SINHGAD COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

SUBJECT CODE: 310248

LAB MANUAL
Laboratory Practice-I
(System Programming & Operating System)

RMD Sinhgad School of Engineering

Department of Computer Engineering

310248: Laboratory Practice-I

Teaching Scheme: Examination Scheme: Practical: 4 Hrs/Week Term work: 25 Marks Credits: 02

Practical: 25 Marks

List of Laboratory Assignments

Assignment No.: 01

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

Objectives:

1. To study the design and implementation of 1st pass of two pass assembler.
2. To study the categorized instruction set of assembler.
3. To study the data structure used in assembler implementation.

Theory:

1. Explain various Data and Instruction formats of assembly language programming. 2. Explain the design of Pass- I of assembler with the help of flowchart and example. 3. Discuss various Data structure used in Pass-I along with its format and significance of each field.

1

Q 1] Explain various Data and instruction formats of assembly language programming

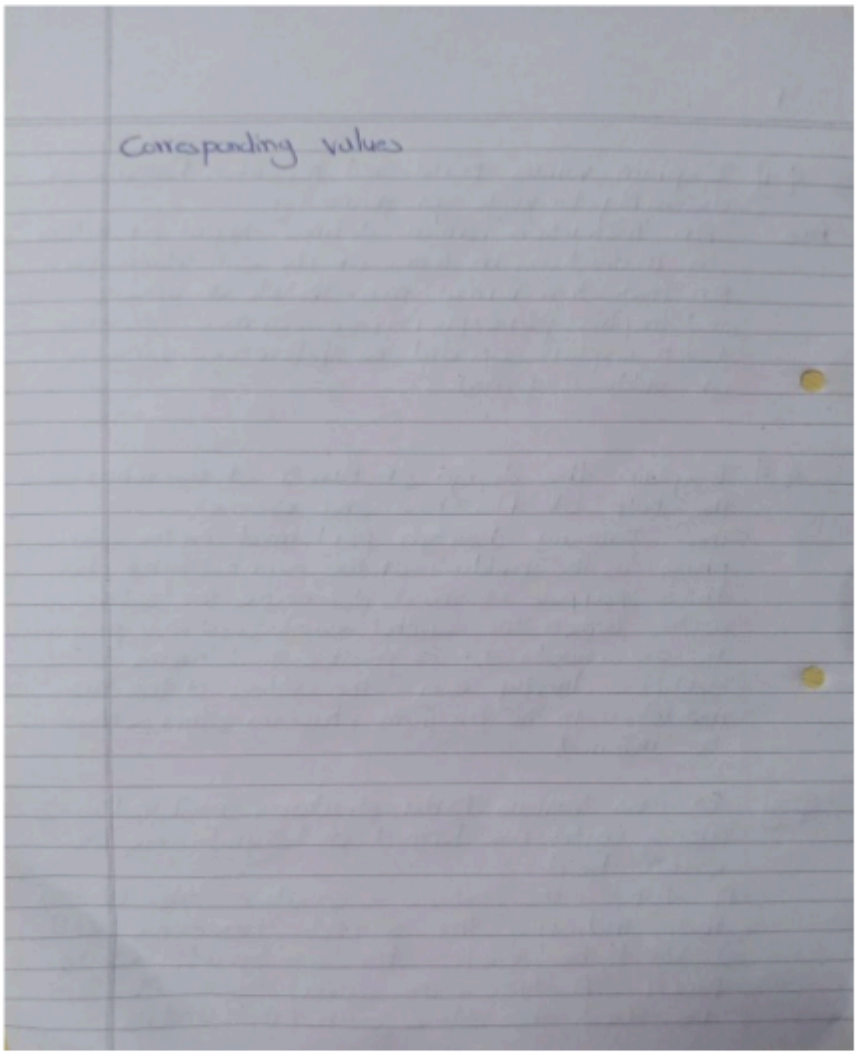
Ans - An instruction format defines layout of bits of an instruction, in terms of its constituent parts. An instruction format must include an opcode and implicity or explicity, zero or more operands. Each explicit operand is referenced using one of addressing modes.

Q 2] Explain the design of Pass-I of assembler with the help of flowchart and example.

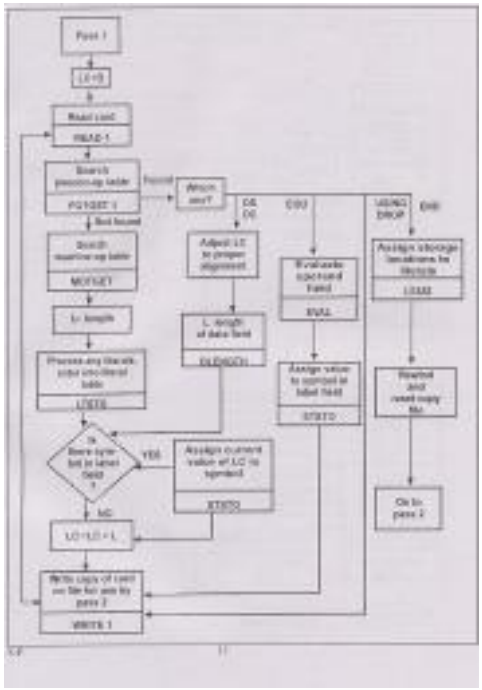
The primary function performed by the analysis phase is the building of the symbol table, for this purpose it must determine the addresses with which the symbol names used in a program are associated. It is possible to determine some address directly e.g. the address of the first instruction in the program, however others must be inferred.

Q 3] Discuss various Data structures used in Pass-I along with its formed and significance of each field.

A table, the pseudo - operation table (POT) that indicates the symbolic mnemonic and action to be taken for each pseudo -OP in Pass-1. A table, the symbol table (ST) that is used to store each label and its


Corresponding values

Algorithm/Flowchart:



Design diagrams (if any):

1. Class Diagram
2. Use case Diagram
3. ER Diagram

Input:

Source code of Assembly Language

```
SAMPLE START 100
        USING *, 15
        L 1, FOUR
```

```
                A 1, =F'3'
                ST 1, RESULT  SR
                1, 2
                LTORG
                L 2, FIVE
                A 2, =F'5'
                A 2, =F'7'
FIVE DC F'5'
FOUR DC F'4'
RESULT DS 1F
                END
```

Output:

```
                100 SAMPLE START 100  100 USING *, 15
                100 L 1, FOUR
        104 A 1, =F'3'  108 ST 1, RESULT
                112 SR 1, 2
                114 LTORG
                124 L 2, FIVE
128 A 2, =F'5'  132 A 2, =F'7'  136 FIVE DC F'5'
    140 FOUR DC F'4'  144 RESULT DS 1F
                152 5
                156 7
                160 END
```

Machine Opcode Table (MOT)

Mnemonic	Hex / Binary Code	Length (Bytes)	Format
L	5A	4	RX
A	1B	4	RX
ST	50	4	RX
SR	18	2	RR

Pseudo Opcode Table (POT)

Pseudo op	Address / Name of Procedure to implement pseudo operation
START	PSTART
USING	PUSING

DC	PDC
DS	PDS
LTORG	PLTORG
END	PEND

Symbol Table (ST)

Sr. No	Symbol name	Address	Value	Length	Relocation
1	SAMPLE	100	--	160	R
2	FIVE	136	5	4	R
3	FOUR	140	4	4	R
4	RESULT	144	__	4	R

Literal Table (LT)

Sr. No	Literal	Address	Length
1	3	120	4
2	5	152	4
3	7	156	4

Instructions :
Not specific

Test Cases:

1. Check syntax of instruction (Correct and wrong)
2. Symbol not found
3. Wrong instruction
4. Duplicate symbol declaration
5. Test the output of program by changing value of START pseudo opcode. 6. Test the output of program by changing position of LTORG pseudo-op.

Software Requirement:

1. Fedora
2. Eclipse
3. JDK

Hardware Requirement:

Not specific

Frequently Asked Questions:

1. What is two pass assembler?
2. What is the significance of symbol table?
3. Explain the assembler directives EQU, ORIGIN.
4. Explain the assembler directives START, END, LTORG.
5. What is the use of POOLTAB and LITTAB?
6. How literals are handled in pass I?
7. What are the tasks done in Pass I?
8. How error handling is done in pass I?
9. Which intermediate data structures are designed and implemented in PassI? 10. What is the format of a machine code generated in PassII?
11. What is forward reference? How it is resolved by assembler?
12. How error handling is done in pass II?
13. What is the difference between IS, DL and AD?

14. What are the tasks done in Pass II?

Group A - Assignment no 1

Q.1] What is two Pass assembler?
The assembler goes through the program one line at a time, and generates machine code for that instruction then the assembler proceeds to the next instruction. In this way, the entire machine code program is created.

Q.2] What is the significance of symbol table?
Symbol table is an important data structure created and maintained by the compiler in order to keep track of semantics of variables i.e it stores information about scope and binding information about names, information about instances of various entities such as variable and function names, classes, objects, etc

Q.3] Explain the assembler directives EQU, Origin
EQU - The EQU directive only tells the assembler to substitute a value for a symbol or label and doesn't involve any type of ROM or RAM, EQU doesn't involve directives are typically placed at the beginning of an assembly program
ORIGIN - The directives help the program in getting compiled and hence won't be there in the object code, This basically is used to replace the variable with a constant value

Q.4] Explain the assembler directives START, END, LTORG

START - Define the start of the first control section in a program

END - END of the assembler module or control section

LTORG - Begin the literal pool

Q.5] What is the use of POOLTAB and LITTAB?

POOLTAB - Addresses of different literal pools is maintained using the auxiliary table POOLTAB, AT Any stage, the current literal pool is the last pool in LITTAB

LITTAB - LTORG directives place the all constants at consecutive memory locations If we are not using LTORG then all literals are placed after END in memory

Q.6] How literals are handling in Pass-I.

Literals are replace table terms because the address of the literal, rather than the literal-generated constant itself is assembled in the statement that reference a literal. The assembler generates the literals, collects them and places them in a specific area of storage as explained under literal pool.

Q.7] What are the tasks done in Pass I
Define Symbols and literals and remember them in Symbol table and literal table respectively
keep track of location counter
process pseudo-operation

Q.8] How error handling is done in Pass e
There are four ways to handle errors in software you can propgate the error from function to the code that call that function, handle the error using do-catch statement, handle the error will not occurs.

Q.9] Which intermediate data structure are designed and implemented in pass I
Assembler implementation is based on two major data structures - operation table (OPTAB) and symbol table (SYMTAB) for each mnemonic N and OPTAB contains

Q.10] What is the format of a machine code generated in pass II ?
Converting Symbolic machine-opcodes into their respective bit configuration machine understandable form. It stores all machine-opcodes in MOT table (Opcode table) with symbolic code, their length and their bit configuration

Q.11] What is forward reference? How it is resolved by assembler?
A forward reference occurs when a label is used as an operand, for example as a branch target, earlier in the code than the definition of the label, the assembler cannot known the address of the forward reference label until it reads the definition of the label.

Q.12] How error handling is does in Pass II
Ext without errors, one or more non-fetal errors found during pass 2, Assembly quite by user, the program terminated because of fetal error, fatal error wrong python version is used

Q.13] What is the difference between DL and New AD?
Distribution groups are used for sending email notification to a group of people security groups are used for grouping access to resources such as share point it is mail enabled security groups are used for granting access to resource such as share point and emailing notifications to those users

Q.15] What are the tasks done in pass II &
Pass-2 - generates object code by converting
symbolic op-code into respective numeric
opcode, generate data for literals and
looks for values of symbols

Code –

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

class MNT {
String macro_name;
int mdt_ind;

MNT(String s, int i) {
macro_name = s;
mdt_ind = i;
}
}

public class MacroP1 {
 static List<MNT> mnt;
 static List<String> mdt;
 static int mdtc;
 static int mntc;
 static List<String>  ala;

 static Scanner sc;
public static void main(String[] args) throws Exception {
 MacroP1 p1=new MacroP1();
 p1.init();
 File f=new File("ip.asm");
 sc=new Scanner(f);
 PrintWriter op=new PrintWriter(new FileOutputStream("op.txt"));
       while(sc.hasNextLine())
       {
          String line=sc.nextLine();
          if(line.equalsIgnoreCase("MACRO"))
          {
                  //definition found
                  p1.Process_Def(line);
          }
          else{
              op.println(line);
              op.flush();
          }
       }//end of while
```

```
        // Display All Tables
        System.out.println("ALA:");
           showAla(1);
           System.out.println("\nMNT:");
           showMnt();
           System.out.println("\nMDT:");
           showMdt();



} //end of main
static void showAla(int pass) throws Exception {

 for(String l : ala) {
            System.out.println(l);

 }
}

static void showMnt() throws Exception {

 int i=0;
 for(MNT l : mnt) {
           System.out.println(i+" "+l.macro_name+" "+l.mdt_ind);
           i++;

 }
}

static void showMdt() throws Exception {

 int i=0;
 for(String l : mdt) {
           System.out.println(i+" "+l);
           i++;

 }

}
void Process_Def(String s)
{
```

```
String l=sc.nextLine();
String tk[]=l.split(" ");
mnt.add(new MNT(tk[0],mdtc)); //store macro name in MNT
mntc++;
//Process Arguments in ala
String arg[]=tk[1].split(",");

for(int i=0;i<arg.length;i++)
   ala.add(arg[i]);

mdt.add(l);
mdtc++;

while(!l.equalsIgnoreCase("MEND"))
{
        //add in MDT
        int ind,i=0;
        String OP_line=new String();
        l=sc.nextLine();
        if((ind=l.indexOf("&"))>0)
        {  //argument exists, replace by ala index
                String wrd[]=l.split(" ");

                String args[]=wrd[1].split(",");
                OP_line=wrd[0]+" "+args[0];
                while(i<args.length)
                {
                        if(args[i].startsWith("&"))
                        {
                                ind=ala.indexOf(args[i]);
                                OP_line=OP_line+",#"+ind;

                        }

                        i++;
                }
        }
        else
                OP_line=l; //no need of index substitution
        mdt.add(OP_line);
        mdtc++;
```

```
 }

}//end of process def method
  void init()
  {
  mnt=new LinkedList<MNT>();
  mdt=new ArrayList<String>();
  ala=new LinkedList<String>();
  mntc=0;
  mdtc=0;
 //  ala_mname_binding=new HashMap<String,Integer>();
  }
}
```

Conclusion:

Input assembly language program is processed by applying Pass-I algorithm of assembler and intermediate data structures, Symbol Table, Literal Table, MOT, POT, BT, etc. are generated.

Assignment No.: 02

Problem Statement:

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

Objectives:

1. To study the design and implementation of 2nd pass of two pass assembler.
2. To study the data structure used in Pass-2 of assembler implementation.

Theory:

1. Explain the design of Pass- II of assembler with the help of flowchart and example.

Theory

Explain the design of Pass - II of the assembler with the help of flowchart and example

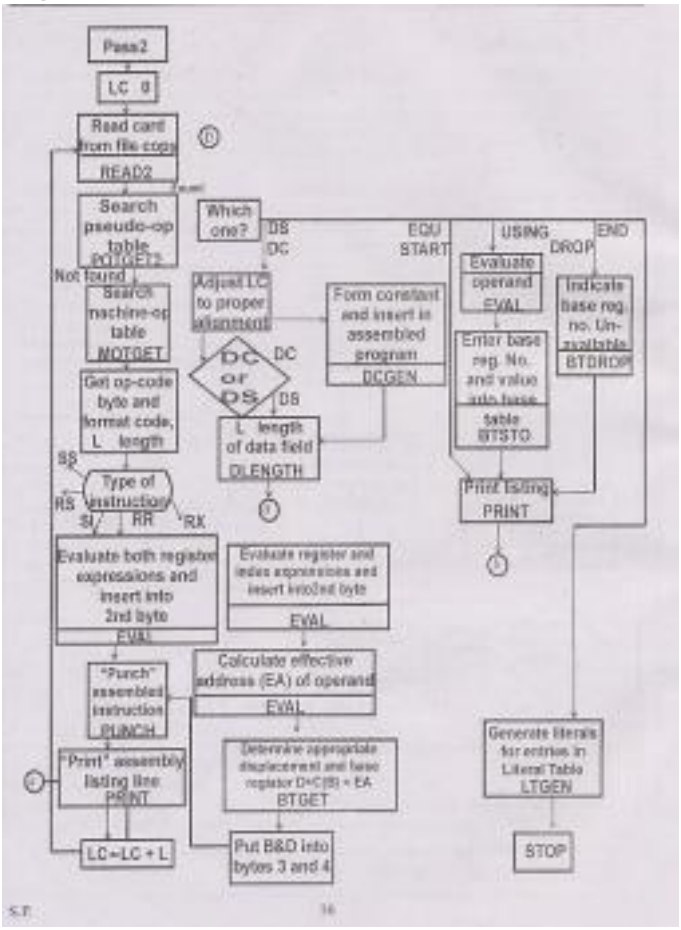
Two pass translation consists of pass I and pass II generally, LC processing performed in the first pass and symbols defined in the program entered into symbol table hence first pass performed analysis of the source program

So, two pass translation of assemblying; the program can handle forward refrence easily The second pass synthesizes the target form using the address information found in the symbol table

more over, the first pass constructs an intermediate representing of the source program and that will be used by the second pass

It, consists of two main components data structure & IC (intermidiate code)

Algorithm/Flowchart:



Design diagrams (if any):

1. Class Diagram
2. Use case Diagram
3. ER Diagram

Input:

Intermediate code of pass-1.

LC LABEL INSTR. OPERANDS

---

100 SAMPLE START 100 100 USING *, 15 100 L 1, FOUR 104 A 1, =F'3' 108 ST 1, RESULT 112 SR 1, 2 114 LTORG 124 L 2, FIVE 128 A 2, =F'5' 132 A 2, =F'7'
136 FIVE DC F'5' 140 FOUR DC F'4' 144 RESULT DS 1F
152 5

156 7
160 END

Machine Opcode Table (MOT)

Mnemonic	Hex / Binary Code	Length (Bytes)	Format
L	5A	4	RX
A	1B	4	RX
ST	50	4	RX
SR	18	2	RR

Pseudo Opcode Table (POT)

Pseudo op	Address / Name of Procedure to implement pseudo operation
START	PSTART
USING	PUSING
DC	PDC
DS	PDS
LTORG	PLTORG
END	PEND

Symbol Table (ST)

Sr. No	Symbol name	Address	Value	Length	Relocation
1	SAMPLE	100	--	160	R
2	FIVE	136	5	4	R
3	FOUR	140	4	4	R
4	RESULT	144	__	4	R

Literal Table (LT)

Sr. No Literal Address Length
1 3 120 4
2 5 152 4
3 7 156 4

Output:

Base Table (BT)

Register no Availability Value/ Contents 1
N --
: : :
: : :
: : :
15 Y 100

Object Code

LC OPCODE OPERAND
---------- 100
5A 1,40(0,15)
104 1B 1,20(0,15)
108 50 1,44(0,15)
112 18 1,2
124 5A 2,36(0,15)
128 1B 2,52(0,15)
132 1B 2,56(0,15)

Instructions :

1.
2.
3.

Test Cases:

1. Check syntax of instruction (Correct and wrong)
2. Symbol not found
3. Wrong instruction
4. Duplicate symbol declaration
5. Test the output of program by changing value of START & USING pseudo opcode.

Software Requirement:

1. Fedora
2. Eclipse
3. JDK

Hardware Requirement:

Frequently Asked Questions:

1. What is two pass assembler?
2. What is the significance of symbol table?
3. Explain the assembler directives EQU, ORIGIN.
4. Explain the assembler directives START, END, LTORG.
5. What is the use of POOLTAB and LITTAB?
6. How literals are handled in pass I?
7. What are the tasks done in Pass I?
8. How error handling is done in pass I?
9. Which intermediate data structures are designed and implemented in PassI? 10. What is the format of a machine code generated in PassII?
11. What is forward reference? How it is resolved by assembler?
12. How error handling is done in pass II?
13. What is the difference between IS, DL and AD?

Q.2] What is two pass assembler?
The assembler goes through the program one line at a time and generates machine code for that instruction. Then the assembler proceeds to the next instruction. In this ways, the entire machine code program is created.

Q.3] What is the significance of symbol table?
Symbol table is an important data structure created and maintained by the compiler in order to keep track of semantics of variable i.e it stores information about scope and binding information about names, information about instruction information about names en classes objects etc.

Q.3] Explain the assembler directives EQU, ORIGIN
EQU - The EQU directive only tells the assembler to substitute values for a symbol or label and doesn't involve any type of ROM or RAM, EQU directives are typically placed at the begining of an assembly program
ORIGIN - The directives help the program in getting compiled and hence won't be there in the object code, this basically is used to replace the variable with a constant value.

Q.4] Explain the assembler directives START, END, LTORG

START - Define the start of the first control section in a program

END - END of the assembler module or control section

LTORG - Begin the literal pool

Q.5] What is the use of POOLTAB and LITTAB?

POOLTAB - Addresses of different literal pools is maintained using the auxiliary table POOLTAB, At any stage, the current literal pool is the last pool in LITTAB

LITTAB - LTORG directive place the all constants at consecutive memory locations If we are not using LTORG then all literals are placed after END in memory

Q.6] How literals are handling in Pass-I

Literals are replace table terms because the address of the literal, rather than the literal-generated constant itself is assembled in the statement that reference a literal. the assembler generate the literals, collects them and places them in a specific area of storage as explained under literal pool.

Q.f] What are the tasks done in Pass I
Define Symbols and literals and remember them in Symbol table and literal table respectively
keep track of location counter
process pseudo-operation

Q.g] How error handling is done in Pass I
There are four ways to handle errors in which you can propagate the error from function to the code that call that function, handle the error using do-catch statement, handle the error will not occurs.

Q.h] Which intermediate data structure are designed and implemented in pass I
Assembler implementation is based on two major data structures - operation table (OPTAB) and symbol table (SYMTAB) for each mnemonic N and OPTAB contains

Q.i] What is the format of a machine code created in pass II ?
Converting symbolic machine-opcodes into their respective bit configuration machine understandable form. It stores all machine-opcodes in MOT table (Opcode table) with symbolic code, their length and their bit configuration

Q.11] What is forward reference ? How it is resolved by assembler ?

A forward reference occurs when a label is used as an operand, for example as a branch target, earlier in the code than the definition of the label, the assembler cannot known the address of the forward reference label until it reads the definition of the label.

Q.12] How error handling is does in Pass II

Ext without errors, one or more non-fetal errors found during pass 2, Assembly quite by user, the program terminated because of fetal error, fatal error wrong python version is used

Q.13] What is the diffirence between IS, DL and Also AD ?

Distribution groups are used for sending email notification to a group of people security groups are used for grouping access to resources such as share point it is mail enabled security groups are used for granting access to resource such as share point and emailing notifications to those users

Q.14) What are the tasks done in pass II & Pass-2 - generates object code by converting symbolic op-code into respective numeric opcode, generate data for literals and looks for values of symbols

Code –

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

class MNT {
String name;
int index;

MNT(String s, int i) {
	name = s;
	index = i;
}
}
class ALA
{
	String formal;
	String actual;
	ALA(String f,String a){
		formal=f;
		actual=a;
	}

}

public class Mpass2 {
	static List<MNT> mnt;
	static List<String> mdt;
	static int mntc;
	static int mdtc;
	static int mdtp;
	static List<ALA> ala;
	static Scanner sc;
	static PrintWriter op;
	public static void main(String args[]) throws Exception {
		op=new PrintWriter(new FileOutputStream("pass2_op.txt"));
		Mpass2 p2=new Mpass2();
		p2.initializeTables();
		System.out.println("ALA:");
		p2.showAla(1);
		System.out.println("\nMNT:");
		p2.showMnt();
		System.out.println("\nMDT:");
		p2.showMdt();
		System.out.println("\n===== PASS 2 =====\n");
```

```
            p2.pass2();
        }
        void pass2() throws Exception {
            File f=new File("op.txt");
            sc=new Scanner(f);

            while(sc.hasNextLine())
            { int flag=0;
                String line=sc.nextLine();

                for(MNT l : mnt){
                    if(line.contains(l.name))
                    {   //macro call found process macro call

                        mdtp=l.index;
                        System.out.println(line);
                        process_call(mdtp,line); //call expansion
                        flag=1;
                        break;
                    }
                }
                if(flag==0)
                {
                    op.println(line);
                    op.flush();
                }

            }
        }
        void process_call(int mdtp,String s) throws Exception
        {
            String mname[]=s.split(" ");
            String actual_args[]=mname[1].split(",");
            String mdt_words[]=mdt.get(mdtp).split(" "); //read line from MDT
and split
            String args[]=mdt_words[1].split(",");

            for(int i=0;i<args.length;i++)
            {
                for(int j=0;j<ala.size();j++) {
                    ALA l=ala.get(j);
                  if(l.formal.equals(args[i]))
                  {
                        //formal argument found, so set actual one
```

```
                        ala.set(j,new ALA(l.formal,actual_args[i]));
                    }
                }
            }
            //Show ALA After setting Actual arguments
            System.out.println("ALA After setting Actual arguments");
            showAla(2);
            mdtp++;
            String final1="";
            while(!mdt.get(mdtp).equals("MEND"))
            {
                String op_line=mdt.get(mdtp);
                mdtp++;
                if(op_line.contains("#"))
                { int ind=op_line.indexOf("#");
                  final1=op_line.substring(0,ind);

ind=Integer.parseInt(op_line.substring(ind+1,op_line.length()));
                  ALA l=ala.get(ind);
                  final1=final1+l.actual;
                }
                else
                    final1=op_line;
              op.println(final1);
              op.flush();
            }
      }
      void showAla(int pass) throws Exception {
            int i=0;
            for(ALA l : ala) {
                System.out.println(i+" "+l.formal+" "+l.actual);
                i++;
            }
      }

      void showMnt() throws Exception {
            int i=0;
            for(MNT l : mnt) {
                System.out.println(i+" "+l.name+" "+l.index);
                i++;

            }
      }
```

```
	void showMdt() throws Exception {
		int i=0;
		for(String l : mdt) {
			System.out.println(i+" "+l);
			i++;

		}
	}

	void initializeTables() throws Exception{
		mnt = new LinkedList<MNT>();
		mdt = new ArrayList<String>();
		ala = new LinkedList<ALA>();

		String mname=new String();
		//Load MNT
		String s=new String();
		File f=new File("MNT.txt");
		Scanner input = new Scanner(f);
		while(input.hasNextLine()) {
			s=input.nextLine();
			String words[]=s.split(" ");
			mnt.add(new MNT(words[0],Integer.parseInt(words[1])));

		}
		//load MDT
	    f=new File("MDT.txt");
		 input = new Scanner(f);
		while(input.hasNextLine()) {
			s=input.nextLine();
			mdt.add(s);
		}
		//Load ALA pass1
		f=new File("ala.txt");
		 input = new Scanner(f);
		while(input.hasNextLine()) {
			s=input.nextLine();
			String words[]=s.split(" ");
			for(int i=0;i<words.length;i++)
			  ala.add(new ALA(words[i],"-"));
		}

	}
}//end of class
```

Conclusion:
The intermediate data structures generated in Pass-I of assembler are given as input to the PassII of assembler, processed by applying Pass-II algorithm of assembler and machine code is generated

Assignment No.: 03

Problem Statement: Design suitable data structures and implement Pass-I of a two pass macro processor using OOP features in Java/C++. The output of Pass-I (MNT, MDT, ALA & Intermediate code file without any macro definitions) should be input for Pass-II.

Objectives:

1. To identify and design different data structure used in macro-processor implementation
2. To apply knowledge in implementation of two pass microprocessor.

Theory:

1. What is macro processor?
2. Differentiate Macro and Function?
3. Explain the design of Pass- I of macro-processor with the help of flowchart? 4. Explain the design of Data structure used in Pass-I?
5. Explain the data structures used in Pass-I?

Theory

1] What is macro processor
macro preprocessor takes a source program containing macro definitions and macro calls and translates into an assembly languages program without any macro definition or calls this program can now be handed over to a conventional assembler to obtain the target language

2] Differentiate macro and function ?

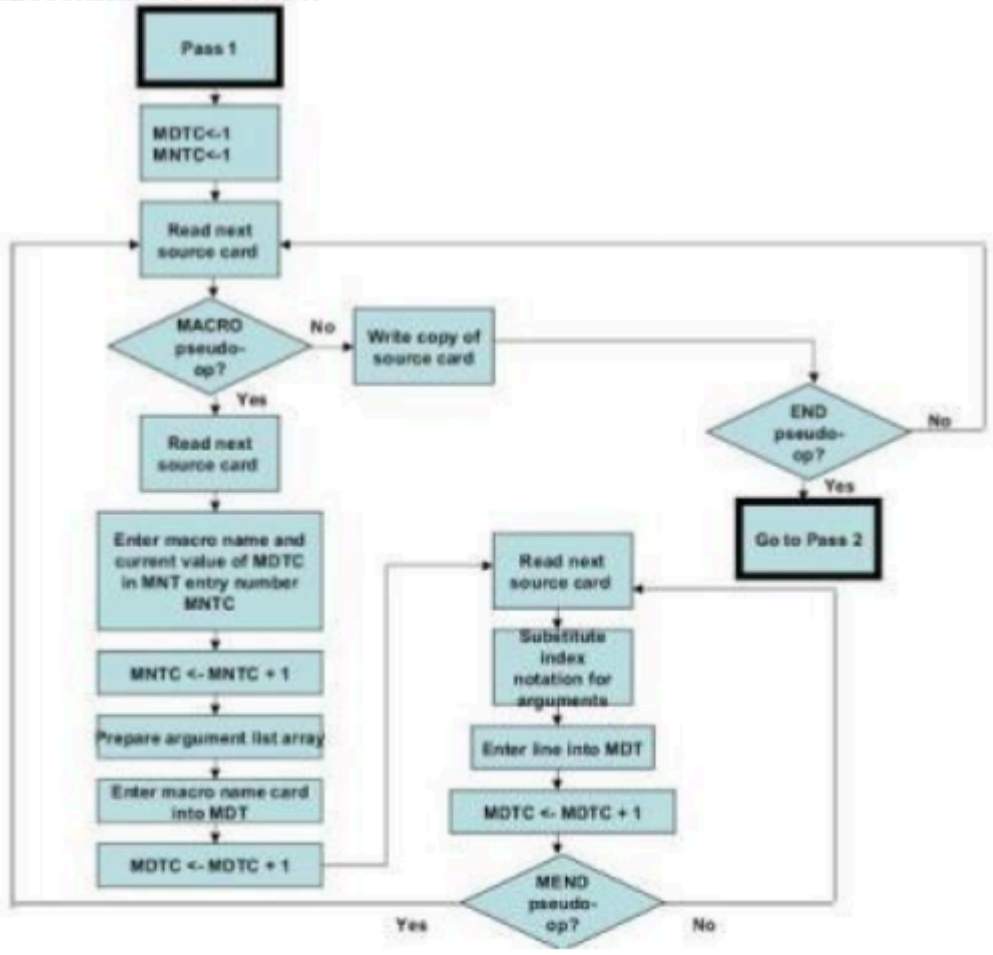
MACRO	Function
MACRO is preprocessed	Function is compiled
No Type checking is done in macro	Type checking is done in Function

3] Explain the design of pass - I of macro-processor with the help of flowchart
A one pass macro processor is another design option available for macro processing. The restriction in working with one pass macro processors is that they strictly require require the definition

Q.4] Explain the design of data structure used in pass-I ?
INPUT source program
A location counter (LC) used to keep track of each instruction location
A table, the machine-operation table (MOT), th

Q.5] Explain the data structure used in pass-I
Input source program
A location counter (LC) : used to keep each instruction location
A table, the machine operation table indicates the symbolic mnemonic for action and its length (two, four, or six)

Algorithm/Flowchart:



Design diagrams (if any):

1. Class diagram 2. Sequence diagram
3.

Input:

Small assembly language program with macros written in file input.asm.

```
        MACRO
&lab ADDS &arg1,&arg2
&lab L 1, &arg1
 A 1, &arg2
 MEND
PROG START 0
 BALR 15,0
 USING *,15
LAB ADDS DATA1, DATA2
 ST 4,1
DATA1 DC F'3'
DATA2 DC F'4'
 END
```

Output:

Assembly language program without macro definition but with macro call.
Note: Follow the following templates during implementation **Macro Name Table (MNT) :**

Macro Definition Table (MDT) :

Argument List Array (ALA) :

Instructions :

1.
2.
3.

Test Cases:

1. Check macro end not found.
2. Duplicate macro name found.
3. Check program output by changing macro name and parameter list.
4. Handle label in macro definition.
5. Handle multiple macro definitions and calls

Software Requirement:

1. Fedora
2. Eclipse
3. JDK

Hardware Requirement: N/A

Frequently Asked Questions:

1. Define macro?
2. Define purpose of pass-1 of two pass macro processor
3. List out types of macro arguments
4. What is the use of MDT-index field in MNT?
5. What we store in ALA?

3.2

Q.1] Define macro ?
A macro is an automated input sequence that imitate keystrokes. A macro is typically used to replace a respective series of Keyboard and mouse actions and used often in spread sheets and word processing application like MS Excel and MS Word

Q.2] Define purpose of pass-1 of the two pass macro processor
A one-pass macro processor that alternate between macro definition and macro expansion in a recursive way is able to handle recursive macro definition

Q.3] List out types of macro arguments
Two types of arguments
keyword and positional, keyword arguments are assigned names in the macro definition in the macro call, they are identified by name positional arguments are defined after the keyword

Q.4] What is the use of MDT - Index field in MNT ?
The macro definitions are stored in MDT (macro definition table and the names of the macros are stored in MNT (macro name table). The macro definition table (MDT), used to store, the body of the

macro definitions

Q.5] What we store in ALA ?
Argument list array used to substituted for value during argument before storing

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("=");
```

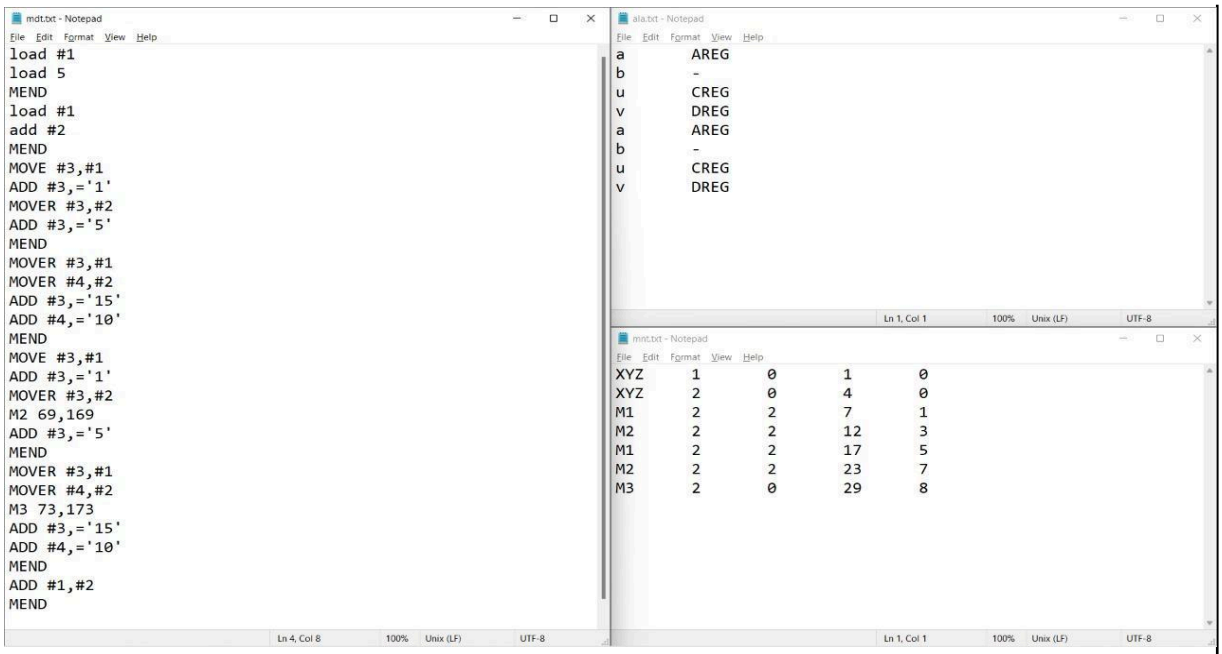
```
pntab.put(keywordParam[0].substring(1,keywordParam[0].length()),paramNo++);
                                        if(keywordParam.length==2)
f4.write(keywordParam[0].substring(1,keywordParam[0].length())+"\t"+keywordParam[1]+"\n");
                                        else
f4.write(keywordParam[0].substring(1,keywordParam[0].length())+"\t"+"-"+"\n");

                                }
                                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 –



Conclusion: We have successfully completed implementation of Pass-I of macro processor.

Assignment No.: 04

Problem Statement: Design suitable data structures and implement Pass-II of a two pass macro processor using OOP features in Java/C++. The output of Pass-I (MNT, MDT, ALA & Intermediate code file without any macro definitions) should be input for Pass-II.

Objectives:

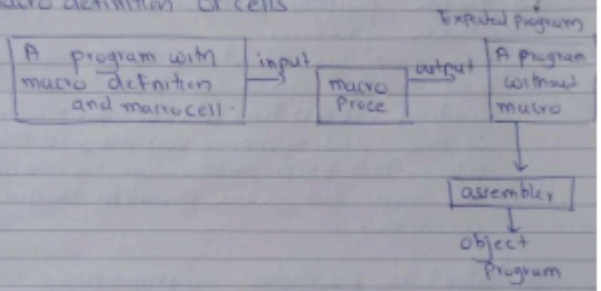
1. To identify and design different data structure used in macro-processor implementation
2. To apply knowledge in implementation of pass-2 of two pass microprocessor.

Theory:

1. Explain design steps of two pass microprocessor, types of statements, data structures required and flowcharts.

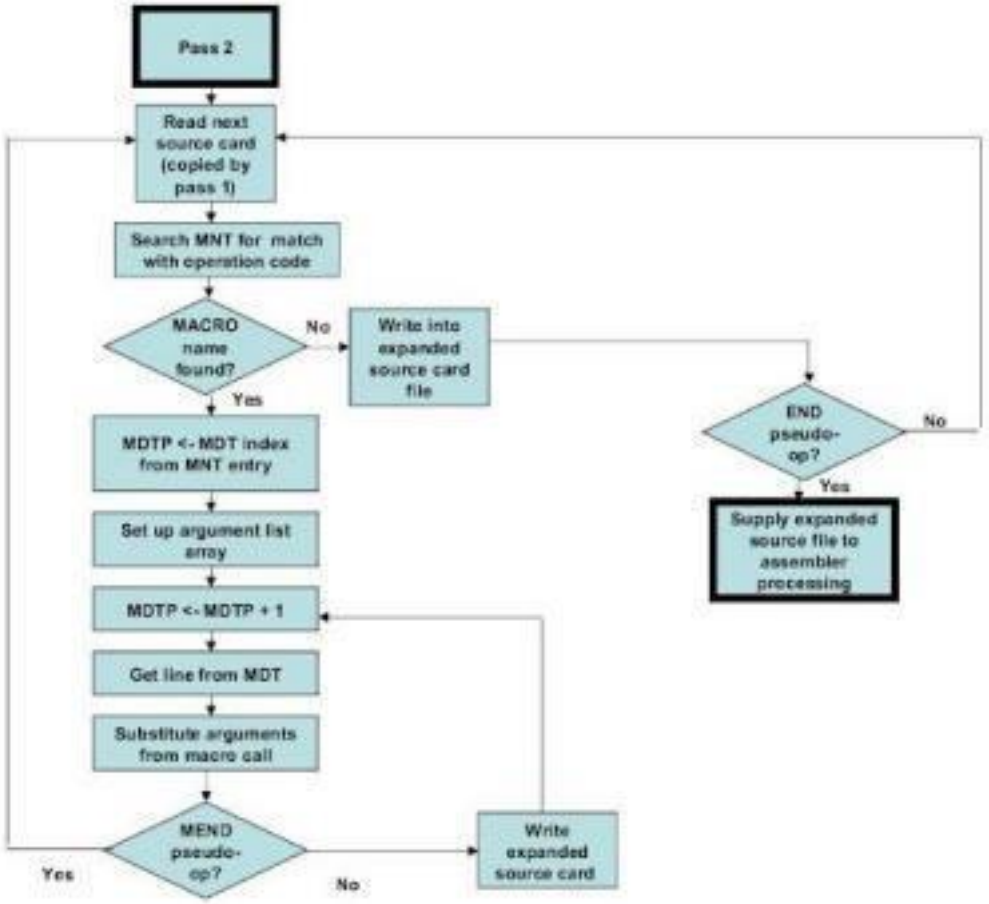
Theory

Q1] Explain design steps of two pass microprocessor, types of statements, data structures required and flowcharts macro processors takes a source program containing macro definitions and macro calls, It converts it into an assembly languages program without macro definition or calls



The macro names are entered into NAMTAB, which serves as an index to DEFTAB, for each macro instruction defined, NAMTAB contains pointers to be begining and end of the definition in DEFTAB

Algorithm/Flowchart:



Design diagrams (if any):

1. Class diagram 2.
Sequence diagram
3.

Input: Output of pass-1 (Intermediate File) given as a input to pass-2.


```
PROG START 0
BALR 15,0
USING *,15
LAB ADDS DATA1, DATA2
ST 4,1
DATA1 DC F'3'
DATA2 DC F'4'
END
```

Output:

Assembly language program without macro definition and macro call.

```
PROG START 0
BALR 15,0
USING *,15
LAB L 1, DATA1
A 1, DATA2
ST 4,1
DATA1 DC F'3'
DATA2 DC F'4'
END
```

Instructions :

1.

2.

3.

Test Cases:

1. Check macro definition not found.
2. Check program output by changing parameter list in macro call.

Software Requirement:

1. Fedora
2. Eclipse
3. JDK

Hardware Requirement: N/A

Frequently Asked Questions:

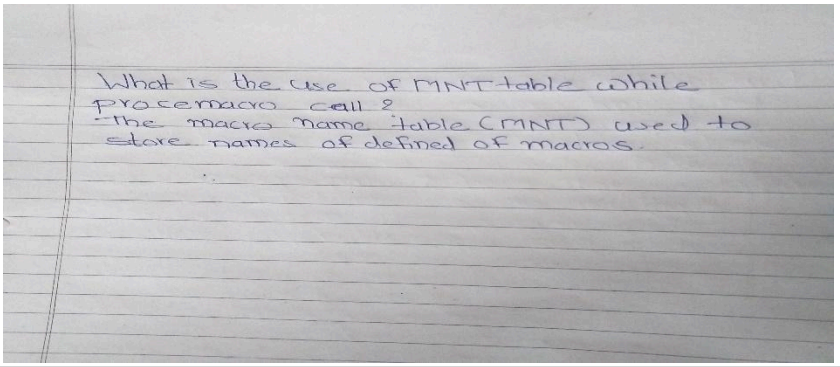
1. What is macro expansion?
2. Define purpose of pass-2 of two pass macro processor
3. What is positional arguments?
4. What is the use of MDT-index field in MNT?
5. What is the use of MNT table while processing macro call?

1] What is macro expansion?
A macro consists of name, set of formal parameters and a body of code. The use of macro a name with set of actual parameters is replaced by some code generated by its body. This is called macro expansion.

2] Define purpose of pass-2 of two pass macro processor.
For each line it checks if op-code of that line matches any of the mnt entry. Reading MEND line in MDT terminates expansion of macro and scanning continues in the input file, when END pseudo-op is encountered the expanded source program is given to the assembler.

3] What is positional arguments?
Positional arguments are arguments that need to be included in the proper position or order. The first positional argument always needs to be listed first when the function is called. The second positional argument needs to be listed second and the third position argument listed third, etc.

Q 9] What is the use of MDT-index field in MNT?
The macro definition are stored in MDT (macro definition table) and the names of the macros are stored in MNT (macro name table).



Code –

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

class MNT {
String name;
int index;

MNT(String s, int i) {
   name = s;
   index = i;
}
}
class ALA
{
   String formal;
   String actual;
   ALA(String f,String a){
      formal=f;
      actual=a;
   }

}

public class Mpass2 {
   static List<MNT> mnt;
   static List<String> mdt;
   static int mntc;
   static int mdtc;
   static int mdtp;
   static List<ALA> ala;
   static Scanner sc;
   static PrintWriter op;
```

```
public static void main(String args[]) throws Exception {
   op=new PrintWriter(new FileOutputStream("pass2_op.txt"));
   Mpass2 p2=new Mpass2();
   p2.initializeTables();
   System.out.println("ALA:");
   p2.showAla(1);
   System.out.println("\nMNT:");
   p2.showMnt();
   System.out.println("\nMDT:");
   p2.showMdt();
   System.out.println("\n===== PASS 2 =====\n");
   p2.pass2();
}
void pass2() throws Exception {
   File f=new File("op.txt");
   sc=new Scanner(f);

   while(sc.hasNextLine())
   { int flag=0;
      String line=sc.nextLine();

      for(MNT l : mnt){
         if(line.contains(l.name))
         {   //macro call found process macro call

            mdtp=l.index;
            System.out.println(line);
            process_call(mdtp,line); //call expansion
            flag=1;
            break;
         }
      }
      if(flag==0)
      {
         op.println(line);
         op.flush();
      }

   }
}
void process_call(int mdtp,String s) throws Exception
{
   String mname[]=s.split(" ");
   String actual_args[]=mname[1].split(",");
```

```
        String mdt_words[]=mdt.get(mdtp).split(" "); //read line from MDT
and split
        String args[]=mdt_words[1].split(",");

        for(int i=0;i<args.length;i++)
        {
           for(int j=0;j<ala.size();j++) {
              ALA l=ala.get(j);
             if(l.formal.equals(args[i]))
             {
                //formal argument found, so set actual one
                ala.set(j,new ALA(l.formal,actual_args[i]));
             }
           }
        }
        //Show ALA After setting Actual arguments
        System.out.println("ALA After setting Actual arguments");
        showAla(2);
        mdtp++;
        String final1="";
        while(!mdt.get(mdtp).equals("MEND"))
        {
           String op_line=mdt.get(mdtp);
           mdtp++;
           if(op_line.contains("#"))
           { int ind=op_line.indexOf("#");
             final1=op_line.substring(0,ind);
             ind=Integer.parseInt(op_line.substring(ind+1,op_line.length()));
             ALA l=ala.get(ind);
             final1=final1+l.actual;
           }
           else
              final1=op_line;
          op.println(final1);
          op.flush();
        }
     }
     void showAla(int pass) throws Exception {
        int i=0;
        for(ALA l : ala) {
           System.out.println(i+" "+l.formal+" "+l.actual);
           i++;
        }
     }
```

```
void showMnt() throws Exception {
   int i=0;
   for(MNT l : mnt) {
      System.out.println(i+" "+l.name+" "+l.index);
      i++;

   }
}

void showMdt() throws Exception {
   int i=0;
   for(String l : mdt) {
      System.out.println(i+" "+l);
      i++;

   }
}

void initializeTables() throws Exception{
   mnt = new LinkedList<MNT>();
   mdt = new ArrayList<String>();
   ala = new LinkedList<ALA>();

   String mname=new String();
   //Load MNT
   String s=new String();
   File f=new File("MNT.txt");
   Scanner input = new Scanner(f);
   while(input.hasNextLine()) {
      s=input.nextLine();
      String words[]=s.split(" ");
      mnt.add(new MNT(words[0],Integer.parseInt(words[1])));
   }
   //load MDT
    f=new File("MDT.txt");
    input = new Scanner(f);
   while(input.hasNextLine()) {
      s=input.nextLine();
      mdt.add(s);
   }
   //Load ALA pass1
   f=new File("ala.txt");
    input = new Scanner(f);
```

```
        while(input.hasNextLine()) {
           s=input.nextLine();
           String words[]=s.split(" ");
           for(int i=0;i<words.length;i++)
             ala.add(new ALA(words[i],"-"));
        }

     }
}//end of class
```

Conclusion: We have successfully completed implementation of Pass-II of macro processor.

Assignment No.: 05

Problem Statement:

Write a program to create a Dynamic Link Library for any mathematical operations (arithmetic, trigonometric and string operation) and write an application program to test it. (Java Native Interface/Use VB/VC++)

Objectives:

1. To study and understand concept of DLL
2. To understand JNI
3. To implement DLL using JNI

Theory:

1. What is DLL? Significance of DLL. Advantages/ Disadvantages of DLL
2. What is Native Interface? Reasons to use JNI.
3. What is shared object?

E-1

Theory

What is DLL? Significance of DLL advantages / Disadvantages of DLL

A DLL is a library that contains code and data that can be used by more than one program at the same time. A DLL (.dll) file contains a library of functions and other information that can be accessed by a windows program.

Advantages - The undeniable advantage of DLL is to save memory for applications that share basic features. Share less memory. A DLL file is inserted only once in memory regardless of which programs it accesses its function.

Disadvantages - An application that uses a dynamic-link library is not self-contained and it relies on a DLL module to exists and if you use dynamic linking at load time, the program starts to discover that the DLL does not exists and the system terminates the program and gives an error message.

Q.2] What is Native Interface? Reasons to use JNI

Native method Interface (JNI) is a part of JDK that connects Java code with native application and libraries that are written in other programming languages. programmers use the JNI to write Java

native method to handle those situation when an application cannot be written entirely in java

Q.3] What is shared object
A shared object is an individual unit generated from one or more relocatable shared object can be bound ~~which~~ with dynamically to from a runnable process. As the implies shared object can be shared one application.

Algorithm/Flowchart:

1. **Write a Java Class that uses C Codes - TestJNI.java** public

class TestJNI {

static {

System.loadLibrary(“cal"); // Load native library at runtime

// cal.dll (Windows) or libcal.so (Unix)

}

// Declare a native method add() that receives nothing and returns void private native

int add (int n1,int n2); // Test Driver

public static void main(String[] args) {

// invoke the native method

System.out.println(“Addition is=”+new TestJNI().add(10,20);

} }

Compile Java code: javac

TestJNI.java

2. **Create the C/C++ Header file - TestJNI.h** javah -jniTestJNI
3. **C Implementation - TestJNI.c**

#include <jni.h>
#include <stdio.h>
#include “TestJNI.h"
// Implementation of native method add() of TestJNI class

```
JNIEXPORT jint JNICALL Java_TestJNI_add(JNIEnv *env, jobjectthisObj,jint n1,jint
 n2) {
jint res;
res=n1+n2;
return res;
 }
```

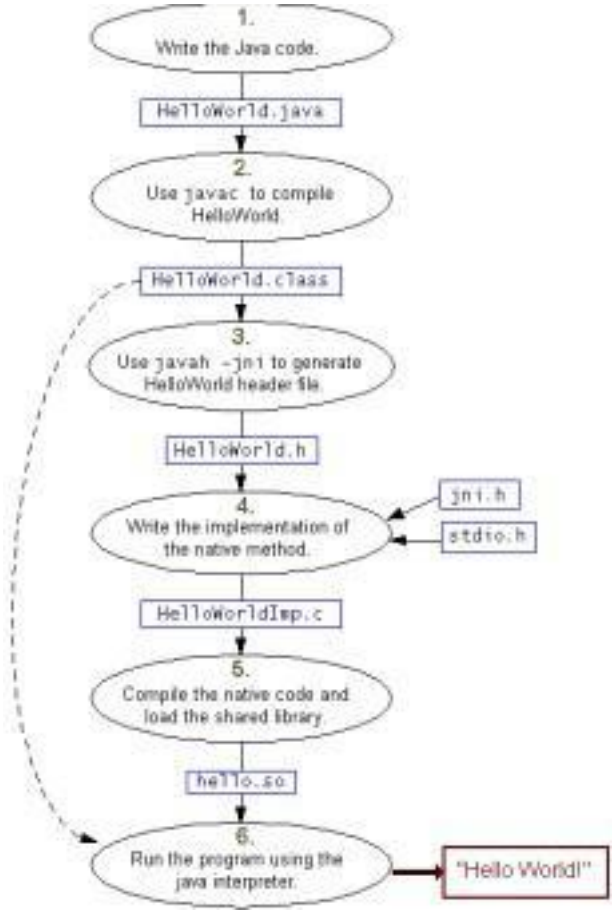
Compile c-program:

```
$gcc -I /usr/local/jdk1.8.0_91/include /usr/local/jdk1.8.0_91/include/linux -o
libcal.so –shared TestJNI.c
```

4. Run java program

```
$java -Djava.library.path=. TestJNI Addition
is=30
```

5. Repeat step 1-4 for all mathematical operations mentioned in problem statement. Flowchart:



Design diagrams (if any):

1. Use Case Diagram
2. Sequence Diagram

Input:

1. n1=20
2. n2=10

Output: 1.
Addition=30

Instructions :
1. This assignment can be implemented using VB application and C++ DLL using visual studio on windows

Test Cases:
1. Divide by zero
2. Missing arguments

Software Requirement:
1. Fedora
2. Jdk
3. Eclipse/ equivalent IDE

Hardware Requirement:

Frequently Asked Questions:
1. Difference between static link library and dynamic link library 2. What is shared object?
3. Advantages/Disadvantages of using JNI

Q.1] Difference between static link library and dynamic link library

static link library	dynamic link library
static linking is performed by progress called linkers as the last step in compiling a program, linkers are also called link editors	dynamic linking is performed at run time by the operating system
statically linked program takes constant load time every time it is loaded into the memory for execution	In dynamic linking load time might be reduced if the shared library code is already present in memory

Q.2] What is shared object ?

A shared object is an indivisible unit and that is generated from one or more relocatable objects shared objects can form a runnable process.

Q.3] Advantages/ Disadvantages of using JNI

Advantages - Use the existing library that was previously written in various languages Use of windows API function ; Increasing the speed of execution Invoke API function is server product that is developed in C or C++ from java client

Disadvantage : Write once run anywhere is not possible Run time errors debugging is difficult in native code an applet can not a native method security risk is potential

Code –

ClassLibrary2.vb

```
Public Class MyFunctions
   Public Function AddMyValues(ByVal Value1 As Double, ByVal Value2 As
Double)
       Dim Result As Double
       Result = Value1 + Value2
       Return Result
   End Function
End Class
```

Form1.vb

```
Imports ClassLibrary2

Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
       Dim Obj As New ClassLibrary2.MyFunctions
       TextBox3.Text = Obj.AddMyValues(CDbl(TextBox1.Text),
      CDbl(TextBox2.Text)).ToString
   End Sub
End Class
```

Conclusion:

Successfully implemented DLL and tested it with java application