pass2.java

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

public class pass2 {
    static Obj[] symb_table = new Obj[10];
    static Obj[] literal_table = new Obj[10];
    static int symb_found = 0;

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

        Scanner sc = new Scanner(System.in);
        System.out.println("ENTER TOTAL NUMBER OF SYMBOLS: ");
        int total_symb = sc.nextInt();
        int pos, num;
        for (int i = 0; i < total_symb; i++) {
            symb_table[i] = new Obj("", 0);
            System.out.println("ENTER SYMBOL NAME: ");
            symb_table[i].name = sc.next();
            System.out.println("ENTER SYMBOL ADDRESS: ");
            symb_table[i].addr = sc.nextInt();
        }
        System.out.println("ENTER TOTAL NUMBERS OF LITERALS: ");
        int total_ltr = sc.nextInt();
        for (int i = 0; i < total_ltr; i++) {
            literal_table[i] = new Obj("", 0);
            System.out.println("ENTER LITERAL NAME: ");
            literal_table[i].name = sc.next();
            System.out.println("ENTER LITERAL ADDRESS: ");
            literal_table[i].addr = sc.nextInt();
        }
        System.out.println("************SYMBOL***********");
        System.out.println("\nSYMBOL\t ADDRESS");
        for (int i = 0; i < total_symb; i++)
            System.out.println(symb_table[i].name + "\t" + symb_table[i].addr);


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


        BufferedReader br2 = new BufferedReader(new
FileReader("C:\\Users\\Vishal\\OneDrive\\Desktop\\SPOS\\AssemblerPass2\\src\\Output.txt"));
        String line;
        boolean symbol_error = false, undef_mnemonic = false;
        System.out.println("\n********OUTPUT FILE**********\n");


        lab:
        while ((line = br2.readLine()) != null) {
            String[] token_list = line.split("\\s+", 5);
            symbol_error = undef_mnemonic = false;
            lab1:
            for (String token : token_list) {
                if (token.length() > 0) {
                    pos = -1;


                    // If token is "---", print it as is (error case)
                    if (token.matches("---")) {
                        System.out.print("\t---");
                        undef_mnemonic = true;
                    } else if (token.matches("[0-9]+")) {
                        // If the token is a number, print it
                        System.out.print("\n\n" + token);
                    } else {
```

```
// If token is not numeric, extract the letters and number
String letters = token.replaceAll("[^a-zA-Z]+", "");
String numberStr = token.replaceAll("[^0-9]+", "");

// Check if numberStr is not empty before parsing it
if (!numberStr.isEmpty()) {
    num = Integer.parseInt(numberStr);
} else {
    continue; // Skip if no number to parse
}

if (token.matches("\\([0-9]+\\)")) {
    System.out.print("\t" + num);
} else {
    switch (letters.toUpperCase()) {
        case "S":
            if (symb_table[num - 1].addr == 0) {
                System.out.print("\t---");
                symbol_error = true;
            } else {
                System.out.println("\t" + symb_table[num - 1].addr);
            }
            break;

        case "L":
            System.out.println("\t" + literal_table[num - 1].addr);
            break;

        case "AD":
            System.out.print("\n");
            continue lab;
```

```
                    case "DL":
                        switch (num) {
                            case 1:
                                System.out.print("\n");
                                continue lab;

                            case 2:
                                System.out.print("\t00\t00");
                        }
                        continue lab1;

                    case "C":
                        System.out.print("\t" + num);
                        break;

                    default:
                        System.out.print("\t" + "00" + num);
                }
            }
        }
    }
}

    if (symbol_error) {
        System.out.println("\n SYMBOL IS NOT DEFINED\n");
    }
    if (undef_mnemonic)
        System.out.println("\n\n INVALID MNEMONIC");

}
int[] flag = new int[total_symb];
for (int i = 0; i < total_symb; i++) {
```

```
            symb_found = 0;
            for (int j = 0; j < total_symb; j++) {
                if (symb_table[i].name.equalsIgnoreCase(symb_table[j].name) && flag[j] == 0) {
                    symb_found++;
                    flag[i] = flag[j] = 1;
                }
                if (symb_found > 1) {
                    System.out.println("\n\n " + symb_table[i].name + " IS DUPLICATE SYMBOL");
                }

            }

        }
        br2.close();
        sc.close();
    }

}
```

Pool.java

```
class Pooltable
{
        int first,total_literals;
        public Pooltable(int f, int l) {

        this.first=f;
        this.total_literals=l;
        }
}
```

Obj.java

```
class Obj {
    String name;
    int addr;

    Obj(String name, int addr) {
        this.name = name;
        this.addr = addr;
    }
}
```

Output.txt

```
(AD,1) (C,100)
100 (IS,5) (RG,1) (C,05)
101 (IS,5) (RG,2) (C,10)
102 (S,1) (IS,2) (RG,1) (RG,2)
103 (IS,6) (S,2) (L.1)
104 (IS,4) (RG, 1) (S,1)
105 (AD,3) (C,102)
102 (DL,1) (C,5)
103 (IS,6) (S.3) (L.2)
104 (IS,6) (S.4) (L.3)
105 (DL,1) (C,8)
106 (DL,1) (C,8)
107 (IS,6) (S,2) (L,4)
108 (IS.6) (S,3) (L.5)
109 (DL.1) (C,02)
110 (DL,2) (C,10)
111 (DL,1) (C,09)
112 (S,5) (AD,4) (S,1)
113 (AD,2) (DL.1) (C.7)
114 (DL,1) (C,8)
```

Final output:-

```
ENTER TOTAL NUMBER OF SYMBOLS:
5
ENTER SYMBOL NAME:
```

```
up
ENTER SYMBOL ADDRESS:
102
ENTER SYMBOL NAME:
a
ENTER SYMBOL ADDRESS:
109
ENTER SYMBOL NAME:
b
ENTER SYMBOL ADDRESS:
110
ENTER SYMBOL NAME:
c
ENTER SYMBOL ADDRESS:
111
ENTER SYMBOL NAME:
next
ENTER SYMBOL ADDRESS:
102
ENTER TOTAL NUMBERS OF LITERALS:
5
ENTER LITERAL NAME:
5
ENTER LITERAL ADDRESS:
102
ENTER LITERAL NAME:
8
ENTER LITERAL ADDRESS:
105
ENTER LITERAL NAME:
8
ENTER LITERAL ADDRESS:
```

```
106
ENTER LITERAL NAME:
7
ENTER LITERAL ADDRESS:
113
ENTER LITERAL NAME:
8
ENTER LITERAL ADDRESS:
114
************SYMBOL***********

SYMBOL          ADDRESS
up      102
a       109
b       110
c       111
next    102
************LITERAL TABLE***********

INDEX   LITERALADDRESS
1       5       102
2       8       105
3       8       106
4       7       113
5       8       114

********OUTPUT FILE**********




100     005     001     5
```

101	005	002	10
102	102		
	002	001	002
103	006	109	
	102		
104	004	001	102
105			
102			
103	006	110	
	105		
104	006	111	
	106		
105			
106			

107 006 109

113

108 006 110

114

109

110 00 00 10

111

112 102

113

114