**Symbol Table Generation – Problems (1) to (3)**

**package** lpcc;

**import** java.util.\*;

**public** **class** SymbolTableGenerator {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

Map<String, Integer> symbolTable = **new** LinkedHashMap<>();

List<String> code = **new** ArrayList<>();

System.***out***.println("Enter assembly code line by line (type END to stop):");

**while** (**true**) {

String line = sc.nextLine().trim();

code.add(line);

**if** (line.equals("END")) **break**;

}

**int** LC = 0;

**for** (String line : code) {

String[] tokens = line.split("\\s+");

**if** (tokens[0].equals("START")) {

LC = Integer.*parseInt*(tokens[1]);

**continue**;

}

**if** (tokens.length >= 3 && tokens[1].equals("DS")) {

symbolTable.put(tokens[0], LC);

LC += Integer.*parseInt*(tokens[2]);

**continue**;

}

**if** (!Arrays.*asList*("READ", "MOVER", "COMP", "BC", "STOP", "END").contains(tokens[0])) {

symbolTable.put(tokens[0], LC);

}

**if** (!tokens[0].equals("END")) {

LC++;

}

}

System.***out***.println("Symbol Table:");

System.***out***.println("Symbol\tAddress");

**for** (Map.Entry<String, Integer> entry : symbolTable.entrySet()) {

System.***out***.println(entry.getKey() + "\t" + entry.getValue());

}

sc.close();

}

}

**Literal Table Generation – Problems (4) to (6)**

**package** lpcc;

**import** java.util.\*;

**public** **class** LiteralTableGenerator {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

List<String> code = **new** ArrayList<>();

Map<String, Integer> literalTable = **new** LinkedHashMap<>();

System.***out***.println("Enter assembly code line by line (type END to stop):");

**int** LC = 0;

**while** (**true**) {

String line = sc.nextLine().trim();

code.add(line);

**if** (line.equals("END")) **break**;

}

**for** (String line : code) {

String[] tokens = line.split("\\s+");

// START directive

**if** (tokens[0].equals("START")) {

LC = Integer.*parseInt*(tokens[1]);

**continue**;

}

// Detect and store unique literals

**for** (String token : tokens) {

**if** (token.startsWith("='") && token.endsWith("'")) {

**if** (!literalTable.containsKey(token)) {

literalTable.put(token, LC);

LC++; // Each literal takes one memory word

}

}

}

// Skip END

**if** (!tokens[0].equals("END")) {

LC++;

}

}

// Print Literal Table

System.***out***.println("\nLiteral Table:");

System.***out***.println("Literal\tAddress");

**for** (Map.Entry<String, Integer> entry : literalTable.entrySet()) {

System.***out***.println(entry.getKey() + "\t" + entry.getValue());

}

sc.close();

}

}

**Pool Table Generation – Problems (7) to (9)**

**package** lpcc;

**import** java.util.\*;

**public** **class** PoolTableGenerator {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

List<String> code = **new** ArrayList<>();

Map<String, Integer> literalTable = **new** LinkedHashMap<>();

List<Integer> poolTable = **new** ArrayList<>();

List<String> literalOrder = **new** ArrayList<>();

**int** LC = 0;

System.***out***.println("Enter assembly code line by line (type END to stop):");

poolTable.add(1); // Pool 1 starts at literal 1

**while** (**true**) {

String line = sc.nextLine().trim();

code.add(line);

**if** (line.equals("END")) **break**;

}

**for** (String line : code) {

String[] tokens = line.split("\\s+");

**if** (tokens[0].equals("START")) {

LC = Integer.*parseInt*(tokens[1]);

**continue**;

}

**for** (String token : tokens) {

**if** (token.startsWith("='") && token.endsWith("'")) {

**if** (!literalTable.containsKey(token)) {

literalTable.put(token, -1); // placeholder

literalOrder.add(token);

}

}

}

**if** (tokens[0].equals("LTORG") || tokens[0].equals("END")) {

**for** (String lit : literalOrder) {

**if** (literalTable.get(lit) == -1) {

literalTable.put(lit, LC);

LC++;

}

}

**if** (!tokens[0].equals("END")) {

poolTable.add(literalOrder.size() + 1); // next pool starts

}

}

**if** (!tokens[0].equals("END") && !tokens[0].equals("LTORG")) {

LC++;

}

}

System.***out***.println("\nLiteral Table:");

System.***out***.println("Index\tLiteral\tAddress");

**int** index = 1;

**for** (String lit : literalOrder) {

System.***out***.println(index + "\t" + lit + "\t" + literalTable.get(lit));

index++;

}

System.***out***.println("\nPool Table:");

System.***out***.println("Pool No.\tLiteral Index Start");

**for** (**int** i = 0; i < poolTable.size(); i++) {

System.***out***.println((i + 1) + "\t\t" + poolTable.get(i));

}

sc.close();

}

}

**Intermediate Code Generation (Assembler) – Problems (10) to (12)**

**package** lpcc;

**import** java.util.\*;

**public** **class** IntermediateCodeGenerator {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

List<String> code = **new** ArrayList<>();

System.***out***.println("Enter assembly instructions line by line (type END to stop):");

**while** (**true**) {

String line = sc.nextLine().trim();

code.add(line);

**if** (line.equals("END")) **break**;

}

**int** LC = 0;

**for** (String line : code) {

String[] tokens = line.split("\\s+");

**if** (tokens[0].equals("START")) {

LC = Integer.*parseInt*(tokens[1]);

**continue**;

}

System.***out***.print(LC + ": ");

System.***out***.println(line);

LC++;

}

sc.close();

}

}

**Intermediate Code Generation (Macro) – Problems (13) to (16)**

**package** lpcc;

**import** java.util.\*;

**public** **class** MacroIntermediateCode{

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

List<String> code = **new** ArrayList<>();

**boolean** inMacro = **false**;

System.***out***.println("Enter macro code line by line (type END to stop):");

**while** (**true**) {

String line = sc.nextLine().trim();

code.add(line);

**if** (line.equals("END")) **break**;

}

System.***out***.println("\nIntermediate Code:");

**for** (String line : code) {

**if** (line.startsWith("MACRO")) {

inMacro = **true**;

System.***out***.println("[MACRO DEF] " + line);

} **else** **if** (line.equals("MEND")) {

inMacro = **false**;

System.***out***.println("[MACRO END]");

} **else** **if** (inMacro) {

System.***out***.println("[M] " + line);

} **else** {

System.***out***.println("[IC] " + line);

}

}

sc.close();

}

}

**MDT (Macro Definition Table) – Problems (17), (18)**

**package** lpcc;

**import** java.util.\*;

**public** **class** MDTGenerator {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

List<String> MDT = **new** ArrayList<>();

**int** index = 1;

System.***out***.println("Enter macro definitions (type END to stop):");

**boolean** insideMacro = **false**;

**while** (**true**) {

String line = sc.nextLine().trim();

**if** (line.equals("END")) **break**;

**if** (line.startsWith("MACRO")) {

insideMacro = **true**;

MDT.add(index++ + "\t" + line);

} **else** **if** (line.equals("MEND")) {

MDT.add(index++ + "\tMEND");

insideMacro = **false**;

} **else** **if** (insideMacro) {

MDT.add(index++ + "\t" + line);

}

}

System.***out***.println("\nMacro Definition Table (MDT):");

**for** (String entry : MDT) {

System.***out***.println(entry);

}

sc.close();

}

}

**MNT (Macro Name Table) – Problems (19), (20)**

package lpcc;

import java.util.\*;

public class MNTGenerator {

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

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

int index = 1;

System.*out*.println("Enter macro definitions (type END to stop):");

while (true) {

String line = sc.nextLine().trim();

if (line.equals("END")) break;

if (line.startsWith("MACRO")) {

String[] parts = line.split("\\s+");

if (parts.length > 1) {

MNT.add(index++ + "\t" + parts[1]);

}

}

}

System.*out*.println("\nMacro Name Table (MNT):");

System.*out*.println("Index\tMacroName");

for (String entry : MNT) {

System.*out*.println(entry);

}

sc.close();

}

}

**TAC for Expressions – Problems (41) to (48)**

**package** lpcc;

**import** java.util.\*;

**public** **class** TACGenerator {

**static** **int** *tempCount* = 1;

**static** List<String> *tac* = **new** ArrayList<>();

**static** String newTemp() {

**return** "t" + (*tempCount*++);

}

**static** String generate(String expr) {

Stack<String> operands = **new** Stack<>();

Stack<Character> operators = **new** Stack<>();

**for** (**int** i = 0; i < expr.length(); i++) {

**char** ch = expr.charAt(i);

**if** (ch == ' ') **continue**;

**if** (Character.*isLetterOrDigit*(ch)) {

operands.push(String.*valueOf*(ch));

} **else** **if** (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^') {

**while** (!operators.isEmpty() && *precedence*(operators.peek()) >= *precedence*(ch)) {

*makeTAC*(operators.pop(), operands);

}

operators.push(ch);

} **else** **if** (ch == '(') {

operators.push(ch);

} **else** **if** (ch == ')') {

**while** (!operators.isEmpty() && operators.peek() != '(') {

*makeTAC*(operators.pop(), operands);

}

operators.pop(); // remove '('

}

}

**while** (!operators.isEmpty()) {

*makeTAC*(operators.pop(), operands);

}

**return** operands.pop();

}

**static** **void** makeTAC(**char** op, Stack<String> operands) {

String b = operands.pop();

String a = operands.pop();

String t = *newTemp*();

*tac*.add(t + " = " + a + " " + op + " " + b);

operands.push(t);

}

**static** **int** precedence(**char** op) {

**switch** (op) {

**case** '^': **return** 3;

**case** '\*': **case** '/': **return** 2;

**case** '+': **case** '-': **return** 1;

}

**return** 0;

}

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.print("Enter LHS variable: ");

String lhs = sc.nextLine();

System.***out***.print("Enter arithmetic expression: ");

String expr = sc.nextLine().replace(" ", ""); // remove spaces

String resultTemp = *generate*(expr);

*tac*.add(lhs + " = " + resultTemp);

System.***out***.println("\nThree Address Code:");

**for** (String line : *tac*) {

System.***out***.println(line);

}

sc.close();

}

}