Sinhgad College of Engineering

Department of Computer Engineering

Name : Prashant Kumar

Roll : 305139 **PRN** : 71813716H

Class : TE **DIV** : 2 **BATCH** : B

Name of Laboratory : System Programming And Operating System Lab

List of Assignments

Sr.No.	Title of Assignment	Remark	Signature
1	Implementation of Pass-1 of two pass assembler.		
2	Implementation of Pass-2 of two pass assembler.		
3	Implementation of pass-1 of Two Pass Macro Processor		
4	Implementation of pass-2 of Two Pass Macro Processor		
5	Implementation of Dynamic Link Library		
6	Lexical analyzer for subset of 'Java' program tokenization using LEX.		
7	Implementation of lexical analysis phase of compiler to count no. of words, lines and characters of given input file.		
8	Implementation of syntax analysis phase of compiler to validate type and syntax of variable declaration in Java.		
9	Implementation of syntax analysis phase of compiler to recognize simple and compound sentences.		
10	Banker’s Algorithm		
11	Simulation of paging		

Assignment No : 1

Title of Program : Implementation of Pass-1 of two pass assembler.

Objective :

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.

Code of Program :

AssmeblerPass1.java

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;

//Desgining MOT
class Tuple {

   String mnemonic, bin_opcode, type;
   int length;

   Tuple() {
   }
```

```
    Tuple(String s1, String s2, String s3, String s4) {
        mnemonic = s1;
        bin_opcode = s2;
        length = Integer.parseInt(s3);
        type = s4;
    }
}
//Desgining ST

class SymTuple {

    String symbol, ra;
    int value, length;

    SymTuple(String s1, int i1, int i2, String s2) {
        symbol = s1;
        value = i1;
        length = i2;
        ra = s2;
    }
}
//Designing Literal

class LitTuple {

    String literal, ra;
    int value, length;

    LitTuple() {
    }

    LitTuple(String s1, int i1, int i2, String s2) {
        literal = s1;
        value = i1;
        length = i2;
        ra = s2;
    }
}
public class AssemblerPass1 {

    static int lc;
    static List<Tuple> mot; //required to read MOT
    static List<String> pot; //required to read POT
    static List<SymTuple> symtable; //generate symbol table
    static List<LitTuple> littable; //generate literal table
    static List<Integer> lclist;
    static Map<Integer, Integer> basetable; //base table
    static PrintWriter out_pass2; //output of pass 2
    static PrintWriter out_pass1; //output of pass 1
    static int line_no;

    public static void main(String[] args) throws Exception {
        initializeTables(); //initialize everything needed
        System.out.println("====== PASS 1 ======\n");
        pass1(); //Run Pass 1

        //exporting lclist to file, so that it can be used in pass2
        PrintWriter lclistWriter = new PrintWriter(new FileWriter("/home/student/workspace/SPOSL/src/
lclist.txt"), true); //generate ST
        for (int i = 0; i < lclist.size(); i++) {
            lclistWriter.println(lclist.get(i));
        }
        lclistWriter.close();
    }

    static void initializeTables() throws Exception {
        symtable = new LinkedList<>();
        littable = new LinkedList<>();
        lclist = new ArrayList<>();
        basetable = new HashMap<>();
        mot = new LinkedList<>();
        pot = new LinkedList<>();
        String s;
        BufferedReader br;
        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/
SPOSL/src/mot.txt")));//reading MOT
        while ((s = br.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens
            mot.add(new Tuple(st.nextToken(), st.nextToken(), st.nextToken(), st.nextToken())); //adding token
into list
        }
        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/
SPOSL/src/pot.txt")));//reading POT
        while ((s = br.readLine()) != null) {
            pot.add(s); //adding token into POT list
        }
        Collections.sort(pot); //sorting all the POT as per their index
    }
    //Pass 1 Srarts here

    static void pass1() throws Exception {
        //Read Input file
        BufferedReader input = new BufferedReader(new InputStreamReader(new FileInputStream("/home/
student/workspace/SPOSL/src/input.txt")));
        out_pass1 = new PrintWriter(new FileWriter("/home/student/workspace/SPOSL/src/
output_pass1.txt"), true); //writing to Output file pass1
        PrintWriter out_symtable = new PrintWriter(new FileWriter("/home/student/workspace/SPOSL/src/
out_symtable.txt"), true); //generate ST
        PrintWriter out_littable = new PrintWriter(new FileWriter("/home/student/workspace/SPOSL/src/
out_littable.txt"), true); //generate LT
        String s;
        while ((s = input.readLine()) != null) { //till end of file is reached
            StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens
            String s_arr[] = new String[st.countTokens()]; //initialized s_arr
            for (int i = 0; i < s_arr.length; i++) {
                s_arr[i] = st.nextToken(); //get all tokens into s_arr
            }
            if (searchPot1(s_arr) == false) { //if the token is not available in POT
                searchMot1(s_arr); //search in MOT
                out_pass1.println(s); //write to file pass1
            }
            lclist.add(lc); //add lc into lc list
        }
        int j; //to be used
        String output = new String(); //to be used to print on console
        System.out.println("Symbol Table:");
        System.out.println("Symbol    Value  Length   R/A");
        for (SymTuple i : symtable) { //traverse all symbols from symbol table
            output = i.symbol; //store in output
            for (j = i.symbol.length(); j < 10; j++) { //show symbols
                output += " ";
            }
            output += i.value;
            for (j = new Integer(i.value).toString().length(); j < 7; j++) { //show values
                output += " ";
            }
            output += i.length + "        " + i.ra; //instruction length and relative or absolute
            System.out.println(output);
            out_symtable.println(output);
        }
        System.out.println("\nLiteral Table:"); //printing literal table
        System.out.println("Literal   Value  Length   R/A");
        for (LitTuple i : littable) { //traverse the literal tuple to print
            output = i.literal;
            for (j = i.literal.length(); j < 10; j++) {
                output += " ";
            }
            output += i.value;
            for (j = new Integer(i.value).toString().length(); j < 7; j++) {
                output += " ";
            }
            output += i.length + "        " + i.ra;
```

```
        System.out.println(output);
        out_littable.println(output);
    }
}

static boolean searchPot1(String[] s) {
    int i = 0; //to be used
    int l = 0; //to be used
    int potval = 0; //to be used

    if (s.length == 3) {
        i = 1;
    }
    s = tokenizeOperands(s); //tokenize all the operands given by s_arr

    if (s[i].equalsIgnoreCase("DS") || s[i].equalsIgnoreCase("DC")) {
        potval = 1; //if DC or DS
    }
    if (s[i].equalsIgnoreCase("EQU")) {
        potval = 2; //if EQU
    }
    if (s[i].equalsIgnoreCase("START")) {
        potval = 3; //if START
    }
    if (s[i].equalsIgnoreCase("LTORG")) {
        potval = 4; //if LTORG
    }
    if (s[i].equalsIgnoreCase("END")) {
        potval = 5; //if END
    }

    switch (potval) { //doing actions as per input from POT
        case 1:
            // DS or DC statement
            String x = s[i + 1]; //point to next token after DC or DS
    int index = x.indexOf("F"); //get the index position of F
    if (i == 1) {
        symtable.add(new SymTuple(s[0], lc, 4, "R"));
    }
    if (index != 0) {
        // Ends with F
        l = Integer.parseInt(x.substring(0, x.length() - 1));
        l *= 4;
    } else {
        // Starts with F
        for (int j = i + 1; j < s.length; j++) {
            l += 4;
        }
    }
    lc += l; //update LC
    return true;

case 2:
    // EQU statement
    if (!s[2].equals("*")) { //check if there is no pointer
        symtable.add(new SymTuple(s[0], Integer.parseInt(s[2]), 1, "A")); //add absolute address in ST
    } else {
        symtable.add(new SymTuple(s[0], lc, 1, "R")); //else add Relative address in ST
    }
    return true;

case 3:
    // START statement
    symtable.add(new SymTuple(s[0], Integer.parseInt(s[2]), 1, "R"));  //add program name in ST
    return true;

case 4:
    // LTORG statement
    ltorg(false); //call to LTORG method
    return true;
```

```
        case 5:
            // END statement
            ltorg(true); //call to LTORG method
            return true;
    }
    return false;
}

static void searchMot1(String[] s) {
    Tuple t = new Tuple(); //MOT object
    int i = 0;
    if (s.length == 3) { //check if 3 tokens
        i = 1; //keep i=1
    }
    s = tokenizeOperands(s); //again tokenize the operands
    for (int j = i + 1; j < s.length; j++) {
        if (s[j].startsWith("=")) { //check if literal
            littable.add(new LitTuple(s[j].substring(1, s[j].length()), -1, 4, "R")); //add into LT
        }
    }
    if ((i == 1) && (!s[0].equalsIgnoreCase("END"))) { //if 3 tokens in a line and not an END statement
        symtable.add(new SymTuple(s[0], lc, 4, "R")); //add entry to symbol table
    }
    for (Tuple x : mot) { //traverse all MOTs
        if (s[i].equals(x.mnemonic)) { //if mnemonic is found
            t = x; //store all mnemonics in t
            break;
        }
    }
    lc += t.length; //update location counter
}

static String[] tokenizeOperands(String[] s) {
    List<String> temp = new LinkedList<>(); //to be used
    for (int j = 0; j < s.length - 1; j++) { //adding all tokens into temp
        temp.add(s[j]);
    }
    StringTokenizer st = new StringTokenizer(s[s.length - 1], " ,", false); //convert line into tokens
    while (st.hasMoreTokens()) {
        temp.add(st.nextToken()); //adding all tokens
    }
    s = temp.toArray(new String[0]); //convert linked list to array list
    return s;
}

static void ltorg(boolean isEnd) { //adding literals in LT
    Iterator<LitTuple> itr = littable.iterator();  //Iterator used to store literal objects
    LitTuple lt = new LitTuple(); //created object
    boolean isBroken = false; //to be used
    while (itr.hasNext()) { //check the iterators
        lt = itr.next(); //check the literals
        if (lt.value == -1) {
            isBroken = true;
            break;
        }
    }
    if (!isBroken) { //if LTORG occurs
        return;
    }
    if (!isEnd) { //if not END
        while (lc % 8 != 0) {
            lc++; //reach up to END statement
        }
    }
    lt.value = lc;
    lc += 4;
    while (itr.hasNext()) {
        lt = itr.next(); //adding literals to lt
        lt.value = lc; //update LT Value
```

```
          lc += 4; //update location counter
       }
    }
}
```

INPUT :

Input.txt

```
PRGAM2     START   0
          USING   *,15
          LA      15,SETUP
          SR      TOTAL,TOTAL
AC          EQU     2
INDEX      EQU     3
TOTAL      EQU     4
DATABASE    EQU     13
SETUP      EQU     *
          USING   SETUP,15
          L       DATABASE,=A(DATA1)
          USING   DATAAREA,DATABASE
          SR      INDEX,INDEX
LOOP       L       AC,DATA1(INDEX)
          AR      TOTAL,AC
          A       AC,=F'5'
          ST      AC,SAVE(INDEX)
          A       INDEX,=F'4'
          C       INDEX,=F'8000'
          BNE     LOOP
          LR      1,TOTAL
          BR      14
          LTORG
SAVE       DS      3F
DATAAREA    EQU     *
DATA1      DC      F'25,26,27'
          END
```

mot.txt

```
LA     01h     4     RX
SR     02h     2     RR
L      03h     4     RX
AR     04h     2     RR
A      05h     4     RX
C      06h     4     RX
BNE    07h     4     RX
LR     08h     2     RR
ST     09h     4     RX
BR     15h     2     RR
```

pot.txt

```
START
END
LTORG
DC
DS
DROP
USING
EQU
```

OUTPUT :

outputpass1.txt

```
        USING   *,15
        LA      15,SETUP
        SR      TOTAL,TOTAL
        USING   SETUP,15
        L       DATABASE,=A(DATA1)
        USING   DATAAREA,DATABASE
        SR      INDEX,INDEX
LOOP    L       AC,DATA1(INDEX)
```

```
          AR     TOTAL,AC
          A      AC,=F'5'
          ST     AC,SAVE(INDEX)
          A      INDEX,=F'4'
          C      INDEX,=F'8000'
          BNE    LOOP
          LR     1,TOTAL
          BR     14
```

out_symtable.txt

```
PRGAM2    0     1       R
AC        2     1       A
INDEX     3     1       A
TOTAL     4     1       A
DATABASE  13    1       A
SETUP     6     1       R
LOOP      12    4       R
SAVE      64    4       R
DATAAREA  76    1       R
DATA1     76    4       R
```

out_littable.txt

```
A(DATA1)  48    4       R
F'5'      52    4       R
F'4'      56    4       R
F'8000'   60    4       R
```

lclist.txt

```
0
0
4
6
6
6
6
6
```

6

6

10

10

12

16

18

22

26

30

34

38

40

42

64

76

76

88

88

Assignment No : 2

Title : Implementation of Pass-2 of two pass assembler.

Objective :

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

Code of Program :

AssemblerPass2.java

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;

//Desgining MOT
class Tuple {
        String mnemonic, bin_opcode, type;
        int length;

        Tuple() {}

        Tuple(String s1, String s2, String s3, String s4) {
                mnemonic = s1;
                bin_opcode = s2;
                length = Integer.parseInt(s3);
                type = s4;
        }
}
```

```
//Desgining ST
class SymTuple {
        String symbol, ra;
        int value, length;

        SymTuple(String s1, int i1, int i2, String s2) {
                symbol = s1;
                value = i1;
                length = i2;
                ra = s2;
        }
}
//Designing Literal
class LitTuple {
        String literal, ra;
        int value, length;

        LitTuple() {}

        LitTuple(String s1, int i1, int i2, String s2) {
                literal = s1;
                value = i1;
                length = i2;
                ra = s2;
        }
}
public class AssemblerPass2 {

  static int lc;
  static List<Tuple> mot; //required to read MOT
  static List<String> pot; //required to read POT
  static List<SymTuple> symtable; //generate symbol table
  static List<LitTuple> littable; //generate literal table
  static List<Integer> lclist;
  static Map<Integer, Integer> basetable; //base table
```

```
    static PrintWriter out_pass2; //output of pass 2
    static PrintWriter out_pass1; //output of pass 1
    static int line_no;

    public static void main(String[] args) throws Exception {
        initializeTables(); //initialize everything needed

        //initialize evrything as per output of pass 1
        //initialize symtable from out_symtable.txt
        String s;
        BufferedReader br;
        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/out_symtable.txt")));//reading Symbol table
        while ((s = br.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens
            symtable.add(new SymTuple(st.nextToken(), Integer.parseInt(st.nextToken()), Integer.parseInt(st.nextToken()), st.nextToken())); //adding token into list
        }

        //initialize littable from out_littable.txt

        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/out_littable.txt")));//reading literal table
        while ((s = br.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens
            littable.add(new LitTuple(st.nextToken(), Integer.parseInt(st.nextToken()), Integer.parseInt(st.nextToken()), st.nextToken())); //adding token into list
        }

        //initialize lclist from lclist.txt
        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/lclist.txt")));//reading lclist
        while ((s = br.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, "\n", false); //convert line into tokens
            lclist.add(Integer.parseInt(st.nextToken())); //adding token into list
        }
        System.out.println("\n====== PASS 2 ======\n");
            pass2(); //Run Pass 2


      }
      static void initializeTables() throws Exception {
        symtable = new LinkedList<>();
        littable = new LinkedList<>();
        lclist = new ArrayList<>();
        basetable = new HashMap<>();
        mot = new LinkedList<>();
        pot = new LinkedList<>();
        String s;
        BufferedReader br;
        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/mot.txt")));//reading MOT
        while ((s = br.readLine()) != null) {
            StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens
            mot.add(new Tuple(st.nextToken(), st.nextToken(), st.nextToken(), st.nextToken())); //adding token into list
        }
        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/pot.txt")));//reading POT
        while ((s = br.readLine()) != null) {
            pot.add(s); //adding token into POT list
        }
        Collections.sort(pot); //sorting all the POT as per their index
      }
      static void pass2() throws Exception {
                    line_no = 0; //give line number as 0 for checking output pass 1 file
                    out_pass2 = new PrintWriter(new FileWriter("/home/student/workspace/SPOSL/src/output_pass2.txt"), true);
                    BufferedReader input = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/output_pass1.txt"))); //read output pass 1
                    String s; //to be used
                    System.out.println("Pass 2 input:");
                    while((s = input.readLine()) != null) { //read the complete pass 2 input from pass1 output file
```

```
            System.out.println(s);
            StringTokenizer st = new StringTokenizer(s, " ", false); //dividing line into tokens
            String s_arr[] = new String[st.countTokens()]; //initialize the s_arr
            for(int i=0 ; i < s_arr.length ; i++) {
                s_arr[i] = st.nextToken(); //read all tokens
            }
            if(searchPot2(s_arr) == false) { //check if entry in POT
                searchMot2(s_arr); //if not, check in MOT
            }
            line_no++; //update line no
        }
        System.out.println("\nPass 2 output:");
        input = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/output_pass2.txt")));
        while((s = input.readLine()) != null) {
            System.out.println(s);
        }
    }
    static boolean searchPot2(String[] s) {
        int i = 0; //to be used

        if(s.length == 3) { //check if 3 tokens in a line
            i = 1;
        }
        if(Collections.binarySearch(pot, s[i]) >= 0) { //check all symbols and pseudo-ops in a file by using binary search
            if(s[i].equalsIgnoreCase("USING")) { //if USING occurs
                s = tokenizeOperands(s); //tokenize operands

                if(s[i+1].equals("*")) { //if there is a pointer after USING
                    s[i+1] = lclist.get(line_no) + ""; //get next value as location counter which is line_no
                } else {
                    for(int j=i+1 ; j<s.length ; j++) {
                        int value = getSymbolValue(s[j]); //get symbol value in value
                        if(value != -1) {
                            s[j] = value + ""; //get symbol value
                        }
                    }
                }
                basetable.put(new Integer(s[i+2].trim()), new Integer(s[i+1].trim())); //store base register and offset
            }
            return true; //got POT
        }
        return false; //go for MOT
    }
    static int getSymbolValue(String s) { //get the symbol value from symbol table
        for(SymTuple st : symtable) {
            if(s.equalsIgnoreCase(st.symbol)) {
                return st.value;
            }
        }
        return -1;
    }
    static void searchMot2(String[] s) {
        Tuple t = new Tuple(); //create new MOT object
        int i = 0;
        int j;

        if(s.length == 3) { //if three tokens in a line
            i = 1;
        }
        s = tokenizeOperands(s); //convert line into tokens

        for(Tuple x : mot) { //traverse through MOT entries
            if(s[i].equals(x.mnemonic)) { //get all mnemonics in t
                t = x;
                break;
            }
```

```
}

String output = new String();
String mask = new String();
if(s[i].equals("BNE")) { //mask BNE with 7
        mask = "7";
} else if(s[i].equals("BR")) { //mask BR with 15
        mask = "15";
} else {
        mask = "0";
}
if(s[i].startsWith("B")) { //check for BCR or BR instruction
        if(s[i].endsWith("R")) {
                s[i] = "BCR";
        } else {
                s[i] = "BC";
        }
        List<String> temp = new ArrayList<>();
        for(String x : s) {
                temp.add(x); //get all tokens into temp
        }
        temp.add(i+1, mask); //add masks to temp
        s = temp.toArray(new String[0]); //convert list into arrayList and store in x
}
if(t.type.equals("RR")) { //check for instruction type, if 'RR'
        output = s[i]; //write to output string
        for(j=s[i].length() ; j<6 ; j++) { //get symbol name in output
                output += " ";
        }
        for(j=i+1 ; j<s.length ; j++) { //get symbol value
                int value = getSymbolValue(s[j]);
                if(value != -1) {
                        s[j] = value + "";
                }
        }
```

```
                        output += s[i+1]; //append output
                        for(j=i+2 ; j<s.length ; j++) {
                                output += ", " + s[j]; //append the instruction length
                        }
                } else { //if RX instruction
                        output = s[i]; //get s[i] in output
                        for(j=s[i].length() ; j<6 ; j++) { //get name
                                output += " ";
                        }
                        for(j=i+1 ; j<s.length-1 ; j++) { //get instruction value
                                int value = getSymbolValue(s[j]);
                                if(value != -1) {
                                        s[j] = value + "";
                                }
                        }
                        s[j] = createOffset(s[j]); //create offset of RX type instructions
                        output += s[i+1];
                        for(j=i+2 ; j<s.length ; j++) {
                                output += ", " + s[j]; //get length of instruction
                        }
                }
                out_pass2.println(output); //print output of pass 2
        }
    static String[] tokenizeOperands(String[] s) {
                List<String> temp = new LinkedList<>(); //to be used
                for(int j=0 ; j<s.length-1 ; j++) { //adding all tokens into temp
                        temp.add(s[j]);
                }
                StringTokenizer st = new StringTokenizer(s[s.length-1], " ,", false); //convert line into tokens
                while(st.hasMoreTokens()) {
                        temp.add(st.nextToken()); //adding all tokens
                }
                s = temp.toArray(new String[0]); //convert linked list to array list
                return s;
        }
```

```
  static String createOffset(String s) {
        String original = s; //get s in original
        Integer[] key = basetable.keySet().toArray(new Integer[0]); //get base register number in key
        int offset, new_offset; //to be used
        int index = 0; //to be used
        int value = -1; //to be used
        int index_reg = 0; //to be used
        if(s.startsWith("=")) { //check RX by checking '=' in an output pass 1 line
            value = getLiteralValue(s); //get literal value ahead of '='
        } else {
            int paranthesis = s.indexOf("("); //check '(' in line
            String index_string = new String(); //index_string
            if(paranthesis != -1) { //check index of paranthesis
                s = s.substring(0, s.indexOf("(")); //store substring in s
                index_string = original.substring(original.indexOf("(")+1, original.indexOf(")"));//get index_string '(offset)'
                index_reg = getSymbolValue(index_string); //get symbol value
            }
            value = getSymbolValue(s); //get symbol value here
        }
        offset = Math.abs(value - basetable.get(key[index])); //calculate offset by offset=value in ST - contents of Base Register
        for(int i=1 ; i<key.length ; i++) {
            new_offset = Math.abs(value - basetable.get(key[i])); //calculate offset by offset=value in ST - contents of Base Register
            if(new_offset < offset) { //check if new offset is in range
                offset = new_offset; //give new offset
                index = i; //update index position
            }
        }
        String result = offset + "(" + index_reg + ", " + key[index] + ")"; //represent index_register and base register
        return result; //give in '(index_reg,Base_register)' format
    }
    static int getLiteralValue(String s) {
        s = s.substring(1, s.length());
```

```
                for(LitTuple lt : littable) { //traverse literal table and get literal value
                        if(s.equalsIgnoreCase(lt.literal)) {
                                return lt.value;
                        }
                }
                return -1; //if not present then return -1
        }

}
```

INPUT :

outputpass1.txt

```
        USING   *,15
        LA      15,SETUP
        SR      TOTAL,TOTAL
        USING   SETUP,15
        L       DATABASE,=A(DATA1)
        USING   DATAAREA,DATABASE
        SR      INDEX,INDEX
LOOP    L       AC,DATA1(INDEX)
        AR      TOTAL,AC
        A       AC,=F'5'
        ST      AC,SAVE(INDEX)
        A       INDEX,=F'4'
        C       INDEX,=F'8000'
        BNE     LOOP
        LR      1,TOTAL
        BR      14
```

pot.txt

```
START
END
LTORG
DC
DS
DROP
```

```
USING
EQU
```

out_symtable.txt

```
PRGAM2    0    1     R
AC        2    1     A
INDEX     3    1     A
TOTAL     4    1     A
DATABASE  13   1     A
SETUP     6    1     R
LOOP      12   4     R
SAVE      64   4     R
DATAAREA  76   1     R
DATA1     76   4     R
```

out_littable.txt

```
A(DATA1)  48   4     R
F'5'      52   4     R
F'4'      56   4     R
F'8000'   60   4     R
```

mot.txt

```
LA     01h    4    RX
SR     02h    2    RR
L      03h    4    RX
AR     04h    2    RR
A      05h    4    RX
C      06h    4    RX
BNE    07h    4    RX
LR     08h    2    RR
ST     09h    4    RX
BR     15h    2    RR
```

lclist.txt

```
0
0
4
6
6
6
6
6
6
6
10
10
12
16
18
22
26
30
34
38
40
42
64
76
76
88
88
```

OUTPUT :

output_pass2.txt

```
LA    15, 6(0, 15)

SR    4, 4

L     13, 42(0, 15)

SR    3, 3

L     2, 0(3, 13)

AR    4, 2

A     2, 24(0, 13)

ST    2, 12(3, 13)

A     3, 20(0, 13)

C     3, 16(0, 13)

BC    7, 6(0, 15)

LR    1, 4

BCR   15, 14
```

Assignment No : 3

Title : Implementation of pass-1 of Two Pass Macro Processor

Objective :

1. To study the data structure used in macro-processor implementation
2. To study design and implementation of two pass microprocessor.

Code of Program :

MacroPass1.java

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;

class MntTuple { //INITIALIZATION OF MNT TUPLE (Consist of MNT Index, Macro Name, MDT Index)

  int mnti;
  String name;
  int index;

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

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

public class MacroPass1 {

  static List<MntTuple> mnt; //MNT List
  static List<String> mdt; //MDT List
  static int mntc; //Initialized to 1
  static int mdtc; //Initialized to 1
  static int mdtp; //used in Pass 2
  static BufferedReader input; //reading Files
  static List<List<String>> ala; //Prepare Argument List Array
  static Map<String, Integer> ala_macro_binding; //used for binding ALA

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

  }

  static void pass1() throws Exception {
    String s = new String(); //to be used ahead as line in a code
    input = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/input.txt"))); //reading input file
    PrintWriter output = new PrintWriter(new FileOutputStream("/home/student/workspace/SPOSL/src/output_pass1.txt"), true); //writing into this file
    while ((s = input.readLine()) != null) { //while the code ends
      if (s.equalsIgnoreCase("MACRO")) { //If we get MACRO in code
        processMacroDefinition(); //go for macro processing
      } else {
        output.println(s); //otherwise, print line as it is in file
      }
    }
    System.out.println("ALA:"); //print ALA for pass 1
```

```
      showAla(1); //pass 1 ALA
      System.out.println("\nMNT:"); //print MNT for pass 1
      showMnt();
      System.out.println("\nMDT:"); //print MDT for pass 1
      showMdt();
    }

    static void initializeTables() {
      mnt = new LinkedList<>();
      mdt = new ArrayList<>();
      ala = new LinkedList<>();
      mntc = 1;
      mdtc = 1;
      ala_macro_binding = new HashMap<>();
    }
    static void showAla(int pass) throws Exception {
                PrintWriter out = new PrintWriter(new FileOutputStream("/home/student/workspace/SPOSL/src/out_ala_pass" + pass + ".txt"), true); //write in this file
                for(List l : ala) { //till all Arguments reached
                        System.out.println(l); //print
                        out.println(l); //write to file
                }
        }

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

        static void showMdt() throws Exception {
                PrintWriter out = new PrintWriter(new FileOutputStream("/home/student/workspace/SPOSL/src/out_mdt.txt"), true);
```

```
            for(String l : mdt) {
                System.out.println(l);
                out.println(l);
            }
        }
    static void processMacroDefinition() throws Exception {
            String s = input.readLine(); //reading line of code
            String macro_name = s.substring(0, s.indexOf(" ")); //reading MACRO_NAME
            mnt.add(new MntTuple(mntc, macro_name, mdtc)); //make entry in MNT
            mntc++; //increment MNT Counter/Index
            pass1Ala(s); //call to ALA of pass 1
            StringTokenizer st = new StringTokenizer(s, " ,", false); //convert next line into tokens for MDT
            String x = st.nextToken(); //read next token in x
            for(int i=x.length() ; i<12 ; i++) { //max 12 characters allowed in token
                x += " ";
            }
            String token = new String(); //to be used to store tokens in MDT
            int index;
            token = st.nextToken();
            x += token; //appending all tokens in a line MDT
            while(st.hasMoreTokens()) { //read until all tokens reached
                token = st.nextToken();
                x += "," + token;
            }
            mdt.add(x); //add x into mdt
            mdtc++; //increment MDT Counter
            addIntoMdt(ala.size()-1); //add all ALA into MDT
        }
    static void addIntoMdt(int ala_number) throws Exception {
            String temp = new String(); //to be used
            String s = new String(); //to be used
            List l = ala.get(ala_number); //add all ALA in List l
            boolean isFirst; //to be used
            while(!s.equalsIgnoreCase("MEND")) { //until MEND is reached
                isFirst = true; //keep this true
                s = input.readLine(); //read all MACRO Lines/Instructions
                String line = new String(); //just initialized
                StringTokenizer st = new StringTokenizer(s, " ,", false); //convert line into tokens
                temp = st.nextToken(); //keep next token in temp
                for(int i=temp.length() ; i<12 ; i++) { //check for instruction length
                    temp += " ";
                }
                line += temp; //append temp into line
                while(st.hasMoreTokens()) {
                    temp = st.nextToken(); //read tokens
                    if(temp.startsWith("&")) { //check if it is argument
                        int x = l.indexOf(temp);
                        temp = ",#" + x; //reformatting
                        isFirst = false; //now make it false as it is last keyword in an instruction
                    } else if(!isFirst) { //if not argument then
                        temp = "," + temp; //keep adding into temp
                    }
                    line += temp; //append again
                }
                mdt.add(line); //finally add line into MDT
                mdtc++; //increment MDTC
            }
        }
      static void pass1Ala(String s) {
            StringTokenizer st = new StringTokenizer(s, " ,", false); //converting line into words
            String macro_name = st.nextToken(); //Macro Name stored
            List<String> l = new ArrayList<>(); //ArrayList for adding ALA in one Line
            int index; //used as index for tokens
            while(st.hasMoreTokens()) { //till all tokens are covered
                String x = st.nextToken(); //reading next tokens in x
                if((index = x.indexOf("=")) != -1) { //if parameter is like this (&ARG=DATA1)
                    x = x.substring(0, index); //then take only part before '=' as an Argument
                }
```

```
                        l.add(x); //finally add all arguments into l i.e. in one line
                    }
                    ala.add(l); //pass to ala
                    ala_macro_binding.put(macro_name, ala_macro_binding.size()); //store all arguments
under one MACRO NAME
        }
}
```

INPUT :

input.txt

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

OUTPUT :

output_ala_pass1.txt

```
[&FIRST, &SECOND]
[&ARG1, &ARG2]
```

out_mdt.txt

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

out_mnt.txt

```
1 INCR1, 1
2 INCR2, 5
```

output_pass1.txt

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

Assignment No : 4

Title : Implementation of pass-2 of Two Pass Macro Processor

Objective :

1. To study design and implementation of pass-2 of two pass microprocessor.

Code of Program :

MacroPass2.java

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;

public class MacroPass2 {

  static List<MntTuple> mnt; //MNT List
  static List<String> mdt; //MDT List
  static int mntc; //Initialized to 1
  static int mdtc; //Initialized to 1
  static int mdtp; //used in Pass 2
  static BufferedReader input; //reading Files
  static List<List<String>> ala; //Prepare Argument List Array
  static Map<String, Integer> ala_macro_binding; //used for binding ALA

  public static void main(String args[]) throws Exception {
    initializeTables(); //Initializing everything

    //mnt touple initializing
```

```
        String s;

        BufferedReader br;

        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/
SPOSL/src/out_mnt.txt")));

        //reading Symbol table

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

           StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens

           mnt.add(new MntTuple(Integer.parseInt(st.nextToken()),
st.nextToken(),Integer.parseInt(st.nextToken()))); //adding token into list

        }


        //mdt initializing

        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/
SPOSL/src/out_mdt.txt")));

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

           mdt.add(s);

        }


        mntc = 3;

        mdtc = 9;

        mdtp = 0;


        br = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/
SPOSL/src/out_ala_pass1.txt")));

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

           StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens

           List<String> temp1 = new ArrayList<String>();

           temp1.add(st.nextToken());

           temp1.add(st.nextToken());

           ala.add(temp1);

        }


        ala_macro_binding.put("INCR1",0);

        ala_macro_binding.put("INCR2",1);

        System.out.println("\n===== PASS 2 =====\n");
```

```
    pass2();
  }

  static void initializeTables() {
    mnt = new LinkedList<>();
    mdt = new ArrayList<>();
    ala = new LinkedList<>();
    mntc = 1;
    mdtc = 1;
    ala_macro_binding = new HashMap<>();
  }

  static void pass2() throws Exception {
    input = new BufferedReader(new InputStreamReader(new FileInputStream("/home/student/workspace/SPOSL/src/output_pass1.txt")));
    //pass 1 as INPUT
    PrintWriter output = new PrintWriter(new FileOutputStream("/home/student/workspace/SPOSL/src/output_pass2.txt"), true);
    //used as MACRO Output expansion
    String token = new String();
    String s;
    while ((s = input.readLine()) != null) { //while reading all lines
      StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens
      while (st.hasMoreTokens()) { //till all tokens are reached
        token = st.nextToken();
        if (st.countTokens() > 2) {
          token = st.nextToken();
        }
        MntTuple x = null;
        for (MntTuple m : mnt) {
          if (m.name.equalsIgnoreCase(token)) { //check the MACRO call
            x = m; //take MACRO_NAME into x
            break;
          }
        }
```

```
if (x != null) {
  mdtp = x.index; //update MDT Index in MDTP
  List<String> l = pass2Ala(s); //call to Pass 2 ALA and storing them in l (SET UP ALA2)
  mdtp++; //update MDTP
  String temp = new String();
  while (!(temp = mdt.get(mdtp)).trim().equalsIgnoreCase("MEND")) { //reach until MEND receives in code
    String line = new String();
    StringTokenizer st2 = new StringTokenizer(temp, " ,", false); //divide line into tokens
    for (int i = 0; i < 12; i++) { //check argument length
      line += " ";
    }
    String opcode = st2.nextToken();
    line += opcode;
    for (int i = opcode.length(); i < 24; i++) { //get actual macro expansion over the call
      line += " ";
    }
    line += st2.nextToken(); //append the macro expansion
    while (st2.hasMoreTokens()) { //check further tokens and arguments
      String token2 = st2.nextToken();
      int index;
      if ((index = token2.indexOf("#")) != -1) { //if MDT gets '#'
        line += "," + l.get(Integer.parseInt(token2.substring(index + 1, index + 2))); //append actual argument
      }
    }
    mdtp++; //now update the pointer
    output.println(line); //write to file
    System.out.println(line); //print everything
  }
  break;
} else {
  output.println(s);
  System.out.println(s);
  break;
```

```
            }
        }
    }
    System.out.println("\nALA:");
    showAla(2); //print ALA of pass 2 Over here
}

static List<String> pass2Ala(String s) {
    StringTokenizer st = new StringTokenizer(s, " ", false); //convert line into tokens
    int num_tokens = st.countTokens(); //count of tokens/arguments
    String macro_name = st.nextToken(); //save macro name of these arguments
    int ala_no = ala_macro_binding.get(macro_name); //get complete key value macro binding
    List<String> l = ala.get(ala_no); //take complete ala binding in l
    int ctr = 0;
    StringTokenizer st2 = null;
    try {
        st2 = new StringTokenizer(st.nextToken(), ",", false);
        while (st2.hasMoreTokens()) {
            l.set(ctr, st2.nextToken()); //set all the tokens to l
            ctr++;
        }
    } catch (Exception e) {
        // do nothing
    }
    if (ctr < num_tokens) {
        String s2 = mdt.get(mdtp); //get complete line from MDT and store it in s2
        StringTokenizer st3 = new StringTokenizer(s2, " ,", false);
        String token = new String();
        int index = 0;
        while (st3.hasMoreTokens()) {
            token = st3.nextToken();
            if ((index = token.indexOf("=")) != -1) {
                try {
                    l.set(ctr++, token.substring(index + 1, token.length())); //Again, forget after '=' part
                } catch (Exception e) {
                    // do nothing
                  }
                }
              }
            }
            ala.set(ala_no, l); //substitute all the actual arguments over here (in Pass 2 ALA)
            return l;
          }

          static void showAla(int pass) throws Exception {
            PrintWriter out = new PrintWriter(new FileOutputStream("/home/student/workspace/SPOSL/src/
out_ala_pass" + pass + ".txt"), true); //write in this file
            for (List l : ala) { //till all Arguments reached
              System.out.println(l); //print
              out.println(l); //write to file
            }
          }
        }
```

INPUT :

out_ala_pass12.txt

```
[&FIRST, &SECOND]
[&ARG1, &ARG2]
```

out_mdt.txt

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

out_mnt.txt

```
1 INCR1, 1
2 INCR2, 5
```

output_pass1.txt

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

OUTPUT :

output_pass2.txt

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

output_ala_pass2.txt

```
[[&FIRST,, &SECOND]]
[[&ARG1,, &ARG2]]
```

Assignment No : 5

Title : Implementation of Dynamic Link Library

Objective :

1. To study and understand concept of DLL.
2. To understand JNI
3. To be able to create and use DLL.

Code of Program :

Cal1.java

```
import java.io.*;


import java.util.*;


public class cal1

{


static

            {


            System.loadLibrary("abc");

            }




            private native double add(double a, double b);




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


            {


            Scanner sc= new Scanner(System.in);


double n1, n2;

            System.out.println("enter n1");
n1=sc.nextDouble();

System.out.println("enter n2");

n2=sc.nextDouble();

        System.out.println("Add="+new cal1().add(n1,n2));
        }
}
```

try1.c

```
#include <jni.h>
#include <stdio.h>
#include "cal1.h"

JNIEXPORT jdouble JNICALL Java_cal1_add(JNIEnv *env, jobject obj, jdouble a, jdouble b)
{
return a+b;
}
```

OUTPUT :

```
[fedora@localhost ~]$ javac cal1.java

[fedora@localhost ~]$ javah -jni  cal1

[fedora@localhost ~]locate jni.h
/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc20.x86_64/include/jni.h

[fedora@localhost ~]$ locate jni_md.h
/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc20.x86_64/include/linux/jni_md.h

[fedora@localhost ~]$ gcc -I/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc20.x86_64/include -I/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-2.4.3.0.fc20.x86_64/include/linux -shared -o libabc.so try1.c
```

```
[fedora@localhost ~]$ java -Djava.library.path=`pwd` cal1

enter n1

374

enter n2

7899

Add=8273.0
```

Assignment No : 6

Title : Lexical analyzer for subset of 'Java' program tokenization using LEX.

Objective :

1. To understand working of LEX and lexical analyzer.
2. To understand token generation.
3. To understand file handling with command line arguments using LEX.

Code of Program :

token.l

```
%{
#include<stdio.h>
char fname[20];
struct ST{
char lexname[100],token[100];
};
struct ST s[100];
int cnt=0;
%}
acsp ("public"|"private"|"protected")
keyword ("static"|"class"|"throws"|"import")
datatype ("void"|"int"|"char"|"float"|"String")
inte  [0-9]+
floa [0-9]+"."[0-9]+
operator [=+*/-]
sc [/]{1}[/]{1}[a-zA-Z ]*
mc1 [/]{1}[*]{1}[\na-zA-Z ]*[*]{1}[/]{1}



%%
{acsp}                          { strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"access specifier"); cnt++; }
System\.out\.println     {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"print statement"); cnt++;}
[A-Za-z]+"{"    {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"class name"); cnt++;}
[a-zA-Z]+"("       {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"function name"); cnt++;}
[a-zA-Z]+"["       {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"array name"); cnt++;}
```

```
{keyword}      {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"keyword"); cnt++;}
{datatype}     {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"datatype"); cnt++;}
java\.[a-z]*\.\* {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"package"); cnt++;}
\"[a-zA-Z ]+\"                 {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"output string"); cnt++;}
[a-zA-Z][a-zA-Z0-9_]*                  {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"identifier"); cnt++;}
{operator}                     {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"operator"); cnt++;}
{sc}                           {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"single line comment"); cnt++;}
{mc1}                          {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"multiline comment"); cnt++;}
{inte}               {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"integer"); cnt++;}
{floa}               {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"float"); cnt++;}

.                                      {}
%%
void main()
{
int i;
        printf("\n Enter File name:");
        scanf("%s",fname);
        yyin=fopen(fname,"r");
  yylex();
  printf("Token name \t Lex name\n");
  for(i=0;i<cnt;i++)
  {
        printf("%s\t\t%s\n",s[i].token,s[i].lexname);
  }
}

int yywrap()
{
        return 1;
}
```

input.java

```
import java.io.*;
import java.awt.*;
/* class defination */
public class test{
int a=10.6,b=10;
char c;
//main function begins
public static void main(String args[])
{
a=b+c; /*Computing Addition*/
System.out.println("Be the best"); //printing
add();
}

void add(int a,int b)throws IOException{
a+b=c;/* this is an example
of multiline comment*/
}
}
```

OUTPUT :

```
[exam1@localhost B2]$ lex token.l
[exam1@localhost B2]$ gcc lex.yy.c
[exam1@localhost B2]$ ./a.out

 Enter File name:input.java

import is a keyword.
java.io.* is a package.

import is a keyword.
java.awt.* is a package.
```

/* class defination */ is a multiline comment.

public is an access specifier.

class is a keyword.

test{ is a class name.

public is an access specifier.

int is a datatype.

a is an identifier.

b is an identifier.

= is an operator.

10 is a value.

char is a datatype.

c is an identifier.

//main function begins is a single line comment.

public is an access specifier.

static is a keyword.

void is a datatype.

main(is a function name.

String is a datatype.

args[is an array name.

a is an identifier.

= is an operator.

b is an identifier.

+ is an operator.

c is an identifier.

/*Computing Addition*/ is a multiline comment.

System.out.println is a print statement.

```
"Be the best" is an output  String.
//printing is a single line comment.

add( is a function name.



void is a datatype.
add( is a function name.
int is a datatype.
a is an identifier.
int is a datatype.
b is an identifier.
throws is a keyword.
IOException{ is a class name.

a is an identifier.
+ is an operator.
b is an identifier.
= is an operator.
c is an identifier.
/* this is an example
of multiline comment*/ is a multiline comment.
```

Assignment No : 7

Title : Implementation of lexical analysis phase of compiler to count no. of words, lines and characters of given input file.

Objective :

1. To understand working of LEX.
2. To understand file handling with LEX.

Code of Program :

token.l

```
%{
#include<stdio.h>
char fname[20];
struct ST{
char lexname[100],token[100];
};
struct ST s[100];
int cnt=0;
%}
acsp ("public"|"private"|"protected")
keyword ("static"|"class"|"throws"|"import")
datatype ("void"|"int"|"char"|"float"|"String")
inte  [0-9]+
floa [0-9]+"."[0-9]+
operator [=+*/-]
sc [/]{1}[/]{1}[a-zA-Z ]*
mc1 [/]{1}[*]{1}[\na-zA-Z ]*[*]{1}[/]{1}


%%
{acsp}                     { strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"access specifier"); cnt++; }
System\.out\.println     {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"print statement"); cnt++;}
[A-Za-z]+"{"    {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"class name"); cnt++;}
[a-zA-Z]+"("       {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"function name"); cnt++;}
[a-zA-Z]+"["        {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"array name"); cnt++;}
{keyword}       {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"keyword"); cnt++;}
{datatype}      {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"datatype"); cnt++;}
java\.[a-z]*\.\* {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"package"); cnt++;}
\"[a-zA-Z ]+\"                  {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"output string"); cnt++;}
[a-zA-Z][a-zA-Z0-9_]*                   {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"identifier"); cnt++;}
{operator}                      {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"operator"); cnt++;}
{sc}                            {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"single line comment"); cnt++;}
{mc1}                           {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"multiline comment"); cnt++;}
{inte}              {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"integer"); cnt++;}
{floa}              {strcpy(s[cnt].lexname,yytext); strcpy(s[cnt].token,"float"); cnt++;}

.                                       {}
%%
void main()
{
int i;
            printf("\n Enter File name:");
            scanf("%s",fname);
            yyin=fopen(fname,"r");
   yylex();
   printf("Token name \t Lex name\n");
   for(i=0;i<cnt;i++)
   {
            printf("%s\t\t%s\n",s[i].token,s[i].lexname);
   }
}

int yywrap()
{
            return 1;
}
```

token.java

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


public class token

{

   int var1 = 20; //variable

   int var2 = 30;

   int sum;

            public static void main(String args[])

            {

                        sum = var1 + var2; // addition of two numbers

                  System.out.println("The Sum is :"+sum);

                        System.out.println("Simple Program");




            }

}
```

OUTPUT :

```
(base) fedora@fedora:~$ lex token.l
(base) fedora@fedora:~$ gcc lex.yy.c
(base) fedora@fedora:~$ ./a.out

 Enter File name:token.java

Token name Lex name
keyword import
package java.io.*
keyword import
package java.util.*
access specifier public
keyword class
identifier token
datatype int
identifier var1
operator =
integer 20
single line comment //variable
datatype int
identifier var2
operator =
integer 30
datatype int
identifier sum
access specifier public
```

```
keyword static
datatype void
function name main(
datatype String
array name args[
identifier sum
operator =
identifier var1
operator +
identifier var2
single line comment // addition of two numbers
print statement System.out.println
identifier The
identifier Sum
identifier is
operator +
identifier sum
print statement System.out.println
output string "Simple Program"
```

Assignment No : 8

Title : Implementation of syntax analysis phase of compiler to validate type and syntax of variable declaration in Java.

Objective :

1. To understand working of YACC to recognize sentences.
2. To understand how LEX and YACC work together.
3. To understand implementation of grammar for recognition of variable declaration in Java.

Code of Program :

ltoken.l

```
%{
        #include <stdio.h>
        #include<string.h>
        #include"y.tab.h"
        struct tokenrecord{
        char lexname[50];
        char tokenname[100];
        };
        struct tokenrecord tr[50];
        int rcnt=0;

%}
%%



"int "|"String "|"char "|"float "|"double " { return DT;
                                             //strcpy(tr[rcnt].lexname,yytext);
                                             // strcpy(tr[rcnt].tokenname,"Datatype");
                                             // rcnt++;
                                          }

"["             { return oBR;
                        //strcpy(tr[rcnt].lexname,yytext);
                        //strcpy(tr[rcnt].tokenname,"Bracket");
```

```
                                //rcnt++;
                                }

"]"                 { return cBR;
                                //strcpy(tr[rcnt].lexname,yytext);
                                //strcpy(tr[rcnt].tokenname,"Bracket");
                                //rcnt++;
                                }
"new"           {
            return KW;
        }

[0-9]([0-9])*       {  return INT;
                                //strcpy(tr[rcnt].lexname,yytext);
                                //strcpy(tr[rcnt].tokenname,"Integer");
                                //rcnt++;
                                }
[0-9]([0-9])*"."[0-9]([0-9])*       { return FLT;
                                            // strcpy(tr[rcnt].lexname,yytext);
                                            // strcpy(tr[rcnt].tokenname,"Float");
                                            // rcnt++;
                                            }

";"             {
                   return SEM;
                                  //strcpy(tr[rcnt].lexname,yytext);
                                 // strcpy(tr[rcnt].tokenname,"Variable");
                                 // rcnt++;
                }

"="             {
                   return OP;
                                // strcpy(tr[rcnt].lexname,yytext);
                                // strcpy(tr[rcnt].tokenname,"Variable");
                               //  rcnt++;
```

```
                    }
","                       {
                       return COM;
                        }

"\n"          {
                return NL;
              }

\t          {;}




[_a-zA-Z]([a-z]|[0-9]|_)*    {return VAR;
                              //strcpy(tr[rcnt].lexname,yytext);
                             // strcpy(tr[rcnt].tokenname,"Variable");
                             // rcnt++;
                              }

%%

int yywrap()
{
  return 1;
}
```

ptoken.y

```
%{
       #include <stdio.h>
            #include<string.h>
            char fname[20];

%}
```

```
%token DT SEM NL OP INT FLT VAR COM cBR oBR KW

%%
S:S DT E SEM
 |S NL  {printf("\n\tValid Syntax Declaration");}
 |
 ;
E:VAR
 |VAR OP INT
 |VAR OP FLT
 |VAR COM E
 |VAR oBR cBR {printf("Arr1\n");}
 ;

%%

extern FILE *yyin;
main()
{
     printf("\nEnter the file name :");
         scanf("%s",fname);
         yyin = fopen(fname,"r");
         yyparse();
}


void yyerror()
{
  printf("\nInvalid Syntax");

}
```

OUTPUT :

```
Token name Lex name
datatype int
identifier var1
operator =
integer 20
datatype int
identifier var2
operator =
integer 30
datatype int
identifier sum
datatype String
identifier a
identifier double
identifier c
operator =
float 10.5
datatype int
identifier a1
identifier b
operator =
integer 25
datatype int
array name st[
```

Assignment No : 9

Title : Implementation of syntax analysis phase of compiler to recognize simple and compound sentences.

Objective :

1. To understand working of YACC to recognize sentences.
2. To understand how LEX and YACC work together.
3. To understand implementation of grammar for sentence recognition.

Code of Program :

sentence.l

```
%{
#include "y.tab.h" //Contains Token Definiation
%}
%%
[\t ] ;     //IGNORE WHITE SPACES
am|is|are|have|has|can|will|shall|eat|sing|go|goes { printf("VERB\t==>%s\n",yytext);return VERB;}
very|simply|gently { printf("VERB\t==>%s\n",yytext);return(ADVERB); }
and|or|also|so|but|if|then {printf("CONJUNCTION\t==>%s\n",yytext);return (CONJUNCTION);}
fast|good|honest {printf("ADJECTIVE\t==>%s\n",yytext);return (ADJECTIVE);}
I|he|she|we|they|you|this {printf("PRONOUN\t==>%s\n",yytext);return (PRONOUN);}
in|on|to {printf("PREPOSITION\t==>%s\n",yytext);return (PREPOSITION);}
[a-zA-Z]+ {printf("NOUN\t==>%s\n",yytext);return (NOUN);}
. ; //IGNORE ANYTHING ELSE
%%
int yywrap()
{
return 1;
}
```

sentence.y

```
%{
#include<stdio.h>
void yyerror(char*);
int yylex();
FILE* yyin;
%}

%token NOUN PRONOUN ADJECTIVE VERB ADVERB CONJUNCTION PREPOSITION

%%
sentence: compound { printf("COMPOUND SENTENCE\n");}
        |
        simple {printf("SIMPLE SENTENCE\n");}
        ;
simple: subject VERB object;

compound: subject VERB object CONJUNCTION subject VERB object;

subject: NOUN|PRONOUN;
```

```
object: NOUN|ADJECTIVE NOUN|ADVERB NOUN|PREPOSITION NOUN;
%%
void yyerror(char *s)
{
printf("ERROR:%s",s);
}
int main(int argc,char* argv[])
{
yyin=fopen(argv[1],"r");
yyparse();
fclose(yyin);
return 0;
}
```

OUTPUT :

```
fedora@fedora:~$ yacc sentence.y
(base) fedora@fedora:~$ lex sentence.l
(base) fedora@fedora:~$ gcc lex.yy.c y.tab.c
(base) fedora@fedora:~$ ./a.out a.txt
NOUN ==>ram
VERB ==>is
NOUN ==>boy
CONJUNCTION ==>and
PRONOUN ==>he
VERB ==>is
NOUN ==>student
COMPOUND SENTENCE

(base) fedora@fedora:~$ yacc sentence.y
(base) fedora@fedora:~$ lex sentence.l
(base) fedora@fedora:~$ gcc lex.yy.c y.tab.c
(base) fedora@fedora:~$ ./a.out a.txt
NOUN ==>ram
VERB ==>is
PREPOSITION ==>in
NOUN ==>SCOE

SIMPLE SENTENCE
```

Assignment No : 11

Title : Banker’s Algorithm

Objective :

1. To understand safe and unsafe state to handle deadlock situation in the system.
2. To handle deadlock condition.
3. To implement banker’s algorithm to avoid deadlock.

Code of Program :

Bankers.java

```
// Java program to illustrate Banker's Algorithm
import java.util.*;

class GFG
{

// Number of processes
static int P = 5;

// Number of resources
static int R = 3;

// Function to find the need of each process
static void calculateNeed(int need[][], int maxm[][],
                                        int allot[][])
{
        // Calculating Need of each P
        for (int i = 0 ; i < P ; i++)
                for (int j = 0 ; j < R ; j++)

                        // Need of instance = maxm instance -
                        //                                  allocated instance
                        need[i][j] = maxm[i][j] - allot[i][j];
}

// Function to find the system is in safe state or not
static boolean isSafe(int processes[], int avail[], int maxm[][],
                        int allot[][])
{
        int [][]need = new int[P][R];

        // Function to calculate need matrix
        calculateNeed(need, maxm, allot);

        // Mark all processes as infinish
        boolean []finish = new boolean[P];

        // To store safe sequence
        int []safeSeq = new int[P];

        // Make a copy of available resources
        int []work = new int[R];
        for (int i = 0; i < R ; i++)
                work[i] = avail[i];
```

```
// While all processes are not finished
// or system is not in safe state.
int count = 0;
while (count < P)
{
    // Find a process which is not finish and
    // whose needs can be satisfied with current
    // work[] resources.
    boolean found = false;
    for (int p = 0; p < P; p++)
    {
        // First check if a process is finished,
        // if no, go for next condition
        if (finish[p] == false)
        {
            // Check if for all resources of
            // current P need is less
            // than work
            int j;
            for (j = 0; j < R; j++)
                if (need[p][j] > work[j])
                    break;

            // If all needs of p were satisfied.
            if (j == R)
            {
                // Add the allocated resources of
                // current P to the available/work
                // resources i.e.free the resources
                for (int k = 0 ; k < R ; k++)
                    work[k] += allot[p][k];

                // Add this process to safe sequence.
                safeSeq[count++] = p;

                // Mark this p as finished
                finish[p] = true;

                found = true;
            }
        }
    }

    // If we could not find a next process in safe
    // sequence.
    if (found == false)
    {
        System.out.print("System is not in safe state");
        return false;
    }
}

// If system is in safe state then
// safe sequence will be as below
System.out.print("System is in safe state.\nSafe"
        +" sequence is: ");
    for (int i = 0; i < P ; i++)
        System.out.print(safeSeq[i] + " ");

    return true;
}

// Driver code
public static void main(String[] args)
{
    int processes[] = {0, 1, 2, 3, 4};

    // Available instances of resources
    int avail[] = {3, 3, 2};

    // Maximum R that can be allocated
    // to processes
    int maxm[][] = {{7, 5, 3},
                    {3, 2, 2},
                    {9, 0, 2},
                    {2, 2, 2},
                    {4, 3, 3}};

    // Resources allocated to processes
    int allot[][] = {{0, 1, 0},
                    {2, 0, 0},
                    {3, 0, 2},
                    {2, 1, 1},
                    {0, 0, 2}};

    // Check system is in safe state or not
    isSafe(processes, avail, maxm, allot);
}
}
```

OUTPUT :

```
System is in safe state.
Safe sequence is: 1 3 4 0 2
```

Assignment No : 14

Title : Simulation of paging

Objective :

1. To study page replacement policies to understand memory management.
2. To understand efficient frame management using replacement policies.

Code of Program :

Test.java

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;

class Test
{
        // Method to find page faults using indexes
        static int pageFaults(int pages[], int n, int capacity)
        {
                // To represent set of current pages. We use
                // an unordered_set so that we quickly check
                // if a page is present in set or not
                HashSet<Integer> s = new HashSet<>(capacity);

                // To store least recently used indexes
                // of pages.
                HashMap<Integer, Integer> indexes = new HashMap<>();

                // Start from initial page
                int page_faults = 0;
                for (int i=0; i<n; i++)
                {
                        // Check if the set can hold more pages
                        if (s.size() < capacity)
                        {
                                // Insert it into set if not present
                                // already which represents page fault
                                if (!s.contains(pages[i]))
                                {
                                        s.add(pages[i]);

                                        // increment page fault
                                        page_faults++;
                                }

                                // Store the recently used index of
                                // each page
                                indexes.put(pages[i], i);
                        }

                        // If the set is full then need to perform lru
                        // i.e. remove the least recently used page
```

```
            // and insert the current page
            else
            {
                // Check if current page is not already
                // present in the set
                if (!s.contains(pages[i]))
                {
                    // Find the least recently used pages
                    // that is present in the set
                    int lru = Integer.MAX_VALUE, val=Integer.MIN_VALUE;

                    Iterator<Integer> itr = s.iterator();

                    while (itr.hasNext()) {
                        int temp = itr.next();
                        if (indexes.get(temp) < lru)
                        {
                            lru = indexes.get(temp);
                            val = temp;
                        }
                    }

                    // Remove the indexes page
                    s.remove(val);
                //remove lru from hashmap
                indexes.remove(val);
                    // insert the current page
                    s.add(pages[i]);

                    // Increment page faults
                    page_faults++;
                }

                // Update the current page index
                indexes.put(pages[i], i);
            }
        }

        return page_faults;
    }

    // Driver method
    public static void main(String args[])
    {
        int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

        int capacity = 4;

        System.out.println(pageFaults(pages, pages.length, capacity));
    }
}
```

OUTPUT : 6

Optimal Alogorith

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

// creation of the main class to implement Optimal page replacement algorithm
public static void main(pagestring[] args) throws IOException
{
Countbuffer bfr = new Countbuffer(new InputStreamReader(System.in));
int frames, pointer = 0, hit = 0, fault = 0,strng_size;
boolean isFull = false;
int buffer[];
int ref[];
int mem_layout[][];

//Entering the number of frames
System.out.println(" Enter the total number of Frames: ");
frames = Integer.parseInt(br.readLine());

//Entering the string size of the reference
System.out.println(" Enter the reference string size:");
strng_size = Integer.parseInt(br.readLine());

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

//code to enter the reference string to carry out optimal page replacement
System.out.println(" Enter the reference string: ");
for(int i = 0; i < strng_size; i++)
{
ref[i] = Integer.parseInt(br.readLine());
}

System.out.println();
for(int i = 0; i < strng_size; i++)
{
int search = -1;
for(int j = 0; j < frames; j++)
{
if(buffer[j] == ref[i])
{
search = j;
hit++;
break;
}
}
// code to update the stack checking its capacity
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((ref[j] == buffer[k]) && (index_flag[k] == false))
{
index[k] = j;
index_flag[k] = true;
break;
}
}
}

//updating pointer to the correct memory location after checking capacity
buffer[pointer] = ref[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];
}

// code to display the number strings
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("Hits: " + hit);
System.out.println("Hit Ratio: " + (float)((float)hit/str_len));
System.out.println("Faults: " + fault);
}

}
```

OUTPUT :

```
Enter the total number of Frames:
3
Enter the reference string size:
20
Enter the reference string:
1
2
3
2
1
5
2
1
6
2
5
6
3
1
3
6
1
2
4
3

1 1 1 1 1 1 1 1 6 6 6 6 6 6 6 6 6 2 4 4
-1 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1
-1 -1 3 3 3 5 5 5 5 5 5 5 3 3 3 3 3 3 3 3
Hits: 11
Hit Ratio: 0.55
Faults: 9
```