НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

##### Лабораторна робота №2

**«Розробка генератора коду»**

***з дисципліни "Інженерія програмного забезпечення1. Основи проектування трансляторів"***

**Варіант № 13**

Виконав: Микитенко С.С.

#### Група: КB-51

Номер залікової книжки: КВ-5113

Оцінка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 семестр 2017/2018

***Варіант 13***

1. <signal-program> --> <program>

2. <program> --> PROCEDURE <procedure-identifier> <parameters-list> ; <block> ;

3. <block> --> <declarations> BEGIN <statementslist> END

4. <declarations> --> <procedure-declarations>

5. <procedure-declarations> --> <procedure> <procedure- declarations> |<empty>

6. <procedure> --> PROCEDURE <procedureidentifier><parameters-list> ;

7. <parameters-list> --> ( <variable-identifier> <identifiers-list> ) | <empty>

8. <identifiers-list> --> , <variable-identifier> <identifiers-list> | <empty>

9. <statements-list> --> <statement> <statementslist> | <empty>

10. <statement> --> <procedure-identifier> <actualarguments> ; | RETURN ;

11. <actual-arguments> --> ( <unsigned-integer><actual- arguments-list> ) | <empty>

12. <actual-arguments-list> --> , <unsigned-integer> <actual-arguments-list> | <empty>

13. <variable-identifier> --> <identifier>

14. <procedure-identifier> --> <identifier>

15. <identifier> --> <letter><string>

16. <string> --> <letter><string> | <digit><string> | <empty>

17. <unsigned-integer> --> <digit><digits-string>

18. <digits-string> --> <digit><digits-string> | <empty>

19. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

20. <letter> --> A | B | C | D | ... | Z

**Лістинг програми**

**public** **class** TreeNode {

ArrayList<TreeNode> kids=

**new** ArrayList<TreeNode>();

**public** **int** value;

TreeNode(**int** \_code){

value = \_code;

}

TreeNode(){

value = -1;

}

**void** add(TreeNode kid) {

kids.add(kid);

}

**void** add(**int** code) {

TreeNode kid = **new** TreeNode(code);

kids.add(kid);

}

TreeNode get(**int** id) {

**if** (kids.size() < id) {

TreeNode kid = **new** TreeNode();

System.***out***.println("err: empty Node");

**return** kid;

}

**return** kids.get(id);

}

}

**public** **class** Lexeme {

**public** **int** code = -1;

**public** **int** row = -1;

**public** **int** pos = -1;

**public** **void** set(**int** \_c, **int** \_r, **int** \_p) {

code = \_c;

row = \_r;

pos = \_p;

}

**public** Lexeme() {

}

**public** Lexeme(**int** \_c, **int** \_r, **int** \_p) {

code = \_c;

row = \_r;

pos = \_p;

}

}

**public** **class** Compiler {

**public** String compile(String fileName) {

String output = "";

Tables table = **new** Tables();

Lexer lexer = **new** Lexer();

lexer.analize(table, fileName);

table.printTables();

**if** (table.errors.size()==0) {

Parser parser = **new** Parser();

parser.analize(table);

table.printTree();

**if** (table.errors.size()==0) {

Coder coder = **new** Coder();

output = coder.generate(table);

System.***out***.print("\nCode:\n" + output);

}

}

table.printErrors();

**if** (table.errors.size()==0) {

**return** output;

} **else** {

**return** ";Error\n";

}

}

**public** Compiler() {

}

}

**public** **class** Main {

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

Tests test = **new** Tests();

System.***out***.println("\ntest1");

test.coder("test1.txt");

System.***out***.println("\ntest2");

test.coder("test2.txt");

System.***out***.println("\ntest3");

test.coder("test3.txt");

}

}

**public** **class** Tables {

//lexer

ArrayList<String> keys = **new** ArrayList<String>(4);// 400 - 499

ArrayList<String> consts = **new** ArrayList<String>();// 500 - 999

ArrayList<String> ids = **new** ArrayList<String>();// 1000 - ...

ArrayList<Integer> attrs = **new** ArrayList<Integer>(256);

ArrayList<Lexeme> lexemes = **new** ArrayList<Lexeme>();

ArrayList<String> errors = **new** ArrayList<String>();

//parser

TreeNode treeRoot = **new** TreeNode();

**static** **final** **int** ***signalProgram*** = -2;

**static** **final** **int** ***program*** = -3;

**static** **final** **int** ***block*** = -4;

**static** **final** **int** ***declarations*** = -5;

**static** **final** **int** ***procedureDeclarations*** = -6;

**static** **final** **int** ***empty*** = -7;

**static** **final** **int** ***procedure*** = -8;

**static** **final** **int** ***paramatersList*** = -9;

**static** **final** **int** ***identifiersList*** = -10;

**static** **final** **int** ***statementsList*** = -11;

**static** **final** **int** ***statement*** = -12;

**static** **final** **int** ***actualArguments*** = -13;

**static** **final** **int** ***actualArgumentsList*** = -14;

**static** **final** **int** ***unsignedInteger*** = -15;

**static** **final** **int** ***procedureIdentifier*** = -16;

**static** **final** **int** ***variableIdentifier*** = -17;

**static** **final** **int** ***identifier*** = -18;

//lexer

**public** String getTokken(**int** code){

**if** (code < 0) {

**return** "ERROR: Tokken doesn't exist";

}

**if** (code < 128) {

**return** String.*valueOf*((**char**)code);

}

**if** (code < 500) {

**return** keys.get(code-400);

}

**if** (code < 1000) {

**return** consts.get(code-500);

}

**return** ids.get(code-1000);

}

**public** **void** printTables(){

**int** i;

Lexeme lexeme;

System.***out***.println("\nLexemes:");

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

lexeme = lexemes.get(i);

System.***out***.format("%04d %04d %05d "+ getTokken(lexeme.code)+"%n", lexeme.row, lexeme.pos, lexeme.code);

}

System.***out***.println("\nKeys:");

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

System.***out***.format("%05d "+ keys.get(i)+"%n", i+400);

}

System.***out***.println("\nConsts:");

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

System.***out***.format("%05d "+ consts.get(i)+"%n", i+500);

}

System.***out***.println("\nIdentifier:");

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

System.***out***.format("%05d "+ ids.get(i)+"%n", i+1000);

}

}

**public** **void** printErrors(){

**int** i;

**if** (errors.size()>0) {

System.***out***.println("\nErrors:");

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

System.***out***.format(errors.get(i)+"%n");

}

} **else** {

System.***out***.println("\nNo errors");

}

}

**public** **int** getConstsId(String str){

**int** id = consts.indexOf(str);

**if** (id == -1) {

**if**(consts.size()<=500) {

id = consts.size();

consts.add(str);

} **else** {

**return** -1;

}

}

**return** id + 500;

}

**private** **int** getKeyId(String str) {

//return index of element in keys(+400) or ids(+1000)

**int** id = keys.indexOf(str);

**if** (id == -1) {

id = ids.indexOf(str);

**if** (id == -1) {

id = ids.size();

ids.add(str);

}

**return** id + 1000;

}

**return** id + 400;

}

**public** **void** addConst(String str, **int** j, **int** i) {

**int** constsId = getConstsId(str);

**if** (constsId == -1){

addError("Lexer: Error (line "+j+", column "+i +"): more than 500 consts");

} **else** {

addLexeme(constsId, j, i);

}

}

**public** **void** addIds(String str, **int** j, **int** i) {

addLexeme(getKeyId(str), j, i);

}

**public** **void** addError(String str) {

errors.add(str);

}

**public** **void** addLexeme(**int** code, **int** j, **int** i) {

Lexeme lexeme = **new** Lexeme(code, j, i);

lexemes.add(lexeme);

}

**public** **int** getAttrs(**int** digit) {

**if** ((digit >= 0)&&(digit <=127)){

**return** attrs.get(digit);

}

**return** 6;

}

**public** Tables() {

**for** (**int** i = 0; i <= 127; i++) {

attrs.add(6);

}

//white space

attrs.set(8, 0);//backspace

attrs.set(9, 0);//tab

attrs.set(10, 0);//next line

attrs.set(11, 0);//tab

attrs.set(13, 0);//carriage return

attrs.set(32, 0);//space

**for** (**int** i = 48; i <= 57; i++) {

attrs.set(i, 1);//digits

}

**for** (**int** i = 65; i <= 90; i++) {

attrs.set(i, 2);//letters

}

attrs.set(59, 3);//;

attrs.set(40, 5);//( ==> (\*

attrs.set(41, 3);//)

attrs.set(44, 3);//,

keys.add("PROCEDURE");//0

keys.add("BEGIN");//1 401

keys.add("END");//2

keys.add("RETURN");//3

}

//parser

**public** **void** addParserError(String str, **int** index) {

**if** (index<lexemes.size()){

Lexeme buff = lexemes.get(index);

addError("Parser: Error (line "+buff.row+", column "+buff.pos +"): expected {"+str+"} but found {"+getTokken(buff.code)+"}");

} **else** {

addError("Parser: Error (on eof): expected {#} but found EOF");

}

}

String getNodeName(**int** id) {

**switch**(id) {

**case** -1:

**return** "!<Parser Error: 'TreeNode.value=-1'>";

**case** ***signalProgram***:

**return** "<signal-program>";

**case** ***program***:

**return** "<program>";

**case** ***block***:

**return** "<block>";

**case** ***declarations***:

**return** "<declarations>";

**case** ***procedureDeclarations***:

**return** "<procedure-declarations>";

**case** ***empty***:

**return** "<empty>";

**case** ***procedure***:

**return** "<procedure>";

**case** ***paramatersList***:

**return** "<paramaters-list>";

**case** ***identifiersList***:

**return** "<identifiers-list>";

**case** ***statementsList***:

**return** "<statements-list>";

**case** ***statement***:

**return** "<statement>";

**case** ***actualArguments***:

**return** "<actual-arguments>";

**case** ***actualArgumentsList***:

**return** "<actual-arguments-list>";

**case** ***unsignedInteger***:

**return** "<unsigned-integer>";

**case** ***procedureIdentifier***:

**return** "<procedure-identifier>";

**case** ***variableIdentifier***:

**return** "<variable-identifier>";

**case** ***identifier***:

**return** "<identifier>";

}

**if** (id < lexemes.size()) {

**return** getTokken(lexemes.get(id).code);

} **else** {

**return** "!<Parser Error: lexemes id Out Of Range>";

}

}

**void** printTreeNode(**int** level, TreeNode root) {

String buff = "";

**for** (**int** i=0;i<level;i++) {

buff += '.';

}

**if** (root.value>-1) {

buff += lexemes.get(root.value).code +" "+ getNodeName(root.value) +"\n";

} **else** {

buff += getNodeName(root.value) +"\n";

}

System.***out***.print(buff);

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

printTreeNode(level + 2, root.get(i));

}

}

**void** printTree() {

System.***out***.println("\nTree:\n");

System.***out***.println(getNodeName(treeRoot.value));

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

printTreeNode(2, treeRoot.get(i));

}

}

**public** **void** addCodeError(Lexeme buff, String message) {

addError("Code Generator: Error (line "+buff.row+", column "+buff.pos +"): " + message);

}

}

**public** **class** Coder {

Tables table;

ArrayList <Integer> proceduresNamesCodeList = **new** ArrayList<Integer>();

ArrayList <Integer> procedureArgsNumList = **new** ArrayList<Integer>();

String beforeCall = "";

String afterCall = "";

ArrayList <Integer> procedureArgsList = **new** ArrayList<Integer>();

**private** Lexeme identifier(TreeNode root) {

**return** table.lexemes.get(root.get(0).value);

}

**private** Lexeme variableIdentifier(TreeNode root) {

**return** identifier(root.get(0));

}

**private** Lexeme procedureIdentifier(TreeNode root) {

**return** identifier(root.get(0));

}

**private** Integer unsignedInteger(TreeNode root) {

**return** root.get(0).value;

}

**private** **void** actualArgumentsList(TreeNode root, ArrayList<Integer> list) {

**if** (root.get(0).value == Tables.***empty***) {

**return**;

}

list.add(unsignedInteger(root.get(1)));

actualArgumentsList(root.get(2), list);

}

**private** **void** actualArguments(TreeNode root, ArrayList<Integer> list) {

**if** (root.get(0).value == Tables.***empty***) {

**return**;

}

list.add(unsignedInteger(root.get(1)));

actualArgumentsList(root.get(2), list);

}

**private** String statement(TreeNode root) {

**if** (root.get(0).value != Tables.***procedureIdentifier***) {

**return** "RET\n";

}

ArrayList<Integer> list = **new** ArrayList<Integer>();

actualArguments(root.get(1),list);

Lexeme id = procedureIdentifier(root.get(0));

**if** (proceduresNamesCodeList.indexOf(id.code) == -1) {

table.addCodeError(id, table.getTokken(id.code) + " - unknown procedure identifier");

**return** "NOP\n";

}

String code = "";

**int** j = 0;

**int** n = procedureArgsNumList.get(proceduresNamesCodeList.indexOf(id.code));

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

**if** (j < n) {

code +="MOV EAX, " + table.getTokken(table.lexemes.get(list.get(i)).code) + "\nPUSH EAX\n";

j++;

} **else** {

table.addCodeError(table.lexemes.get(list.get(i)), table.getTokken(id.code) + " expected "+n+" arguments, given "+list.size());

**return** "NOP\n";

}

}

**if** (n > list.size()) {

**if** (list.size() > 0) {

table.addCodeError(table.lexemes.get(list.get(list.size()-1)), table.getTokken(id.code) +" expected "+n+" arguments, given "+list.size());

**return** "NOP\n";

}

table.addCodeError(id, table.getTokken(id.code) +" expected "+n+" arguments, given 0");

**return** "NOP\n";

}

code = beforeCall + code + "CALL " + table.getTokken(id.code) + "\n" + afterCall;

**return** code;

}

**private** String statementsList(TreeNode root) {

**if** (root.get(0).value == Tables.***empty***) {

**return** "NOP\n";

}

**return** statement(root.get(0)) + statementsList(root.get(1));

}

**private** ArrayList<Lexeme> identifiersList(TreeNode root, ArrayList<Lexeme> list) {

**if** (root.get(0).value == Tables.***empty***) {

**return** list;

}

list.add(variableIdentifier(root.get(1)));

**return** identifiersList(root.get(2), list);

}

**private** ArrayList<Lexeme> parametersList(TreeNode root) {

ArrayList <Lexeme> list = **new** ArrayList<Lexeme>();

**if** (root.get(0).value == Tables.***empty***) {

**return** list;

}

list.add(variableIdentifier(root.get(1)));

**return** identifiersList(root.get(2), list);

}

**private** String procedure(TreeNode root) {

Lexeme buff = procedureIdentifier(root.get(1));

ArrayList <Lexeme> list = parametersList(root.get(2));

ArrayList <Integer> parameters = **new** ArrayList<Integer>(list.size());

**int** j = 0;

**int** id = proceduresNamesCodeList.indexOf(buff.code);

**if** (id != -1) {

table.addCodeError(buff, table.getTokken(buff.code) + " has already been used as procedure name");

**return** "";

}

**if** (procedureArgsList.indexOf(buff.code) != -1){

table.addCodeError(buff, table.getTokken(buff.code) + " has already been used as procedure argument");

**return** "";

}

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

id = parameters.indexOf(list.get(i).code);

**if** (id != -1) {

table.addCodeError(list.get(i), table.getTokken(list.get(i).code) + " has been used twice in this procedure");

j++;

**return** "";

} **else** {

parameters.add(list.get(i).code);

j++;

}

}

proceduresNamesCodeList.add(buff.code);

procedureArgsNumList.add(j);

**return** "";

}

**private** String procedureDeclarations(TreeNode root) {

**if** (root.get(0).value!=Tables.***procedure***) {

**return** "";

}

**return** procedure(root.get(0)) + procedureDeclarations(root.get(1));

}

**private** String declarations(TreeNode root) {

**return** procedureDeclarations(root.get(0));

}

**private** String block(TreeNode root) {

**return** declarations(root.get(0)) + statementsList(root.get(2));

}

**private** String program(TreeNode root) {

Lexeme buff = procedureIdentifier(root.get(1));

ArrayList <Lexeme> list = parametersList(root.get(2));

ArrayList <Integer> parameters = **new** ArrayList<Integer>(list.size());

String code;

**int** j = 0;

proceduresNamesCodeList.add(buff.code);

code = table.getTokken(buff.code) + " PROC\n" + "POP EBX\n";

**for** (**int** i = list.size() - 1; i >= 0; i--) {

**int** id = parameters.indexOf(list.get(i).code);

**if** (id != -1) {

table.addCodeError(list.get(i), table.getTokken(list.get(i).code) + " has already been used as argument in this procedure");

code += "POP EAX\n";

code += "NOP; " + table.getTokken(list.get(i).code) + "\n";

j++;

} **else** **if** (buff.code == list.get(i).code) {

table.addCodeError(list.get(i), table.getTokken(list.get(i).code) + " has already been used as name of this procedure");

code += "POP EAX\n";

code += "NOP; " + table.getTokken(list.get(i).code) + "\n";

j++;

} **else** {

code += "POP EAX\n";

code += "MOV " + table.getTokken(list.get(i).code) + ", EAX\n";

procedureArgsList.add(list.get(i).code);

j++;

}

}

code += "PUSH EBX\n";

String vars = "";

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

vars += table.getTokken(procedureArgsList.get(i)) + " dword ?\n";

beforeCall += "MOV EAX, " + table.getTokken(procedureArgsList.get(i)) + "\n";

beforeCall += "PUSH EAX\n";

afterCall = "MOV " + table.getTokken(procedureArgsList.get(i)) + ", EAX\n" + afterCall;

afterCall = "POP EAX\n" + afterCall;

}

procedureArgsNumList.add(j);

**return** vars + code + block(root.get(4)) + table.getTokken(buff.code) + " ENDP\n";

}

**private** String signalProgram(TreeNode root) {

**return** program(root.get(0));

}

**public** String generate(Tables inTable) {

table = inTable;

**return** signalProgram(table.treeRoot);

}

**public** Coder() {

}

}

**True-Тест**

PROCEDURE ROCKET(ALD,BOR,DEI);

PROCEDURE ROCKET1(ALD,DEI);

PROCEDURE ROCKET2(ALD);

PROCEDURE ROCKET3;

BEGIN

ROCKET(1,2,3);

ROCKET1(1,2);

ROCKET2(1);

ROCKET3;

RETURN;

END;

DEI dword ?

BOR dword ?

ALD dword ?

ROCKET PROC

POP EBX

POP EAX

MOV DEI, EAX

POP EAX

MOV BOR, EAX

POP EAX

MOV ALD, EAX

PUSH EBX

MOV EAX, DEI

PUSH EAX

MOV EAX, BOR

PUSH EAX

MOV EAX, ALD

PUSH EAX

MOV EAX, 1

PUSH EAX

MOV EAX, 2

PUSH EAX

MOV EAX, 3

PUSH EAX

CALL ROCKET

POP EAX

MOV ALD, EAX

POP EAX

MOV BOR, EAX

POP EAX

MOV DEI, EAX

MOV EAX, DEI

PUSH EAX

MOV EAX, BOR

PUSH EAX

MOV EAX, ALD

PUSH EAX

MOV EAX, 1

PUSH EAX

MOV EAX, 2

PUSH EAX

CALL ROCKET1

POP EAX

MOV ALD, EAX

POP EAX

MOV BOR, EAX

POP EAX

MOV DEI, EAX

MOV EAX, DEI

PUSH EAX

MOV EAX, BOR

PUSH EAX

MOV EAX, ALD

PUSH EAX

MOV EAX, 1

PUSH EAX

CALL ROCKET2

POP EAX

MOV ALD, EAX

POP EAX

MOV BOR, EAX

POP EAX

MOV DEI, EAX

MOV EAX, DEI

PUSH EAX

MOV EAX, BOR

PUSH EAX

MOV EAX, ALD

PUSH EAX

CALL ROCKET3

POP EAX

MOV ALD, EAX

POP EAX

MOV BOR, EAX

POP EAX

MOV DEI, EAX

RET

NOP

ROCKET ENDP

No errors

**false-Тест**

PROCEDURE ROCKET(ALD,BOR,DEI);

PROCEDURE ROCKET(ALD,DEI);

PROCEDURE ALD(ALD);

PROCEDURE ROCKET3(ALD,DEI,ALD,DEI);

BEGIN

ROCKET(1);

ROCKET(1,2,3,4);

ROCKET1(1,2);

ALD(1);

RETURN;

END;

DEI dword ?

BOR dword ?

ALD dword ?

ROCKET PROC

POP EBX

POP EAX

MOV DEI, EAX

POP EAX

MOV BOR, EAX

POP EAX

MOV ALD, EAX

PUSH EBX

NOP

NOP

NOP

NOP

RET

NOP

ROCKET ENDP

Errors:

Code Generator: Error (line 2, column 11): ROCKET has already been used as procedure name

Code Generator: Error (line 3, column 11): ALD has already been used as procedure argument

Code Generator: Error (line 4, column 27): ALD has been used twice in this procedure

Code Generator: Error (line 6, column 8): ROCKET expected 3 arguments, given 1

Code Generator: Error (line 7, column 14): ROCKET expected 3 arguments, given 4

Code Generator: Error (line 8, column 1): ROCKET1 - unknown procedure identifier

Code Generator: Error (line 9, column 1): ALD - unknown procedure identifier