;
}
```

OUTPUT

```
bhushan@bhushan:~/Desktop$ lex b5.l
bhushan@bhushan:~/Desktop$ gcc lex.yy.c
bhushan@bhushan:~/Desktop$ ./a.out
Enter sentence:
Hi Friends

simple

bhushan@bhushan:~/Desktop$ lex b5.l
bhushan@bhushan:~/Desktop$ gcc lex.yy.c
bhushan@bhushan:~/Desktop$ ./a.out
Enter sentence:
Hi friends or chai pilo

compound

bhushan@bhushan:~/Desktop$
```

Assignment 8

Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Page No. 1
Date

Aim :- Implement Job Scheduling algorithm 1) FCFS 2) Shortest job first 3) Priority 4) Round Robin

Problem statement :- Write a java program to implement following Scheduling algorithms : FCFS, SJF, Priority and Round Robin

Theory :-

Problem Explanation :-

CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allowed to utilize the CPU. The criteria for selection for an algorithm are,

1. The maximum throughput
2. Least turnaround time
3. maximum waiting time
4. Maximum CPU utilization
5. Also the variance in response time must be minimum. In ~~pre~~ preemptive job, a currently executing job can be removed and a new job can make its place, however in Non-preemptive this is not possible

1) First come First Serve.
This is the simplest CPU scheduling algorithm. The process that request the CPU first, is the one to which it is allocated first. The algorithm is implemented using a job queue. When a process Requests the CPU it is added at the tail of the job queue.

Implementation:-
1. Input the processes along with their burst time (bt)
2. Find waiting time for all processes.
3. As first process that comes need not to wait so waiting time for process 1 will be 0 i.e. wt[0] = 0
4. Find waiting time for all other processes i.e. for process i ->
wt[i] = bt[i-1] + wt[i-1]
5. Find turnaround time = waiting-time + burst time for all processes
6. Find average waiting time = total_waiting_time / no_of_processes
7. Similarly, find average turnaround time = total_turn_around_time / no_of_processes

2) SHORTEST JOB FIRST

This algorithm associates with it the length of the next CPU burst, when the CPU is available, it is assigned to that job with the smallest CPU burst. This algorithms provides the minimum average waiting time. The major problem with this knows the CPU burst of job.

Algorithm:-

1. Sort all the process in increasing order according to burst time.
2. Then simply, apply FCFS.

- SHORTEST REMAINING TIME:

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closet 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 needs to give preference.

3. 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.
- process with same priority are executed on first come served basis.
- Priority can be decided based on memory requirements, time requirement or any other resource requirement.

Implementation :-

1. First input the process with their burst time and priority.
2. Sort the processes, burst time and priority according to the priority.
3. Now simply apply FCFS algorithm.

4. ROUND ROBIN SCHEDULING

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation free as all processes get fair share of CPU.

- one of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for fixed slice of time at most.
- The disadvantage of it's more overhead of context switching.
- 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.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 08

Problem Satement: Write a Java program (using OOP features) to implement following scheduling algorithms:
FCFS , SJF (Preemptive), Priority (Non - Preemptive) and Round Robin (Preemptive)

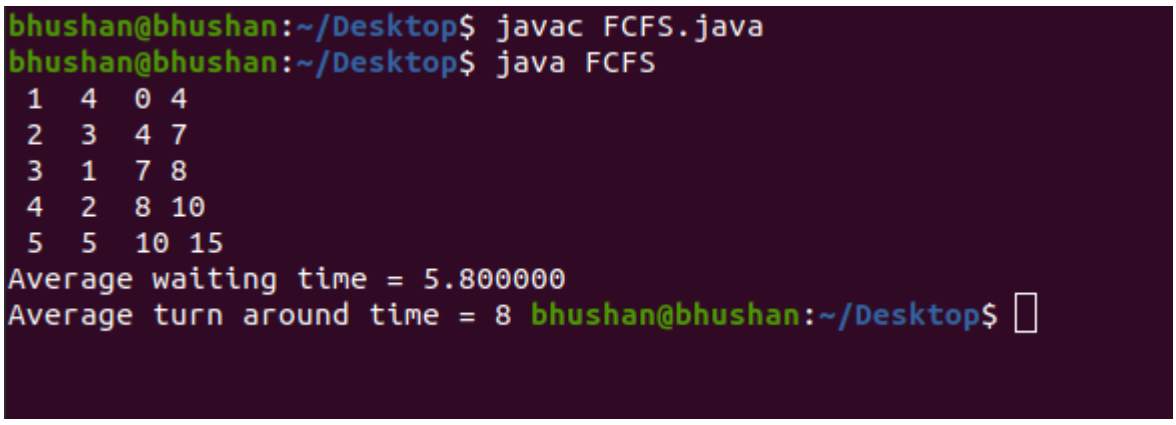
---

1. FCFS Program:

```
// Java program for implementation of FCFS

// scheduling import java.text.ParseException; class FCFS

{

        // Function to find the waiting time for all

        // processes

        static void findWaitingTime(int processes[], int n, int bt[], int wt[]) {

                // waiting time for first process is 0 wt[0] = 0;

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

        }

        // Function to calculate turn around time

        static void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

                // calculating 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

        static void findavgTime(int processes[], int n, int bt[]) { int wt[] = new int[n], tat[] =
                new int[n]; int total_wt = 0, total_tat = 0;

                //Function to find waiting time of all processes findWaitingTime(processes, n, bt, wt);

                //Function to find turn around time for all processes

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

                //Display processes along with all details

                System.out.printf("Processes \t Burst time \t Waiting" +" time Turn around time\n");

                // Calculate total waiting time and total turn
```

```
            // around time for (int i = 0; i < n; i++) { total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i]; System.out.printf(" %d ", (i + 1));

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

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

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

            }

            float s = (float)total_wt /(float) n;

            int t = total_tat / n;

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

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

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

        }

        // Driver code

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

            //process id's int processes[] = {1, 2, 3,4,5}; int n =
            processes.length;

            //Burst time of all processes int burst_time[] = {4,3,1,2,5};

            findavgTime(processes, n, burst_time);

        }

}
```

FCFS OUTPUT:



2. Shrtest Job First Program:

```
import java.util.*;


public class SJF { public static void main(String args[])

        {
```

```
Scanner sc = new Scanner(System.in); System.out.println ("enter no of process:"); int n = sc.nextInt(); int
pid[] = new int[n]; int at[] = new int[n]; // at means arrival time int bt[] = new int[n]; // bt means burst
time int ct[] = new int[n]; // ct means complete time int ta[] = new int[n]; // ta means turn around time
int wt[] = new int[n];  //wt means waiting time int f[] = new int[n];  // f means it is flag it checks process
is completed or not int st=0, tot=0; float avgwt=0, avgta=0;


for(int i=0;i<n;i++)
{
        System.out.println ("enter process " + (i+1) + " arrival time:"); at[i] = sc.nextInt();

        System.out.println ("enter process " + (i+1) + " brust time:"); bt[i] = sc.nextInt(); pid[i] =
        i+1; f[i] = 0;

}


boolean a = true; while(true)

{ int c=n, min=999; if (tot == n) // total no of process = completed process loop will be terminated break;

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

                /*
* If i'th process arrival time <= system time and its flag=0 and
burst<min
* That process will be executed first
                 */ if ((at[i] <= st) && (f[i] == 0) && (bt[i]<min))

                { min=bt[i]; c=i;

                }
        }


        /* If c==n means c value can not updated because no process arrival time<
system time so we increase the system time */ if (c==n) st++;

        else
        {
                ct[c]=st+bt[c]; st+=bt[c]; ta[c]=ct[c]-at[c];
                wt[c]=ta[c]-bt[c]; f[c]=1; tot++;

        }
}


System.out.println("\npid  arrival brust  complete turn waiting"); for(int i=0;i<n;i++)
```

```
{ avgwt+= wt[i]; avgta+= ta[i];
        System.out.println(pid[i]+"\t"+at[i]+"\
        t"+bt[i]+"\t"+ct[i]+"\t"+ta[i]+"\
t"+wt[i]);
    }
    System.out.println ("\naverage tat is "+ (float)(avgta/n)); System.out.println ("average wt is
    "+ (float)(avgwt/n)); sc.close();
  }
}
```

SJF OUTPUT:

```
bhushan@bhushan:~/Desktop$ javac SJF.java
bhushan@bhushan:~/Desktop$ java SJF.java
enter no of process:
4
enter process 1 arrival time:
0
enter process 1 brust time:
5
enter process 2 arrival time:
1
enter process 2 brust time:
3
enter process 3 arrival time:
2
enter process 3 brust time:
3
enter process 4 arrival time:
3
enter process 4 brust time:
1

pid arrival brust complete turn waiting
1       0       5       5       5       0
2       1       3       9       8       5
3       2       3       12      10      7
4       3       1       6       3       2

average tat is 6.5
average wt is 3.5
bhushan@bhushan:~/Desktop$
```

3. Priority Program:

```
import java.util.Scanner; public class Priority {
```

```
 public static void main(String args[]) {  Scanner s = new
Scanner(System.in);  int x,n,p[],pp[],bt[],w[],t[],awt,atat,i;

 p = new int[10];  pp = new int[10];  bt = new
int[10];  w = new int[10];  t = new int[10];  //n is
number of process

 //p is process

 //pp is process priority

 //bt is process burst time

 //w is wait time

 // t is turnaround time

 //awt is average waiting time

 //atat is average turnaround time

 System.out.print("Enter the number of process : ");

 n = s.nextInt();

 System.out.print("\n\t Enter burst time : time priorities \n");  for(i=0;i<n;i++)

 {

 System.out.print("\nProcess["+(i+1)+"]:");

 bt[i] = s.nextInt();  pp[i] = s.nextInt();  p[i]=i+1;

 }

//sorting on the basis of priority  for(i=0;i<n-1;i++)

 {

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

 {  if(pp[i]<pp[j])  {  x=pp[i];
pp[i]=pp[j];  pp[j]=x;  x=bt[i];
bt[i]=bt[j];  bt[j]=x;  x=p[i];  p[i]=p[j];
p[j]=x;  }

 } } w[0]=0; awt=0;

t[0]=bt[0]; atat=t[0]; for(i=1;i<n;i++)

 {  w[i]=t[i-1];  awt+=w[i];
t[i]=w[i]+bt[i];  atat+=t[i];

 }

//Displaying the process

 System.out.print("\n\nProcess \t Burst Time \t Wait Time \t Turn Around Time Priority \n"); for(i=0;i<n;i++)

 System.out.print("\n "+p[i]+"\t\t "+bt[i]+"\t\t "+w[i]+"\t\t"+t[i]+"\t\t "+pp[i]+"\n"); awt/=n; atat/=n;

 System.out.print("\n Average Wait Time : "+awt);

 System.out.print("\n Average Turn Around Time : "+atat);

 } }
```

Priority OUTPUT:

```
bhushan@bhushan:~/Desktop$ javac Priority.java
bhushan@bhushan:~/Desktop$ java Priority.java
Enter the number of process : 5

        Enter burst time : time priorities

Process[1]:7 2

Process[2]:6 4

Process[3]:4 1

Process[4]:5 3

Process[5]:1 0


Process         Burst Time      Wait Time       Turn Around Time Priority
 2              6               0               6               4
 4              5               6               11              3
 1              7               11              18              2
 3              4               18              22              1
 5              1               22              23              0
 Average Wait Time : 11
bhushan@bhushan:~/Desktop$
```

4. Round Robin Program:

```
import java.io.*; class RoundR {

public static void main(String args[])throws IOException

{

DataInputStream in=new DataInputStream(System.in); int i,j,k,q,sum=0;

System.out.println("Enter number of process:"); int
n=Integer.parseInt(in.readLine()); int bt[]=new int[n]; int wt[]=new int[n];
int tat[]=new int[n]; int a[]=new int[n];

System.out.println("Enter brust Time:"); for(i=0;i<n;i++)

{

System.out.println("Enter brust Time for "+(i+1));

bt[i]=Integer.parseInt(in.readLine());

}

System.out.println("Enter Time quantum:");
q=Integer.parseInt(in.readLine()); for(i=0;i<n;i++) a[i]=bt[i];
for(i=0;i<n;i++)

wt[i]=0; do { for(i=0;i<n;i++)

{

if(bt[i]>q) {
```

```
bt[i]-=q; for(j=0;j<n;j++) { if((j!=i)&&(bt[j]!=0))
wt[j]+=q; }

}

else { for(j=0;j<n;j++) { if((j!=i)&&(bt[j]!=0))

wt[j]+=bt[i]; }

bt[i]=0;

} } sum=0; for(k=0;k<n;k++)
sum=sum+bt[k];

} while(sum!=0); for(i=0;i<n;i++)
tat[i]=wt[i]+a[i];

System.out.println("process\t\tBT\tWT\tTAT"); for(i=0;i<n;i++)

{

System.out.println("process"+(i+1)+"\t"+a[i]+"\t"+wt[i]+"\t"+tat[i]);

}

float avg_wt=0; float avg_tat=0;

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

{

avg_wt+=wt[j];

}

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

{

avg_tat+=tat[j]; }

System.out.println("average waiting time"+(avg_wt/n)+"\n Average turn around time"+(avg_tat/n));

}

}
```

Round Robin OUTPUT:

```
bhushan@bhushan:~/Desktop$ javac RoundR.java
Note: RoundR.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
bhushan@bhushan:~/Desktop$ java RoundR
Enter number of process:
4
Enter brust Time:
Enter brust Time for 1
4
Enter brust Time for 2
5
Enter brust Time for 3
6
Enter brust Time for 4
7
Enter Time quantum:
4
process         BT      WT      TAT
process1        4       0       4
process2        5       12      17
process3        6       13      19
process4        7       15      22
average waiting time10.0
 Average turn around time15.5
bhushan@bhushan:~/Desktop$
```

Assignment 9

Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Page No. 1
Date

Aim :- Bankers algorithm for deadlock detection and avoidance

Problem Statement :- Write a java program to implement Banker's Algorithm

Theory :-

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "S-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following data structures are used to implement the Banker's Algorithm:
Let 'n' be the number of processes in the system and 'm' be the number of resources types.

Available :

- It is a 1-d array of size 'm' indicating the number of available resources of each type.

- Available [j] = k means there are 'k' instances of resource type Rj

Max:
- It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system
- Max [i,j] = k means process Pi may request at most 'k' instance of resource type Rj

Allocation:
- It is a 2-d size of 'n*m' that defines the number of resources of each type currently allocated to each process.
- Allocation [i,j] = k means process Pi is currently allocated 'k' instances of resource type Rj

Need:
- It is a 2-d array of size 'n*m' that indicates the remaining resource need each process.
- Need [i,j] = k means process Pi currently need 'k' instance of resource type Rj for its execution

- Need [i, j] = Max [i, j] - Allocation [i, j]
- Allocation specifies the resource currently allocated to process P_i and Need, specifies the additional resources that process P_i may still request to complete its task.
- Banker's algorithm consist of Safety algorithm and Resource request algorithm

- Safety Algorithm:
The algorithm for finding out wheather or not a system is in a safe state can be described as follows:

1) Let work and finish be vectors of length 'm' and 'n' respectively
Initialize: work = Available
Finish [i] = false; for i = 1, 2, 3, 4, ... n

2) Find an i such that both
a) Finish [i] = false
b) Need$_i$ <= work
if no such i exist goto step 4

3) work = work + Allocation [i]
Finish [i] = true
goto step [2]

4) if finish [i] = true all for all i
then the system is in a safe state.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 09

Problem Satement: Write a Java program to implement Banker's Algorithm

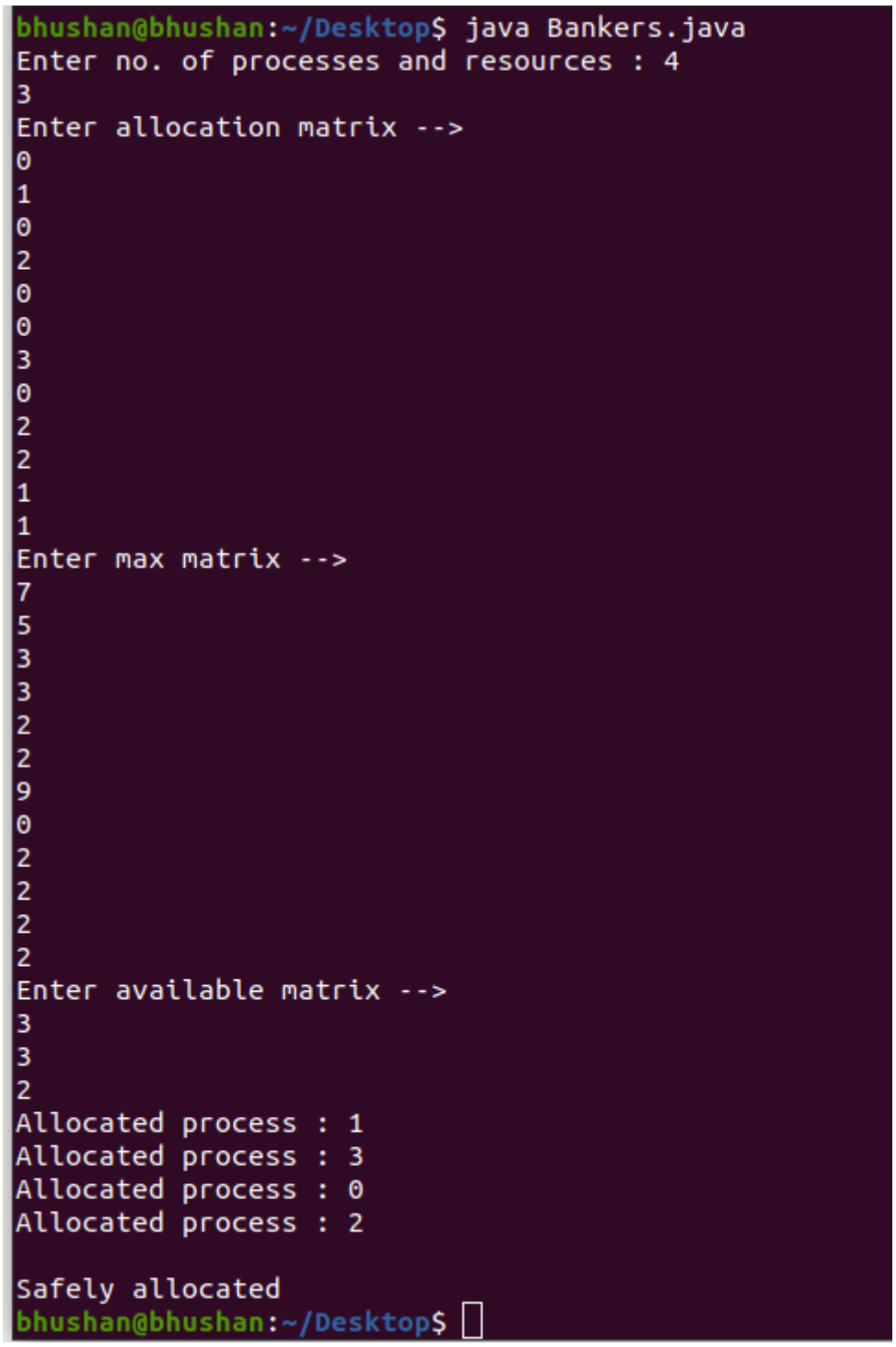
---

1. Banker's Algorithm Program:

```
import java.util.Scanner; public class Bankers{  private int
need[][],allocate[][],max[][],avail[][],np,nr;


 private void input(){

 Scanner sc=new Scanner(System.in);

 System.out.print("Enter no. of processes and resources : ");  np=sc.nextInt(); //no. of
process  nr=sc.nextInt(); //no. of resources  need=new int[np][nr]; //initializing arrays
max=new int[np][nr];  allocate=new int[np][nr];  avail=new int[1][nr];


 System.out.println("Enter allocation matrix -->");  for(int i=0;i<np;i++)
for(int j=0;j<nr;j++)

 allocate[i][j]=sc.nextInt(); //allocation matrix


 System.out.println("Enter max matrix -->");  for(int i=0;i<np;i++)
for(int j=0;j<nr;j++)

 max[i][j]=sc.nextInt(); //max matrix


 System.out.println("Enter available matrix -->");  for(int j=0;j<nr;j++)
avail[0][j]=sc.nextInt(); //available matrix


 sc.close();

 }


 private int[][] calc_need(){  for(int i=0;i<np;i++)  for(int j=0;j<nr;j++)
//calculating need matrix  need[i][j]=max[i][j]-allocate[i][j];


 return need;  }  private boolean check(int i){

 //checking if all resources for ith process can be allocated  for(int j=0;j<nr;j++)
if(avail[0][j]<need[i][j])  return false;


 return true;  }  public void isSafe(){  input();  calc_need();  boolean
done[]=new boolean[np];  int j=0;  while(j<np){ //until all process
allocated  boolean allocated=false;  for(int i=0;i<np;i++)  if(!done[i] &&
check(i)){ //trying to allocate  for(int k=0;k<nr;k++)

 avail[0][k]=avail[0][k]-need[i][k]+max[i][k];
System.out.println("Allocated process : "+i);  allocated=done[i]=true;
j++;  }  if(!allocated) break; //if no allocation
```

```
 } if(j==np) //if all processes are allocated
System.out.println("\nSafely allocated");  else

 System.out.println("All proceess cant be allocated safely");

 }


 public static void main(String[] args) {  new Bankers().isSafe();

 } }
```

OUTPUT:



Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Assignment 10

Page No. 1
Date

Aim - Implement UNIX System calls like for process management

Problem statement :- To write a program to implement UNIX system calls like for process management

Pre-requisites :-
1. Explain concept of system call
2. Explain state diagram working of new process

Software Requirements :-
O/S : Ubuntu kylin
S/W name: C Turbo or Gcc.

Objectives :-
1) To understand UNIX System call
2) To understand concept of process management
3) Implementation of some system call of OS

Theory :-
SYSTEM CALL :
- When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource.

This in done via something called a System call.
- When a program makes a system call, the mode is switched from user mode to kernel mode. This is called context switch.
- Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, System calls are made by the user level programs in the following situations:
- Creating, opening, closing and deleting files in the file system
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.

To understand System calls, first one needs to understand the difference between kernel mode and user mode of a CPU. Every modern operating system supports these two modes.

KERNEL MODE :-

- When CPU is in Kernel mode, the code being executed can access any memory address and any hardware resource.
- In user mode, if any program crashes only that particular program is halted
- That means the system will be in a safe state even if a program in user mode crashes
- Hence, most programs in an OS run in user mode.

SYSCALLS FOR PROCESSES:

- pid_t fork(void)
  - ✓ create new child process, which is a copy of the current process
  - ✓ Parent return value is the PID of the child process
  - ✓ child return value is 0.

- int execl(char *name, char *arg0, ... (char*)0)
  - ✓ change program image of current process
  - ✓ Reset stack and free memory
  - ✓ start at main()
  - ✓ Also see other variants: execlp(), execv(), etc.

- pid_t wait (int *status)
  - wait for a child process (any child) to complete.
  - Also see waitpid() to wait for a specific process.

- void exit (int status)
  - Terminate the calling process.
  - can also achieve with a return from main().

- int kill (pit_t pid, int sig)
  - send a signal to a process
  - send SIGKILL to ~~force termination~~

UNIX SYSTEM CALLS :-

- PS command :
  The ps command is used to provide information about the currently running processes, including their process identification numbers (PIDs). A process, also referred to as a task is an executing instance of a program. Every process is assigned a unique PID by the system. The basic syntax of ps is : ps [options]

- Fork command()

The Fork() system call is used to create processes. When a process makes a Fork() call, an exact copy of the process is created. Now there are two processes, one being the parent process and the other being the child process.

- Join command:

The join command in UNIX is a command line utility for joining lines of two files on a command field. It can be used to join two files by selecting fields within the line and joining the files on them. The result is written to standard output.

Join syntax: join [option]... file1 file2

- Execl() command

The execl() system call is also used to create processes. But there is one big difference between fork() and execl() calls. The fork() call creates a new process while preserving the parent process. But the execl() call replaces the address space, text segment, data segment etc. of the current process with the new process.

- Wait() command.

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait System call instruction.

child process terminate due to any of these.

- ✓ It calls exit();
- ✓ It returns (an int) from main.
- ✓ It receives a signal whose default action is to terminate.

Conclusion :-

Thus, the process System call program is implemented and studied various System calls.

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 10 [UNIX System Calls]

Problem Satement: To write a program to implement UNIX system calls like for process Management.

---

1. Code:

Problem Statement : Write a C program to create a child process using fork system call. Display Status of running processes used in child process(EXEC) & terminate child process before completion of parent task(wait).

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h> #include<sys/types.h>


int main()

{ pid_t pid , ppid , p_status ;

        int status ;

        printf("parent process created \n"); pid = fork();

        if(pid ==0)

        {

                printf("child created succesfull\n"); printf("child process id :
                %d \n", pid); sleep(10); printf("child after sleep \n");
                execlp("/bin/ps","ps",NULL);


                printf("child terminating\n"); exit(0);

        }

        else

        { printf("parent still executing"); p_status = wait(&status);
                printf("status : %d \n",status); printf("p_status :%d
                \n",p_status); sleep(10);

                printf("parent after sleep\n"); ppid = getppid();

                printf("parent process id : %d\n",ppid); printf("parent terminating\n");

                exit(0);

        }

        return 0;

}
```

OUTPUT:

```
bhushan@bhushan:~/Desktop$ gcc Exp10_Spos.cpp
bhushan@bhushan:~/Desktop$ ./a.out
parent process created
parent still executingstatus : 32766
p_status :0
child created succesfull
child process id : 0
child after sleep
parent after sleep
parent process id : 5332
parent terminating
bhushan@bhushan:~/Desktop$     PID TTY          TIME CMD
   5332 pts/0    00:00:00 bash
   5350 pts/0    00:00:00 ps

bhushan@bhushan:~/Desktop$
```

Name- Bhushan Nikumbhe
Roll No. 22
Div - B

Assignment 11

Page No. 1
Date

Title :- Study assignment ~~in android~~ on process scheduling algorithm in Android and Tizen.

Objectives:
1. To understand Android OS - To understand Tizen OS - To understand concept of process management.
2. Problem statement: Study assignment on process scheduling algorithms in Android and Tizen
3. Outcomes : After completion of this assignment students will be able to:- Knowledge of Android and tizen OS - Study of process management in android and tizen OS. - Application of android and tizen OS.
4. Software Requirements : Android SDK
5. Hardware Requirement :- M/C lenovo Think center M700 ci3, 6100, 6th Gen, H81, 4GB RAM, 500 GB HDD.
6. Theory concepts :-

Android OS :-

Android is a mobile operating system developed by google based on a modified version of ~~Kernel~~ Linux kernel and other open source Software and designed primarily for touchscreen mobile devices such as smartphones and tablets

In addition, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of android are also used on game consoles, digital cameras, PCs and other electronics. The android is an open source operating system means that it's free and any one can use it The android has got millions of appl available that can help you managing your life one or other way and it is available low cost in market at that reasons android is very popular. The android development supports with the full java programming language.

- Some android versions:
  - Gingerbread (2.3)
  - Honeycomb (3.0)
  - Ice Cream Sandwich (4.0)
  - Jelly Bean (4.3 / 4.2 / 4.1)
  - Kitkat (4.4)
  - Loppipop (5.0)
  - Marshmallo (6.0)
  - Nougat (7.0)
  - Oreao (8.0)

Advantages :-
1. Suppat 2D & 3D Graphics.
2. Suppat multiple language.
3. Java suppat.
4. Faster web browser.
5. Suppat audio, video etc.

Disadvantages :
1. Slow response.
2. Heat.
3. Advertisement etc.

Tizen OS :-
- Tizen OS is a mobile operating system developed by Samsung that runs on a wide range of Samsung devices, including smartphones, tablets, in-vehicle infotainment (IVI) devices, Smart televisions, Smart cameras, smartwatches, Blu-ray players, smart home appliances and robotic vaccum cleaners.
- Samsung announced in November 2016 that they would be collaborating with Microsot to bring .Net suppat to Tizen.
- As of 2017 Tizen is second largest smartwatch platform, behind watchOS and and ahead of android wear

Android vs Tizen Operating System:

- Easy and Convenient Navigation: Scrolling and navigation becomes smooth with Tizen.
- Fast and Lightweight: Tizen operating system is easy to operate and fast as compared to Google's Android wear.
- Visual Effects: Tizen extends 3D visual effects of various gaming apps installed on the device.
- UI: TouchWiz UI
- Resizable boxes: One of the amazing features of Tizen is its ability to dynamically resize the icons on screen to display more information or less.
- Supporting Devices: Tizen is being used in Smart TV's, refrigerator, smart watches, Smart phones, washing machines, light bulbs, vaccum cleaners while Android is visible only in smart phones, washing computes or smart vehicles.
- IoT Devices: Tizen 3.0 is compatible with Artik cloud which will extend cloud services for IOT devices.
- Battery consumption: Samsung's devices with Tizen OS consume less power than Android devices according to mobile experts.

Advantages of using Tizen OS

- It is open source operating system.
- The OS is compatible with various mobile platform, Application built on Tizen can be launched on iOS and Android too with few changes.
- The Tizen OS is so Flexible to offer many applications and adapt too, with little changes.
- Immense personalization capability supported by ARM x86 processor.

Process scheduling algorithm in Android and Tizen OS.

Normal Scheduling :-

Android is based on Linux and uses the Linux kernel's scheduling mechanism for determining scheduling policies. Process can be given an initial priority from 19-20. This priority will assure that higher priority processes will get more CPU time when needed. These level are however dynamic, low level priority tasks that to do not consume their CPU time will fine their dynamic priority increased.

Real-time Scheduling :-

The standard Linux kernel provides two real-time scheduling process policies, SCHED_FIFO and SCHED_RR. The main real-time policy is SCHED_FIFO task start running, it continues to run until it voluntarily yields the processor, blocks or is preempted by a higher-priority real-time task.

Thread Scheduling :-

A thread Scheduler decides which threads in the Android System should run, when, and for how long Android's thread scheduler uses two main factor to determine the scheduling :

- Niceness Values
- Control Groups (CGroup) Niceness values
- a thread with higher niceness value will run less often than those with a lower niceness value.

Priority Based Pre-emptive Task Scheduling for Android Operating System :-

The key concept present in any Operating system which allow the system to support multitasking, multiprocessing etc. is Task scheduling. Task scheduling is the core which refers to the way

the different processes are allowed to share common CPU.

Dynamic priority pre-emptive scheduling earlies-deadline first scheduling: job's priority is inversely proportional to it's absolute deadline. The difference beth deadline monotonic scheduling and earliest-deadline first scheduling is that DM is a static priority algorith EDF is a dynamic priority algorithm

Conclusion :-

Thus, I have studied concept of process scheduling of Android and Tizen operating system.

Name - Bhushan Nikumbhe
Roll No. 22
Div - B

Assignment 12

Page No. 1
Date

Aim :- Implementing page replacement algorithm 1) LRU
2) Optimal

Problem statement :- To write a java program (using OOP feature) to implement LRU & optimal algorithm for page replacement.

Pre-requisites :-
1. Explain the concept of virtual memory
2. Define page replacement algorithm: LRU & Optimal
3. Explain address translation in paging system.
4. Explain Belady's Anomaly.

Theory :-
Whenever there is a page reference for which the page needed in memory, that event is called page fault or page fetch or page failure situation. In such case we have to make space in memory for this new page by replacing any existing page. But we cannot replace any page, we have to replace a page which is not used currently. There are some algorithms based on them. we can select appropriate

page replacement policy. Designing appropriate algorithms to solve this problem is an important task because disk I/O is expensive.
There are several algorithms to achieve
1) Last recently used (LRU)
2) Optimal

1) LRU page replacement:
The main difference between FIFO and optimal page replacement is that the FIFO algorithms uses the time when the page was brought in to memory and the optimal algorithm uses the time when a page is to be used. If we use the recent past as an approximation of then future when we will replace the page that has not been used for the longest period of time. This approach is called as least recently used (LRU) algorithm.

1	2	3	4	5	6	7	8	9	10	11	12
	7	0	1	2	0	3	0	4	2	3	0
	7	7	7	2	2	2	2	4	4	4	0
		0	0	0	0	0	0	0	0	3	3
		1	1	1	3	9	3	2	2	2	2
	+	+	+	+		+		+			+

2) Optimal page replacement:
The algorithm has lowest page fault rate of all algorithm. This algorithm state that: Replace the page which will not be used for longest period of time i.e. future knowledge of reference string is required.
✓ Often called Belady's Min Basis idea: replace the page that will not be referenced for the longest time.
✓ Impossible to implement.

consider following reference string: 0,2,1,6,4,0,1,0,3,1,2,1
compulsory misses ——→ x x x x x x

0		0		3
2	4 →	2	3 →	2
1		1		1
6		4		4

Fault rate = 6/12 = 0.50
with reference string, this is best we can hope to do.

Algorithm for LRU:
Let capacity be the number of pages that memory can hold. Let set be the current set pages in memory.

1. Start traversing the pages.
i) If set holds less pages than capacity.
a) Insert page into the set one by one until the size of set reaches capacity or all page request are processed.

b) Simultaneously maintain the recent occurred index of each page in a map called indexes.
c) Increment page fault
ii) Else
If current page is present in set, do nothing
Else
a) Find the page in the set that was least recently used. We find it using index array. We basically need to replace the page with minimum index.
b) Replace the found page with current page.
c) Increment page faults.
d) Update index of current page.

2. Return page faults.

Algorithm For Optimal
1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. compare counter label and stack
6. select the optimal page by counter valu
7. Stack them according the selection.
8. Print pages with fault pages.
9. Stop the process

Name: Bhushan Vishram Nikumbhe **DivRoll**: TE-B-22

Assignment No. 12

Problem Satement: To write a java program (using OOP feature) to implement LRU & Optimal algorithm for Page Replacement.

---

1. LRU (Last Recently Used) Program:

```
import java.io.BufferedReader; import
java.io.InputStreamReader; import java.util.Arrays;

public class LRU

{


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

                int hit=0; int miss=0;

                BufferedReader br=new BufferedReader(new

InputStreamReader(System.in));

                System.out.println("Enter total no of frames"); int
                noFrames=Integer.parseInt(br.readLine());

                int[] frames=new int[noFrames]; int[] lruTime=new
                int[noFrames];

                System.out.println("Enter total no of pages"); int totalPages =
                Integer.parseInt(br.readLine());

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

                        System.out.println("Enter page value"); int page=

                        Integer.parseInt(br.readLine()); int searchIndex=isPresent(frames,

                        page );

                                if(searchIndex!=-1){
//                              page fonud

                                hit++; lruTime[searchIndex]=i; System.out.println("Page
                                Hit");

                        } else{

                                System.out.println("Page Miss"); miss++;

//                              page not found

                                        int emptyindex=isEmpty(frames);
                                        if(emptyindex!=-1){
//                                       if frame is empty

                                                frames[emptyindex]=page;
```

```
                                        lruTime[emptyindex]=i;

                                    } else{

                                //user lru algo to find replace location int minLocationIndex=lru(lruTime);

                                                                    System.out.println("Replace "+
frames[minLocationIndex]);

                                        frames[minLocationIndex]=page; lruTime[minLocationIndex]=i;

                                            }

                                }

                        }


                        System.out.println("Total page hit" + hit);
                        System.out.println("Total Page miss " + miss);
                        System.out.println(Arrays.toString(frames));

        }

        public static int lru(int[] lruTime){ int min = 9999; int index = -1; for(int
                        i=0;i<lruTime.length;i++){

                            if(min>lruTime[i]){ min=lruTime[i]; index=i;

                                    }

                        }

                            return index;
        }

        public static int isEmpty(int[] frames){

                            for(int i=0;i<frames.length;i++)
                { if(frames[i]==0){ return i;

                        }

                }

                        return -1;
        }

                        public static int isPresent(int[] frames, int search){

                for(int i=0;i<frames.length;i++){ if(frames[i]==search) return
                        i;

                }

                return -1; }


}
```

OUTPUT:

```
bhushan@bhushan:~/Desktop$ javac LRU.java
bhushan@bhushan:~/Desktop$ java LRU.java
Enter total no of frames
3
Enter total no of pages
8
Enter page value
1
Page Miss
Enter page value
0
Page Hit
Enter page value
2
Page Miss
Enter page value
0
Page Hit
Enter page value
3
Page Miss
Enter page value
1
Page Hit
Enter page value
2
Page Hit
Enter page value
0
Page Miss
Replace 3
Total page hit4
Total Page miss 4
[1, 2, 0]
bhushan@bhushan:~/Desktop$
```

2. Optimal Replacement Program:

```
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: