*Miraç Cumbur B1705.010031 Compiler*

*public class* main {  
 *public static void* main(String[] args) {  
 ProgramText programText = *new* ProgramText();  
 OurScanner scanner = *new* OurScanner(programText);  
 Token token = *new* Token(programText);  
 Parser parser = *new* Parser(scanner, programText, token);  
 parser.parse();  
 }  
}

*public enum* TokenType {  
 LEFT\_CURLY("{"), RIGHT\_CURLY("}"), LEFT\_PAR("("), RIGHT\_PAR(")"),  
 EQUAL("="), SEMI\_COLON(";"), LESS\_THAN("<"),GRATER\_THAN(">"),  
 MINUS("-"), MULTIPLY("\*"), DIVIDE("/"), PLUS("+"),NOT("!"),  
  
 WHILE("while"), IF("if"), OUT("out"), IN("in"),  
 IDENITIFIER, NUMBER, END\_OF\_FILE;  
  
 *public* String getText() {  
 *return* text;  
 }  
  
 *private final* String text;  
  
 TokenType(String text) {  
 *this*.text = text;  
 }  
  
 TokenType() {  
 *this*.text = *this*.toString();  
 }  
}

*public enum* BooleanOperationType {  
 EQUAL\_AND\_EQUAL("=="),NOT\_EQUAL("!="),LESS\_AND\_EQUAL("<="),GRATER\_AND\_EQUAL(">="),  
 LESS("<"),GRATER(">");  
  
 *public* String getText() {  
 *return* text;  
 }  
  
 *private final* String text;  
  
 BooleanOperationType(String text) {  
 *this*.text = text;  
 }  
  
 BooleanOperationType() {  
 *this*.text = *this*.toString();  
 }  
}

*public class* Token {  
  
 *public* TokenType type;  
 *public* String text;  
 *private final* ProgramText source;  
   
 Token(ProgramText source){  
 *this*.source = source;  
 }  
 *public* TokenType getTokenType() {  
 *return* type;  
 }  
 *public* String getText() {  
 *return* text;  
 }  
}

*public class* EOFToken *extends* Token {  
  
 EOFToken(ProgramText source) {  
 *super*(source);  
 type = TokenType.END\_OF\_FILE;  
  
 }  
  
}

*public class* IdentifierToken *extends* Token{  
  
 IdentifierToken(ProgramText source,String text, TokenType type) {  
 *super*(source);  
 *this*.text=text;  
 *this*.type=type;  
  
  
 }  
  
   
  
}

*public class* KeywordToken *extends* Token {  
 KeywordToken(ProgramText source,String text, TokenType type) {  
 *super*(source);  
 *this*.text=text;  
 *this*.type=type;  
 }  
}

*public class* NumberToken *extends* Token{  
   
 NumberToken(ProgramText source,String text,TokenType type) {  
 *super*(source);  
 *this*.text=text;  
 *this*.type=type;  
 }  
}

*public class* SpecialToken *extends* Token {  
  
 SpecialToken(ProgramText source, String text, TokenType Specialtype) {  
 *super*(source);  
 *this*.text = text;  
 *this*.type = Specialtype;  
 *// TODO Auto-generated constructor stub* }  
  
}

*//is responsible for scanning for tokens (it will return tokens)  
//to the parser.  
public class* OurScanner {  
 *private final* ProgramText source;  
 *public* String string = "";  
 *public char* chNext, chCur;  
  
 OurScanner(ProgramText source) {  
 *this*.source = source;  
 }  
  
 *boolean* isSpecial(*char* chNext) {  
 *boolean* control = *false*;  
 *if* (!Character.isWhitespace(chNext)) {  
 *for* (TokenType type : TokenType.values()) {  
 *if* (String.valueOf(chNext).equals(type.getText())) {  
 control = *true*;  
 *break*;  
 }  
 }  
 }  
 *return* control;  
 }  
  
 *//Scanner will ask the Source for characters and one a sequence of  
 //characters form a token it will return immediately.  
 //Scanner needs to know some of rules (for example, what constitutes  
 //a number, what constitutes an identifier and so forth)* Token nextToken() {  
 Token token;  
  
 chCur = source.curChar();  
 chNext = source.nextChar();  
 *while* (Character.isWhitespace(chCur)) {  
 chCur = source.curChar();  
 chNext = source.nextChar();  
 }  
 *if* (!Character.isWhitespace(chCur)) {  
 *for* (TokenType type : TokenType.values()) {  
 *if* (String.valueOf(chCur).equals(type.getText())) {  
 token = *new* SpecialToken(source, String.valueOf(chCur), type);  
 *return* token;  
 }  
 }  
 *if* (Character.isDigit(chCur)) {  
 *//number token  
 //System.out.println(chCur);* string += chCur;  
 *if* (isSpecial(chNext)) {  
 token = *new* NumberToken(source, string, TokenType.NUMBER);  
 string = "";  
 *return* token;  
 }  
  
 } *else if* (Character.isLetter(chCur)) {  
 *//identifier token* string += chCur;  
 *if* (isSpecial(chNext)) {  
 *if* (string.equals(TokenType.WHILE.getText())) {  
 *//System.out.println(TokenType.WHILE.getText());* token = *new* KeywordToken(source, string, TokenType.WHILE);  
 string = "";  
 *return* token;  
 } *else if* (string.equals(TokenType.IF.getText())) {  
 token = *new* KeywordToken(source, string, TokenType.IF);  
 string = "";  
 *return* token;  
 } *else if* (string.equals((TokenType.OUT.getText()))) {  
 token = *new* KeywordToken(source, string, TokenType.OUT);  
 string = "";  
 *return* token;  
 } *else if* (string.equals((TokenType.IN.getText()))) {  
 token = *new* KeywordToken(source, string, TokenType.OUT);  
 string = "";  
 *return* token;  
 } *else* {  
 token = *new* IdentifierToken(source, string, TokenType.IDENITIFIER);  
 string = "";  
 *return* token;  
 }  
  
  
 }  
  
 } *else* {  
 token = *new* EOFToken(source);  
 *return* token;  
 }  
 }  
  
 *return null*;  
  
  
 }  
  
  
}

*import* java.io.IOException;  
*import* java.nio.file.Files;  
*import* java.nio.file.Paths;  
  
*//the purpose of the ProgramText class is to abstract away  
//from where the program is coming. ProgramText provides a  
//single character to the Scanner class when asked for.  
//it reads the program (from a file or as String) line by line  
//from top to bottom  
public class* ProgramText {  
  
 *//private BufferedReader reader;  
 public* String progText;  
 *private int* curPos, rez = 0;  
 *public static char* EOF = '₺';  
  
 ProgramText() {  
  
 curPos = -1;  
  
 *try* {  
 progText = readWholeProgram();  
  
 } *catch* (IOException e) {  
 *// TODO Auto-generated catch block* e.printStackTrace();  
 }  
  
 }  
  
 *private* String readWholeProgram() *throws* IOException {  
 *return new* String(Files.readAllBytes(Paths.get("program2.txt")));  
  
 }  
  
 *char* curChar() {  
 *if* (curPos == -1)  
 curPos++;  
  
 *if* (curPos == progText.length())  
 *return* EOF;  
 *if* (rez <= progText.length()) {  
 *return* progText.charAt(curPos);  
 }  
 *return* EOF;  
 }  
  
 *char* nextChar() {  
 curPos++;  
 rez = curPos;  
 *if* (rez == progText.length())  
 *return* EOF;  
  
 *for* (*int* i = rez; i < progText.length(); i++) {  
 *if* (Character.isWhitespace(progText.charAt(rez))) {  
 rez++;  
 *if* (rez == progText.length()) {  
 *return* EOF;  
 }  
  
 }  
 }  
  
 *if* (rez == progText.length()) {  
 *return* EOF;  
 }  
 *if* (rez <= progText.length()) {  
 *return* progText.charAt(rez);  
 }  
  
  
 *return* EOF;  
  
 }  
  
  
}

*import* javax.script.ScriptEngine;  
*import* javax.script.ScriptEngineManager;  
*import* javax.script.ScriptException;  
*import* java.util.\*;  
*import* java.util.stream.Collectors;  
  
*public class* Parser {  
  
 *private final* OurScanner scanner;  
 *public* ProgramText programText;  
 *private* Token curToken,nextToken;  
 *private int* rightCurly = 0, leftCurly = 0;  
 *public* Node node = *new* Node("Program"),nodeEXP=*new* Node("EXP");  
 *public* Node curNode;  
 *public* String curParentLeftChild;  
 *public* ArrayList<String> arrayListControl = *new* ArrayList<String>();  
 *public* HashMap<String,String> map = *new* HashMap<String,String>();  
 *public* Scanner scan=*new* Scanner(System.in);  
  
 *// eğer parantez kapanırsa current nodeyi parent yap* Parser(OurScanner scanner, ProgramText programText, Token token) {  
  
 *this*.scanner = scanner;  
 *this*.programText = programText;  
 *this*.curToken = token;  
 }  
  
 *void* parse() {  
 curToken = scanner.nextToken();  
 *while* (!(curToken *instanceof* EOFToken)) {  
 *if* (!(curToken *instanceof* EOFToken)) {  
  
 *if* (curToken != *null*) {  
 *//System.out.println(curToken.getText());* curNode=node;  
 *if*(arrayListControl.size()>0){  
 *for*(*int* i=0;i<arrayListControl.size();i++){  
 *for*(Node s : curNode.children){  
 *if*(s.data.equals(arrayListControl.get(i))){  
 curNode=s.children.get(1);  
  
 }  
 }  
  
 }  
 }  
 *//System.out.printf("Type: %s, text: %s\n", curToken.getTokenType(), curToken.getText())* S();  
 }  
 }  
 curToken = scanner.nextToken();  
 }  
 *if* (!curlyController()) {  
 *//System.out.println("1Something is wrong.. " + curToken.getTokenType());* System.exit(0);  
 }  
 Treversal treversal = *new* Treversal();  
 ArrayList<String> treePreOrder=treversal.preorder(node);  
 ArrayList<String> treePostOrder=treversal.postorder(node);  
 System.out.println("Preorder: "+treePreOrder);  
 System.out.println("Postorder: "+treePostOrder);  
 eval(treePreOrder);  
  
  
   
  
 }  
  
 *void* S() {  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 *return*;  
 } *else* {  
 System.out.println("2Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
  
 S1();  
 }  
 *// bi değişken belirleriz adı while olursa while içinde if olursa if içinde  
 //bu sayede onu kontrol ederken ne içinde anlarız  
 //hatta bunu arraylist yaparız sonuncuyu sileriz en sonuncu içinde olduğumuz olur  
 //ya da map te tutarız konumunu while mı if mi olduğunu daha iyi olur  
  
 void* S1() {  
 curParentLeftChild=curToken.getText();  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 System.out.println("end of file");  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("3Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 *return*;  
 } *else if* (curToken.getTokenType().equals(TokenType.RIGHT\_CURLY)) {  
 arrayListControl.remove(arrayListControl.size()-1);  
 rightCurly++;  
 } *else if* (curToken.getTokenType().equals(TokenType.LEFT\_CURLY)) {  
 leftCurly++;  
 }  
  
 *//WHILE  
  
 else if* (curToken.getTokenType().equals(TokenType.WHILE)) {  
 childAdder(curNode,curToken.getText());  
 *for*(Node s : curNode.children){  
 *if*(s.data.equals(curToken.getText())){  
 curNode=s;  
 }  
 }  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("4Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else if* (curToken.getTokenType().equals(TokenType.LEFT\_PAR)) {  
 *//System.out.println(curNode.children.stream().filter(f-> f.data.equals(curToken.getText())).collect(Collectors.toList()));* String s=Boolean();  
 childAdder(curNode,s);  
 childAdder(curNode,"body");  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("5Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.RIGHT\_PAR)) {  
  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("6Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.LEFT\_CURLY)) {  
 arrayListControl.add("while");  
 leftCurly++;  
 } *else* {  
 System.out.println("7Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("8Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("9Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
  
 *//IF  
  
 else if* (curToken.getTokenType().equals(TokenType.IF)) {  
 childAdder(curNode,curToken.getText());  
 *for*(Node s : curNode.children){  
 *if*(s.data.equals(curToken.getText())){  
 curNode=s;  
 }  
 }  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("10Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else if* (curToken.getTokenType().equals(TokenType.LEFT\_PAR)) {  
 String s=Boolean();  
 childAdder(curNode,s);  
 childAdder(curNode,"body");  
 *//Exp();* curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("11Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.RIGHT\_PAR)) {  
  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("12Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.LEFT\_CURLY)) {  
 arrayListControl.add("if");  
 leftCurly++;  
 } *else* {  
 System.out.println("13Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("14Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("15Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
  
 *//IDENITIFIER  
  
 else if* (curToken.getTokenType().equals(TokenType.IDENITIFIER)) {  
 curParentLeftChild=curToken.getText();  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("16Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.EQUAL)) {  
 childAdder(curNode,TokenType.EQUAL.getText());  
 *for*(Node s : curNode.children){  
 *if*(s.data.equals(TokenType.EQUAL.getText())){  
 curNode=s;  
 }  
 }  
 childAdder(curNode,curParentLeftChild);  
 String s=Exp();  
 childAdder(curNode,s);  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("17Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.SEMI\_COLON)) {  
  
 } *else* {  
 System.out.println("18Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("19Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
  
 *//OUT  
  
 else if* (curToken.getTokenType().equals(TokenType.OUT)) {  
 childAdder(curNode,curToken.getText());  
 *for*(Node s : curNode.children){  
 *if*(s.data.equals(curToken.getText())){  
 curNode=s;  
 }  
 }  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("20Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else if* (curToken.getTokenType().equals(TokenType.LEFT\_PAR)) {  
 String s=Exp();  
 childAdder(curNode,s);  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("21Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.RIGHT\_PAR)) {  
  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("22Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.SEMI\_COLON)) {  
  
 } *else* {  
 System.out.println("25Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("26Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("27Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
  
 *//IN  
  
 else if* (curToken.getTokenType().equals(TokenType.IN)) {  
 childAdder(curNode,curToken.getText());  
 *for*(Node s : curNode.children){  
 *if*(s.data.equals(curToken.getText())){  
 curNode=s;  
 }  
 }  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("20Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else if* (curToken.getTokenType().equals(TokenType.LEFT\_PAR)) {  
 String s=Exp();  
 childAdder(curNode,s);  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("21Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.RIGHT\_PAR)) {  
  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("22Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (curToken.getTokenType().equals(TokenType.SEMI\_COLON)) {  
  
 } *else* {  
 System.out.println("25Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("26Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else* {  
 System.out.println("27Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
  
  
 }  
  
 *//EXPRESSION* String Exp() {  
 String rezerve = "";  
 *boolean* control = *true*;  
 *while* (control) {  
 *if* (String.valueOf(scanner.chNext).equals(TokenType.RIGHT\_PAR.getText()) ||  
 String.valueOf(scanner.chNext).equals(TokenType.SEMI\_COLON.getText())) {  
  
 control = *false*;  
  
 }*else*{  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
  
 *if* (curToken.getTokenType().equals(TokenType.END\_OF\_FILE)) {  
 *if* (curlyController()) {  
 System.exit(0);  
 } *else* {  
 System.out.println("36Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 System.exit(0);  
 } *else if* (!(String.valueOf(scanner.chNext).equals(TokenType.RIGHT\_PAR.getText()) ||  
 String.valueOf(scanner.chNext).equals(TokenType.SEMI\_COLON.getText()))) {  
 rezerve += curToken.getText();  
 } *else if* ((String.valueOf(scanner.chNext).equals(TokenType.RIGHT\_PAR.getText()) ||  
 String.valueOf(scanner.chNext).equals(TokenType.SEMI\_COLON.getText()))) {  
 rezerve += curToken.getText();  
 *//System.out.println(rezerve);  
 break*;  
 } *else* {  
 System.out.println(rezerve + " 37Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
  
 }  
  
 }  
 *return* rezerve;  
 }  
  
  
  
  
 *//curlyController  
  
 boolean* curlyController() {  
 *boolean* control = *false*;  
 *if* (rightCurly == leftCurly) {  
 control = *true*;  
 }  
 *return* control;  
 }  
  
 *void* childAdder (Node node,String value){  
 node.addChild(*new* Node(value));  
 }  
 *void* ExpTail(){  
 }  
  
 *void* Term() {  
 }  
  
 *void* TermTail() {  
 }  
 *void* Factor() {  
 }  
  
 *void* FactorTail() {  
 }  
  
 *void* Id() {  
 }  
  
 *void* Char() {  
 }  
  
 *void* Num() {  
 }  
  
 String Boolean() {  
 *boolean* control = *true*,control2=*true*;  
 String booleanValue = "",rezerve="";  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 *if* (!(curToken.getTokenType().equals(TokenType.NUMBER) || curToken.getTokenType().equals(TokenType.IDENITIFIER))) {  
  
 System.out.println(" 38Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
  
 } *else* {  
 *if* (curToken.getTokenType().equals(TokenType.NUMBER) || curToken.getTokenType().equals(TokenType.IDENITIFIER)) {  
 rezerve+=curToken.getText();  
 curToken = scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 nextToken=scanner.nextToken();  
 *while* (nextToken == *null*) {  
 nextToken = scanner.nextToken();  
 }  
 *if* (nextToken.getTokenType().equals(TokenType.IDENITIFIER)|| nextToken.getTokenType().equals(TokenType.NUMBER)) {  
 booleanValue+=curToken.getText();  
 rezerve=rezerve+curToken.getText()+nextToken.getText();  
 *for* (BooleanOperationType type : BooleanOperationType.values()) {  
 *if* (booleanValue.equals(type.getText())) {  
 *//System.out.println(rezerve+" "+type.getText()+" "+type);* control2=*false*;  
 }  
 }  
 *if*(control2){  
 System.out.println(" 39Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 } *else if*(nextToken.getTokenType().equals(TokenType.WHILE)|| nextToken.getTokenType().equals(TokenType.IF)||  
 nextToken.getTokenType().equals(TokenType.OUT)|| nextToken.getTokenType().equals(TokenType.IN)){  
 System.out.println(" 40Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
  
 } *else* {  
 booleanValue=booleanValue+curToken.getText()+nextToken.getText();  
 rezerve=rezerve+curToken.getText()+nextToken.getText();  
 *for* (BooleanOperationType type : BooleanOperationType.values()) {  
 *if* (booleanValue.equals(type.getText())) {  
 curToken= scanner.nextToken();  
 *while* (curToken == *null*) {  
 curToken = scanner.nextToken();  
 }  
 rezerve+=curToken.getText();  
 *//System.out.println(rezerve+" "+type.getText()+" "+type);* control2=*false*;  
 }  
 }  
 *if*(control2){  
 System.out.println(" 41Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
 } *else* {  
 System.out.println(" 42Something is wrong.. " + curToken.getTokenType());  
 System.exit(0);  
 }  
 }  
  
 *return* rezerve;  
 }  
  
 *void* eval(ArrayList<String> tree){  
 *for*(*int* i=0;i<tree.size();i++){  
 *//System.out.println(tree.get(i));  
 if*(tree.get(i).equals("Program")){  
  
 }*else if*(tree.get(i).equals(TokenType.EQUAL.getText())){  
 i++;  
 String rez,rez1;  
 rez=tree.get(i);  
 i++;  
 rez1=tree.get(i);  
 *//System.out.println(rez+" "+rez1);* expressionSolver(rez1);  
 }*else if*(tree.get(i).equals(TokenType.WHILE.getText())){  
 i++;  
 booleanSolver(tree.get(i));  
 }*else if*(tree.get(i).equals(TokenType.IF.getText())){  
 i++;  
 booleanSolver(tree.get(i));  
 }*else if*(tree.get(i).equals(TokenType.IN.getText())){  
 i++;  
 String valueName=tree.get(i);  
 String value=scan.nextLine();  
 map.put(valueName,value);  
 }*else if*(tree.get(i).equals(TokenType.OUT.getText())){  
 i++;  
 System.out.println(map.get(tree.get(i)));  
  
 }  
 }  
  
 }  
 String expressionSolver(String expression){  
 ScriptEngineManager mgr = *new* ScriptEngineManager();  
 ScriptEngine engine = mgr.getEngineByName("JavaScript");  
 String rez="";  
 String rezMain="";  
 *for* (*int* i=0;i<expression.length();i++) {  
 *//System.out.println(foo.charAt(i));  
 if*(Character.isWhitespace(expression.charAt(i))){  
  
 }  
 *else if*(Character.isLetter(expression.charAt(i))||Character.isDigit(expression.charAt(i))){  
 rez+=expression.charAt(i);  
 rezMain+=expression.charAt(i);  
 }  
 *else if*(String.valueOf(expression.charAt(i)).equals("+")||  
 String.valueOf(expression.charAt(i)).equals("-")||  
 String.valueOf(expression.charAt(i)).equals("\*")||  
 String.valueOf(expression.charAt(i)).equals("/")){  
 rezMain+=expression.charAt(i);  
 *//System.out.println(rez);  
 boolean* ctrl=*false*;  
 *for*(*int* l=0;l<rez.length();l++){  
 *if*(Character.isLetter(rez.charAt(l))){  
 ctrl=*true*;  
 }  
 }

*if*(ctrl){  
 String value=map.get(rez);*/\*  
 if(value.isEmpty()){  
 System.out.println("error");  
 System.exit(0);  
 }\*/* rezMain+=value;  
 }*else*{  
 rezMain+=rez;  
 }  
 rez="";  
 */\*  
 for (Map.Entry<String,String> entry: map.entrySet()){  
 String key = entry.getKey();  
 if(key.equals(rez)){  
  
 rez+=map.get(key);  
 }  
 else{  
 System.out.println("error");  
 }  
 }  
 \*/* }  
  
 }*//System.out.println(rezMain);  
 /\*try {  
  
 System.out.println(engine.eval(rezMain));  
 } catch (ScriptException e) {  
 e.printStackTrace();  
 }\*/  
 return null*;  
 }  
  
 Boolean booleanSolver(String expression){  
 *return null*;  
 }  
}

*import* java.util.ArrayList;  
*import* java.util.List;  
  
*public class* Node {  
 *public* String data; *//data for storage  
 public* List<Node> children;*//array will keep children  
 public* Node parent;*//parent to start the tree  
  
 public* Node(String data) {  
 children = *new* ArrayList<>();  
 *this*.data = data;  
 }  
  
 *public* Node addChild(Node node) {  
 children.add(node);  
 node.parent = *this*;  
 *return this*;  
 }  
}

*public class* Pair {  
 *public* Node node;  
 *public int* childrenIndex;  
  
 *public* Pair(Node \_node, *int* \_childrenIndex) {  
 node = \_node;  
 childrenIndex = \_childrenIndex;  
 }  
}

*import* java.util.ArrayList;  
*import* java.util.Stack;  
  
*public class* Treversal {  
  
  
 *int* currentRootIndex = 0;  
 Stack<Pair> stack = *new* Stack<Pair>();  
 ArrayList<String> postorderTraversal = *new* ArrayList<String>();  
 ArrayList<String> preorderTraversal = *new* ArrayList<String>();  
 *// Function to perform iterative postorder traversal  
 public* ArrayList<String> postorder(Node root) {  
 *while* (root != *null* || !stack.isEmpty()) {  
 *if* (root != *null*) {  
  
 *// Push the root and it's index  
 // into the stack* stack.push(*new* Pair(root, currentRootIndex));  
 currentRootIndex = 0;  
  
 *// If root don't have any children's that  
 // means we are already at the left most  
 // node, so we will mark root as null  
 if* (root.children.size() >= 1) {  
 root = root.children.get(0);  
 } *else* {  
 root = *null*;  
 }  
 *continue*;  
 }  
  
 *// We will pop the top of the stack and  
 // add it to our answer* Pair temp = stack.pop();  
 postorderTraversal.add(temp.node.data);  
  
 *// Repeatedly we will the pop all the  
 // elements from the stack till popped  
 // element is last children of top of  
 // the stack  
 while* (!stack.isEmpty() && temp.childrenIndex ==  
 stack.peek().node.children.size() - 1) {  
 temp = stack.pop();  
  
 postorderTraversal.add(temp.node.data);  
 }  
  
 *// If stack is not empty, then simply assign  
 // the root to the next children of top  
 // of stack's node  
 if* (!stack.isEmpty()) {  
 root = stack.peek().node.children.get(  
 temp.childrenIndex + 1);  
 currentRootIndex = temp.childrenIndex + 1;  
 }

}  
  
 *return* postorderTraversal;  
 }

*public* ArrayList<String> preorder(Node root) {  
 Stack<Node> stackNode = *new* Stack<>();  
 stackNode.push(root);  
 *while* (!stackNode.isEmpty()) {  
 *// Store the current node and pop  
 // it from the stack* Node curr = stackNode.pop();  
  
 *// Current node has been travarsed  
 if* (curr != *null*)  
 {  
 *//System.out.print(curr.data + " ");* preorderTraversal.add(curr.data);  
 *// Store all the childrent of  
 // current node from right to left.  
 for*(*int* i = curr.children.size() - 1; i >= 0; i--)  
 {  
 stackNode.add(curr.children.get(i));  
 }

}  
 }

*return* preorderTraversal;  
 }

}

Outputs;

Only dft outputs are available in these outputs. I tried to print out the program's output. I've written some codes. But I didn't have enough time to solve the problems. I couldn't get the rest of my code. For this reason, I could not add outputs that show the results of the program. In the parse class of the code, you can see the codes I wrote for this part. In readme.txt file, i explained the code.

Program1.txt

x = 10;  
sum=0;  
rez=13;  
while(x<20){  
sum=sum+x;  
}  
x = x + 1 ;  
sum=sum+x;  
out(sum);

metin, ekran, iç mekan, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Other program example in other page.

Program2.txt

x = 23 + 12 ;  
in(y);  
while ( x<100 ) {  
 out (x);  
  
 x = x + 1 ;  
 }  
  
if ( y == 10) {  
 out (y);  
 }

metin, ekran, ekran görüntüsü, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

Other program example in other page.

Program3.txt

x = 23+12 ;  
  
while ( x<100 ) {  
  
 out (x);  
  
 x = x + 1 ;  
  
 if ( x == 20) {  
 in (max);  
 }  
  
 if ( x == 10) {  
 out (min);  
 }  
  
}  
  
if ( x == 10) {  
 in (true);  
}

metin, ekran, ekran görüntüsü, dizüstü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Other program example in other page.

Program4.txt

x=7;  
y=10+2\*6;  
while(x<y){  
out(x);  
x=x+1;  
}

metin, ekran, ekran görüntüsü, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu