Лабораторна робота №3

з дисципліни

Теорія формальних мов та компіляція

на тему

«Розробка генератора коду»

Варіант №17

|  |  |
| --- | --- |
| Виконав студент  групи КВ-64М  Подольський С. В.  залікова книжка № КВ6415 | Перевірив:  Марченко О. І.  \_\_\_\_\_\_\_\_\_ |

1. **Постановка задачі**
2. **Разробити програму генератора коду (ГК) для подмножини мови програмування SIGNAL.**

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.

**Входом** 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).

**Виходом** ГК мають бути:

* асемблерний код згенерований для вхідної програми та/або внутрішні таблиці для генерації коду;
* внутрішні таблиці генератора коду (якщо потрібні);
  1. **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.

**Входом** компілятора має бути програма на підмножині мови програмування SIGNAL згідно з варіантом;

**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.

1. **Індивідуальне завдання згідно із варіантом**

Цільовий процесор – ColdFire

**Перетворена граматика:**

<signal-program> → <program>

<program> → PROGRAM <identifier> ; <block> .

* 1. <block> → BEGIN <statements-list> END
  2. <block> → BEGIN END

<statements-list> → <statements>

<statements> → <statement>

<statements> → <statement> <statements>

<statement> → LOOP <statements-list> ENDLOOP ;

<statement> → LOOP ENDLOOP ;

<statement> → FOR <identifier> := <loop-declaration> ENDFOR ;

<loop-declaration> → <expression> TO <expression> DO <statements-list>

<loop-declaration> → <expression> TO <expression> DO

<expression> → <summands-list>

<expression> → - <summands-list>

<summands-list> → <summand>

<summands-list> → <summand> - <summands-list>

<summands-list> → <summand> + <summands-list>

<summand> → <identifier>

<summand> → <unsigned-integer>

<unsigned-integer> → N1 | N2 | … | NK

<identifier> → T1 | T2 | … | TN

Де N1, N2, …, NK – константи користувача, згорнуті лексичним аналізатором;

T1, T2, …, TN – ідентифікатори користувача, згорнуті лексичним аналізатором.

1. **Лістинг програми генератора коду**

using System.Linq;

using SyntaxAnalyzer;

namespace SemanticAnalyzer

{

class CodeGenerator

{

const string label = "?LABEL";

int labelNumber = 0;

string programIdentifier;

/// <summary>

/// Generate ColdFire assembly from parse tree

/// </summary>

/// <param name="tree">Parse tree</param>

/// <param name="misc">Misc params</param>

/// <returns>ColdFire assembly</returns>

public string ColdFire(Tree tree, params string[] misc)

{

switch (tree.Token)

{

case "<unsigned-integer>":

uint \_;

if (uint.TryParse(tree[0].Token, out \_))

goto case "<identifier>";

throw new SemanticException("Value " + tree[0].Token + " was too large for unsigned integer");

case "<identifier>":

if (tree[0].Token == programIdentifier)

throw new SemanticException("Identifier " + tree[0].Token + " is already used as program identifier");

return tree[0].Token;

}

var rule = tree.Aggregate(tree.Token + " →", (result, node) => result + " " + node.Token);

switch (rule)

{

case "<signal-program> → <program>":

return ColdFire(tree[0]);

case "<program> → PROGRAM <identifier> ; <block> .":

return string.Format(".global\t{0}\n{0}:\n{1}\trts\n", programIdentifier = ColdFire(tree[1]), ColdFire(tree[3]));

case "<block> → BEGIN <statements-list> END":

return ColdFire(tree[1]);

case "<block> → BEGIN END":

return string.Empty;

case "<statements-list> → <statements>":

return ColdFire(tree[0]);

case "<statements> → <statement>":

return ColdFire(tree[0]);

case "<statements> → <statement> <statements>":

return ColdFire(tree[0]) + ColdFire(tree[1]);

case "<statement> → LOOP <statements-list> ENDLOOP ;":

return string.Format("{0}{1}:\n{2}\tBRA\t{0}{1}\n", label, labelNumber++, ColdFire(tree[1]));

case "<statement> → LOOP ENDLOOP ;":

return string.Format("{0}{1}:\n\tBRA\t{0}{1}\n", label, labelNumber++);

case "<statement> → FOR <identifier> := <loop-declaration> ENDFOR ;":

return ColdFire(tree[3], ColdFire(tree[1]));

case "<loop-declaration> → <expression> TO <expression> DO <statements-list>":

return string.Format("{0}\tMOVE.L\tD0, {1}\n{2}{3}:\n{4}\tCMP.L\tD0, {1}\n\tBGT\t{2}{5}\n{6}\tADD.L\t#1, {1}\n\tBRA\t{2}{3}\n{2}{5}:\n\tNOP\n",

ColdFire(tree[0]), misc[0], label, labelNumber++, ColdFire(tree[2]), labelNumber++, ColdFire(tree[4]));

case "<loop-declaration> → <expression> TO <expression> DO":

return string.Format("{0}\tMOVE.L\tD0, {1}\n{2}{3}:\n{4}\tCMP.L\tD0, {1}\n\tBGT\t{2}{5}\n\tADD.L\t#1, {1}\n\tBRA\t{2}{3}\n{2}{5}:\n\tNOP\n",

ColdFire(tree[0]), misc[0], label, labelNumber++, ColdFire(tree[2]), labelNumber++);

case "<expression> → <summands-list>":

return string.Format("\tMOVE.L\t{0}", ColdFire(tree[0]));

case "<expression> → - <summands-list>":

return string.Format("\tMOVE.L\t0, D0\n\tSUB.L\t{0}", ColdFire(tree[1]));

case "<summands-list> → <summand>":

return string.Format("{0}, D0\n", ColdFire(tree[0]));

case "<summands-list> → <summand> - <summands-list>":

return string.Format("{0}, D0\n\tSUB.L\t{1}", ColdFire(tree[0]), ColdFire(tree[2]));

case "<summands-list> → <summand> + <summands-list>":

return string.Format("{0}, D0\n\tADD.L\t{1}", ColdFire(tree[0]), ColdFire(tree[2]));

case "<summand> → <identifier>":

return ColdFire(tree[0]);

case "<summand> → <unsigned-integer>":

return "#" + ColdFire(tree[0]);

}

throw new SemanticException("unknown rule\n" + rule);

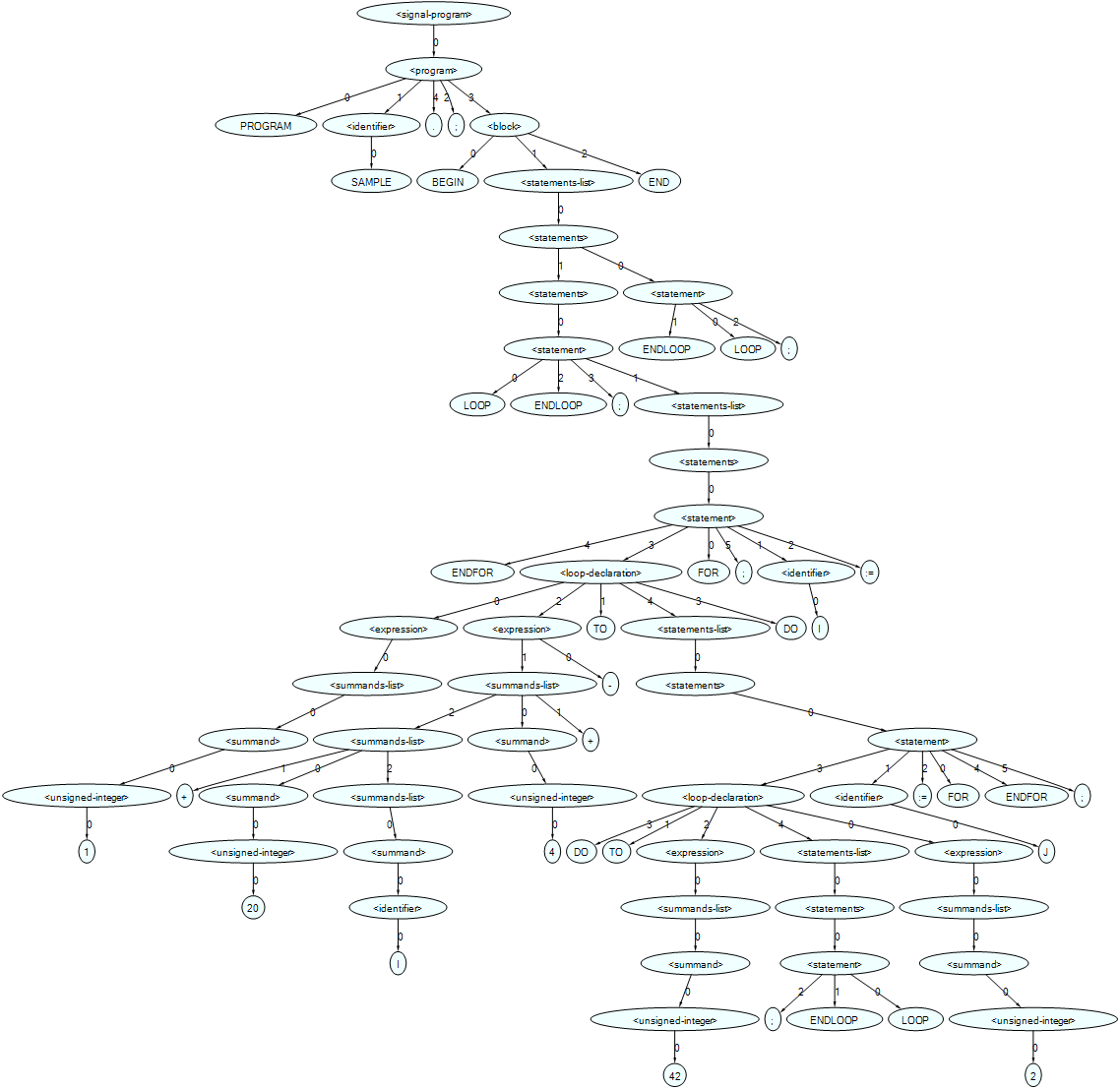
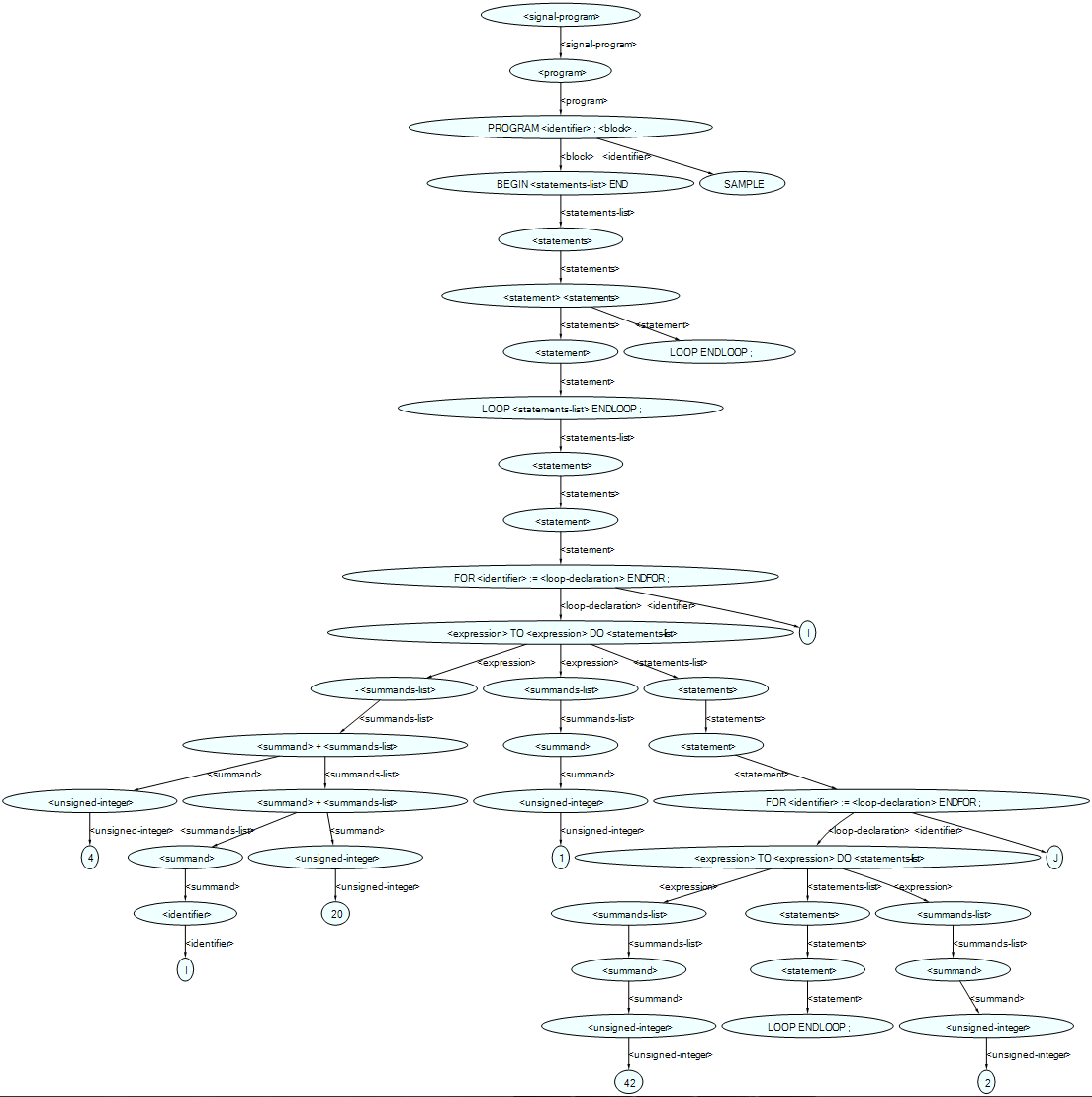
}

}

}

1. **Контрольні приклади**

|  |  |
| --- | --- |
| PROGRAM sample;  BEGIN  LOOP  ENDLOOP;  LOOP  FOR i := 1 TO -4+20+i DO  FOR j := 2 TO 42 DO  LOOP  ENDLOOP;  ENDFOR;  ENDFOR;  ENDLOOP;  (\*comment\*)  END. | .global SAMPLE  SAMPLE:  ?LABEL0:  BRA ?LABEL0  ?LABEL1:  MOVE.L #1, D0  MOVE.L D0, I  ?LABEL2:  MOVE.L 0, D0  SUB.L #4, D0  ADD.L #20, D0  ADD.L I, D0  CMP.L D0, I  BGT ?LABEL3  MOVE.L #2, D0  MOVE.L D0, J  ?LABEL4:  MOVE.L #42, D0  CMP.L D0, J  BGT ?LABEL5  ?LABEL6:  BRA ?LABEL6  ADD.L #1, J  BRA ?LABEL4  ?LABEL5:  NOP  ADD.L #1, I  BRA ?LABEL2  ?LABEL3:  NOP  BRA ?LABEL1  rts  Press any key to continue . . . |

|  |  |
| --- | --- |
| PROGRAM sample;  BEGIN  LOOP  ENDLOOP;  LOOP  FOR i := 1 TO -4+20123423665+i DO  FOR j := 2 TO 42 DO  LOOP  ENDLOOP;  ENDFOR;  ENDFOR;  ENDLOOP;  (\*comment\*)  END. | Semantic error:  Value 20123423665 was too large for unsigned integer  Press any key to continue . . . |
| PROGRAM sample;  BEGIN  LOOP  ENDLOOP;  LOOP  FOR sample := 1 TO -4+20+i DO  FOR j := 2 TO 42 DO  LOOP  ENDLOOP;  ENDFOR;  ENDFOR;  ENDLOOP;  (\*comment\*)  END. | Semantic error:  Identifier SAMPLE is already used as program identifier  Press any key to continue . . . |