Pass1.java

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

class Pass1 {
   public static void main(String args[]) throws NullPointerException, FileNotFoundException {
      String[] REG = {"ax", "bx", "cx", "dx"};
      String[] IS = {"stop", "add", "sub", "mult", "mover", "movem", "comp", "be", "div", "read"};
      String[] DL = {"ds", "dc"};

      int temp1 = 0;
      int f =0;

      int total_symb = 0, total_ltr = 0, optab_cnt = 0, pooltab_cnt = 0, loc = 0, temp, pos;
      boolean start = false, end = false, fill_addr = false, ltorg = false;

      Obj[] literal_table = new Obj[10];
      Obj[] symb_table = new Obj[10];
      Obj[] optab = new Obj[60];
      Pooltable[] pooltab = new Pooltable[5];

      String line;

      try {
         BufferedReader br = new BufferedReader(new
FileReader("C:\\Users\\Vishal\\OneDrive\\Desktop\\SPOS\\SPOS\\src\\sample.txt"));
         BufferedWriter bw = new BufferedWriter(new
FileWriter("C:\\Users\\Vishal\\OneDrive\\Desktop\\SPOS\\SPOS\\Output.txt"));

         while ((line = br.readLine()) != null && !end) {
            String[] tokens = line.split(" ", 4);
```

```
if (loc != 0 && !ltorg) {
    if (f == 1) {
        ltorg = false;
        loc = loc + temp1 - 1;
        bw.write("\n" + String.valueOf(loc));
        f = 0;
        loc++;
    } else {
        bw.write("\n" + String.valueOf(loc));
        ltorg = false;
        loc++;
    }
}
ltorg = fill_addr = false;

for (int k = 0; k < tokens.length; k++) {
    pos = -1;
    if (start) {
        loc = Integer.parseInt(tokens[k]);
        start = false;
    }
    switch (tokens[k]) {
        case "start":
            start = true;
            pos = 1;
            bw.write("\t (AD, " + pos + ")");
            break;
        case "end":
            end = true;
            pos = 2;
            bw.write("\t(AD, " + pos + ")\n");
```

```
                for (temp = 0; temp < total_ltr; temp++) {
                    if (literal_table[temp].addr == 0) {
                        literal_table[temp].addr = loc - 1;
                        bw.write("\t(DL, 2) \t (C, " + literal_table[temp].name + ")" + "\n" + loc++);
                    }
                }
                if (pooltab_cnt == 0) {
                    pooltab[pooltab_cnt++] = new Pooltable(0, temp);
                } else {
                    pooltab[pooltab_cnt] = new Pooltable(pooltab[pooltab_cnt - 1].first +
pooltab[pooltab_cnt - 1].total_literals, total_ltr - pooltab[pooltab_cnt - 1].first - 1);
                    pooltab_cnt++;
                }
                break;
            case "origin":
                pos = 3;
                bw.write("\t(AD, " + pos + ")");
                pos = search(tokens[++k], symb_table, total_symb);
                k++;
                bw.write("\t(C, " + (symb_table[pos].addr) + ")");
                loc = symb_table[pos].addr;
                break;
            case "ltorg":
                ltorg = true;
                pos = 5;
                bw.write("\t(AD, " + pos + ")\n");
                for (temp = 0; temp < total_ltr; temp++) {
                    if (literal_table[temp].addr == 0) {
                        literal_table[temp].addr = loc - 1;
                        bw.write("\t(DL, 2) \t (C, " + literal_table[temp].name + ")" + "\n" + loc++);
                    }
```

```
                    }
                    if (pooltab_cnt == 0) {
                        pooltab[pooltab_cnt++] = new Pooltable(0, temp);
                    } else {
                        pooltab[pooltab_cnt] = new Pooltable(pooltab[pooltab_cnt - 1].first +
pooltab[pooltab_cnt - 1].total_literals, total_ltr - pooltab[pooltab_cnt - 1].first - 1);
                        pooltab_cnt++;
                    }
                    break;
                case "equ":
                    pos = 4;
                    bw.write("\t(AD, " + pos + ")");
                    String prev_token = tokens[k - 1];
                    int posi = search(prev_token, symb_table, total_symb);
                    pos = search(tokens[++k], symb_table, total_symb);
                    symb_table[posi].addr = symb_table[pos].addr;
                    bw.write("\t(S," + (pos + 1) + ")");
                    break;
                default:
                    if (pos == -1) {
                        pos = search(tokens[k], IS);
                        if (pos != -1) {
                            bw.write("\t(IS, " + (pos) + ")");
                            optab[optab_cnt++] = new Obj(tokens[k], pos);
                        } else {
                            pos = search(tokens[k], DL);
                            if (pos != -1) {
                                if (pos == 0) f = 1;
                                bw.write("\t(DL, " + (pos + 1) + ")");
                                optab[optab_cnt++] = new Obj(tokens[k], pos);
                                fill_addr = true;
```

```
                    } else if (tokens[k].matches("[a-zA-Z]+:")) {
                        pos = search(tokens[k], symb_table, total_symb);
                        if (pos == -1) {
                            symb_table[total_symb++] = new Obj(tokens[k].substring(0,
tokens[k].length() - 1), loc - 1);
                            bw.write("\t(S, " + total_symb + ")");
                            pos = total_symb;
                        }
                    } else {
                        pos = search(tokens[k], REG);
                        if (pos != -1) {
                            bw.write("\t(RG, " + (pos + 1) + ")");
                        } else {
                            if (tokens[k].matches("='\\d+'")) {
                                String s = tokens[k].substring(2, 3);
                                literal_table[total_ltr++] = new Obj(s, 0);
                                bw.write("\t(L, " + total_ltr + ")");
                            } else if (tokens[k].matches("\\d+") || tokens[k].matches("\\d+H") ||
tokens[k].matches("\\d+h") || tokens[k].matches("\\d+D") || tokens[k].matches("\\d+d")) { //
constant
                                bw.write("\t(C, " + tokens[k] + ")");
                                temp1 = Integer.parseInt(tokens[k]);
                            } else {
                                pos = search(tokens[k], symb_table, total_symb);
                                if (fill_addr && pos != -1) {
                                    symb_table[pos].addr = loc - 1;
                                    fill_addr = false;
                                } else if (pos == -1) {
                                    symb_table[total_symb++] = new Obj(tokens[k], 0);
                                    bw.write("\t(S," + total_symb + ")");
                                } else {
                                    bw.write("\t(S," + pos + ")");
```

```
                        }
                    }
                }
            }
        }
    }
    break;
  }
 }
}

System.out.println("\n** SYMBOL TABLE **");
System.out.println("\nSYMBOL\tADDRESS");
for (int i = 0; i < total_symb; i++)
  System.out.println(symb_table[i].name + "\t" + symb_table[i].addr);

System.out.println("\n** POOL TABLE **");
System.out.println("\nPOOL\tTOTAL LITERALS");
for (int i = 0; i < pooltab_cnt; i++)
  System.out.println(pooltab[i].first + "\t" + pooltab[i].total_literals);

System.out.println("\n** LITERAL TABLE **");
System.out.println("\nIndex\tLITERAL\tADDRESS");
for (int i = 0; i < total_ltr; i++) {
  if (literal_table[i].addr == 0) literal_table[i].addr = loc++;
  System.out.println((i + 1) + "\t" + literal_table[i].name + "\t" + literal_table[i].addr);
}

System.out.println("\n** OPTABLE **");
System.out.println("\nMNEMONIC\tOPCODE");
for (int i = 0; i < IS.length; i++)
```

```
            System.out.println(IS[i] + "\t\t" + i);

        br.close();
        bw.close();

    } catch (Exception e) {
        System.out.println("Error while reading the file");
        e.printStackTrace();
    }

    try {
        BufferedReader br = new BufferedReader(new FileReader("output.txt"));
        System.out.println("\n** Output.txt **\n");
        while ((line = br.readLine()) != null)
            System.out.println(line);
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static int search(String token, String[] list) {
    for (int i = 0; i < list.length; i++)
        if (token.equalsIgnoreCase(list[i]))
            return i;
    return -1;
}
public static int search(String token, Obj[] list, int cnt) {
    for (int i = 0; i < cnt; i++)
        if (token.equalsIgnoreCase(list[i].name))
            return i;
```

```
        return -1;
    }
}
```

Obj.java

```
public class Obj {
String name;
int addr;
Obj(String nm, int address)
{
this.name=nm;
this.addr=address;
}

}
```

Pooltable.java

```
public class Pooltable {
int first,total_literals;
public Pooltable(int f, int l) {
// TODO Auto-generated constructor stub
this.first=f;
this.total_literals=l;
}
}
```

sample.txt

```
start 100
```

```
movr ax 05
mover bx 10
up: add ax bx
movem a ='5'
origin up
ltorg
movem b ='7'
ds a 02
dc b 10
end
```

Output.txt

empty

output:--

```
          (AD, 1) (C, 100)
100       (S,1)     (RG, 1)  (C, 05)
101       (IS, 4)   (RG, 2)  (C, 10)
102       (S, 2)    (IS, 1)   (RG, 1)  (RG, 2)
103       (IS, 5)   (S,3)     (L, 1)
104       (AD, 3)  (C, 102)
102       (AD, 5)
          (DL, 2)   (C, 5)
103       (IS, 5)   (S,4)     (L, 2)
104       (DL, 1)  (C, 02)
106       (DL, 2)  (C, 10)
107       (AD, 2)
          (DL, 2)   (C, 7)
108
```