НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ “КПІ”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота №3

з дисципліни

“Інженерія програмного забезпечення”

Виконав студент гр. КВ-22

Ткаченко Роман

Перевірив

_____

Київ 2015

Індивідуальне завдання:

Варіант 19

1. Разробити програму генератора коду (ГК) для подмножини мови програмування SIGNAL, заданої за варіантом.
2. Програма має забезпечувати:
· читання дерева розбору та таблиць, створених синтаксичним аналізатором, який було розроблено в розрахунково-графічній роботі;
· виявлення семантичних помилок;
· генерацію коду та/або побудову внутрішніх таблиць для генерації коду.
3. Входом генератора коду (ГК) мають бути:
· дерево розбору;
· таблиці ідентифікаторів та констант з повною інформацією, необхідною для генерації коду;
· вхідна програма на підмножині мови програмування SIGNAL згідно з варіантом (необхідна для формування лістингу програми).
4. Виходом ГК мають бути:
· асемблерний код згенерований для вхідної програми та/або внутрішні таблиці для генерації коду;
· внутрішні таблиці генератора коду (якщо потрібні).
5. Зкомпонувати повний компілятор, що складається з розроблених раніше лексичного та синтаксичного аналізаторів і генератора коду, який забезпечує наступне:
· генерацію коду та/або побудову внутрішніх таблиць для генерації коду;
· формування лістингу вхідної програми з повідомленнями про лексичні, синтаксичні та семантичні помилки.
6. Входом компілятора має бути програма на підмножині мови програмування SIGNAL згідно з варіантом;
7. Виходом компілятора мають бути:
· асемблерний код згенерований для вхідної програми та/або внутрішні таблиці для генерації коду;
· лістинг вхідної програми з повідомленнями про лексичні, синтаксичні та семантичні помилки.
8. Для програмування може бути використана довільна алгоритмічна мова програмування високого рівня. Якщо обрана мова програмування має конструкції або бібліотеки для роботи з регулярними виразами, то використання цих конструкцій та/або бібліотек строго заборонено.

Граматика мови:

1. <signal-program> --> <program>
2. <program> --> PROGRAM <procedure-identifier> ;
<block>.
3. <block> --> <variable-declarations> BEGIN
<statements-list> END
4. <variable-declarations> --> VAR <declarations-list> |
<empty>
5. <declarations-list> --> <declaration> <declarations-list> |
<empty>
6. <declaration> --><variable-identifier>:<attribute>
;
7. <attribute> --> INTEGER |
FLOAT
8. <statements-list> --> <statement> <statements-list> |

<empty>
9. <statement> --> <condition-statement> ENDIF ;
10. <condition-statement> --> <incomplete-
conditionstatement><alternative-part>
11. <incomplete-condition-statement> --> IF <conditional-expression>
THEN <statements-list>
12. <alternative-part> --> ELSE <statements-list> |
<empty>
13. <conditional-expression> --> <expression> = <expression>
14. <expression> --> <variable-identifier> |
<unsigned-integer>
15. <variable-identifier> --> <identifier>
16. <procedure-identifier> --> <identifier>
17. <identifier> --> <letter><string>
18. <string> --> <letter><string> |
<digit><string> |
<empty>
19. <unsigned-integer> --> <digit><digits-string>
20. <digits-string> --> <digit><digits-string> |
<empty>
21. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
9
22. <letter> --> A | B | C | D | ... | Z

Текст програми:

```
using lexer.SyntaxTree;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lexer.AsmCodeGenerator
{
    class AssemblerCodeGenerator
    {
        public AssemblerCodeGenerator()
        {
            XMLSyntaxTree = SerializeTables.DeseriaizeNode();
            resultAsmCode = "";
            posInResultAsmCode = 0;
            dataSegmentPos = 0;
            codeSegmentPos = 0;
            labelNumber = 0;
        }
        private XMLNode XMLSyntaxTree;
        private string resultAsmCode;
        private int posInResultAsmCode;
        private int dataSegmentPos;
        private int codeSegmentPos;
        private static int labelNumber;

        public delegate void WorkDoneHandler(string output);
        public event WorkDoneHandler WorkDone;

        private string generateLabel()
        {
            labelNumber++;
```

```
            return String.Format("L{0}", labelNumber);
        }

        private void WriteHeader(string idn)
        {
            string header = ".386\n.MODEL\tsmall\n.STACK\t256\n";
            string codeSeg = String.Format(".CODE\n{0}\tPROC\n", idn);
            string endProg =
String.Format("mov\tah,4Ch\nmov\tal,0\nint\t21h\n{0}\tENDP\nEND\t{0}", idn);
            resultAsmCode = resultAsmCode.Insert(posInResultAsmCode, header); // insert
header
            dataSegmentPos = resultAsmCode.Length; // set pos to continue writing declar
if needed
            posInResultAsmCode = dataSegmentPos;
            resultAsmCode = resultAsmCode.Insert(posInResultAsmCode, codeSeg); // insert
code start point
            codeSegmentPos = resultAsmCode.Length;
            resultAsmCode = resultAsmCode.Insert(codeSegmentPos, endProg); // insert end
of prog
        }

        private void WriteDataSeg()
        {
            string dataSeg = ".DATA\n";
            resultAsmCode = resultAsmCode.Insert(dataSegmentPos, dataSeg);
            dataSegmentPos += dataSeg.Length;
            codeSegmentPos += dataSeg.Length;
        }

        private void WriteDeclaration(string idn)
        {
            string declar = String.Format("{0}\tdd\t?\n", idn);
            resultAsmCode = resultAsmCode.Insert(dataSegmentPos, declar);
            dataSegmentPos += declar.Length;
            codeSegmentPos += declar.Length;
        }

        private string WriteCondExpr(List<XMLNode> expressions) // returns label
        {
            var operands = expressions.ToArray();
            string label = generateLabel();
            string condition = String.Format("mov\teax, {0}\nmov\tebx, {1}\ncmp\teax,
ebx\njne\t{2}\n", operands[0].value, operands[1].value, label);
            resultAsmCode = resultAsmCode.Insert(codeSegmentPos, condition);
            codeSegmentPos += condition.Length;
            return label;
        }

        private string WriteJumpToEndif()
        {
            string label = generateLabel();
            string jmp = String.Format("jmp\t{0} \n", label);
            resultAsmCode = resultAsmCode.Insert(codeSegmentPos, jmp);
            codeSegmentPos += jmp.Length;
            return label;
        }

        private void WriteLabel(string label)
        {
            string writeLabel = String.Format("{0}:\n", label);
            resultAsmCode = resultAsmCode.Insert(codeSegmentPos, writeLabel);
            codeSegmentPos += writeLabel.Length;
        }

        private void ParseNode(XMLNode parentNode)
```

```
        {
            bool parseStatement = false; // if true then do custom parse not just
straight going throw the tree
            foreach (var item in parentNode.nodes)
            {
                if (item.name == nodesTypes.procedure_idn)
                    WriteHeader(item.value);
                if (item.name == nodesTypes.var_declar && item.nodes.Count > 0)
                    WriteDataSeg();
                if (item.name == nodesTypes.declaration)
                    WriteDeclaration(item.nodes.First(x => x.name ==
nodesTypes.var_idn).value);
                if (item.name == nodesTypes.statement_list)
                {
                    parseStatement = true;
                    List<XMLNode> conditional_statement = item.nodes
                                        .First(x => x.name == nodesTypes.statement)
                                        .nodes.First(x => x.name ==
nodesTypes.conditional_statement).nodes; // get nodes of if cond statement
                    List<XMLNode> incomplete_cond = conditional_statement.First(x =>
x.name == nodesTypes.incomplete_conditional_statement).nodes;
                    List<XMLNode> cond_expression = incomplete_cond.First(x => x.name ==
nodesTypes.conditional_expression).nodes;

                    string label = WriteCondExpr(cond_expression.FindAll(x => x.name ==
nodesTypes.expression));

                    if (incomplete_cond.Exists(x => x.name == nodesTypes.statement_list))
// if after THEN there is another statement
                        ParseNode(conditional_statement.First(x => x.name ==
nodesTypes.incomplete_conditional_statement)); //parse statement

                    string labelEndif = "";
                    bool elsePartExists = conditional_statement.Exists(x => x.name ==
nodesTypes.alternative_part);
                    if (elsePartExists) // if ELSE part exist write jmp to endif if THEN
part executes
                        labelEndif = WriteJumpToEndif();

                    WriteLabel(label);

                    if (elsePartExists) // parse statement in else part if exists
                        if (conditional_statement.First(x => x.name ==
nodesTypes.alternative_part).nodes.Exists(x => x.name == nodesTypes.statement_list))
                            ParseNode(conditional_statement.First(x => x.name ==
nodesTypes.alternative_part));

                    if (labelEndif != "")
                        WriteLabel(labelEndif);
                }
                if (!parseStatement)
                    ParseNode(item);
                else
                    continue;
            }
        }
        public void GenerateCode()
        {
            ParseNode(XMLSyntaxTree);
            if (WorkDone != null) WorkDone(resultAsmCode);
        }
    }
}
```

Приклади:

Вхідна програма:

```
PROGRAM HELLOWORLD; (* valid comment *)
VAR FIRST : INTEGER;
    SECOND : FLOAT;
    THIRD : INTEGER;
BEGIN
IF 1 = FIRST THEN
   IF 5 = 6 THEN ELSE IF 4 = 5 THEN ENDIF; ENDIF;
ELSE IF SECOND = 1 THEN ENDIF;
ENDIF;
END.
```

Вихідний лістинг:

```
.386

.MODEL   small

.STACK   256

.DATA

FIRST    dd   ?

SECOND   dd   ?

THIRD    dd   ?

.CODE

HELLOWORLD   PROC

mov      eax, 1

mov      ebx, FIRST

cmp      eax, ebx

jne      L1

mov      eax, 5

mov      ebx, 6

cmp      eax, ebx

jne      L2

jmp      L3

L2:

mov      eax, 4

mov      ebx, 5

cmp      eax, ebx

jne      L4

L4:

L3:

jmp      L5
```

```
L1:
mov      eax, SECOND
mov      ebx, 1
cmp      eax, ebx
jne      L6
L6:
L5:
mov      ah,4Ch
mov      al,0
int      21h
HELLOWORLD   ENDP
END      HELLOWORLD
```