Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра ТПИ

ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ

Лабораторная работа № 4

# Разработка и реализация блока генерации кода

Факультет: ПМИ Преподаватели:

Еланцева И.Л.,

Петров Р. В.

Группа: ПМ-81

Студенты: Ефремов А. А.,

Ртищева К. С.

Бригада: 1

Вариант: 1

Новосибирск

2021

1. **Цель работы**

Изучить методы генерации кода с учетом различных промежуточных форм представления программ. Изучить методы управления памятью и особенности из использования на этапе генерации кода.

Научиться проектировать генератор кода.

1. **Условие задачи**

Подмножество языка С++ включает:

* данные типа int;
* инструкции описания переменных;
* операторы присваивания, if, if- else любой вложенности и в любой последовательности;
* операции +, –, \*, /, ==, !=, < .

В соответствии с выбранным вариантом реализовать генератор кода. Исходными данными являются:

* синтаксическое дерево или постфиксная запись, построенные в лабораторной работе №3;
* таблицы лексем.

Результатом выполнения лабораторной работы является программа на языке Ассемблер, разработанная на основе знаний и практических навыков, полученных при изучении курса «Языки программирования и методы трансляции (часть I)».

В режиме отладки продемонстрировать работоспособность генератора кода и транслятора в целом.

1. **Тестовые примеры**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входные данные** | **Выходные данные** | **Назначение** |
| 1 | **“prog.txt”**  int main()  {  int a, b = 3;  a = (3 + 13) - 4 \* 2;  return 0;  }  **“postfixSimple.txt”**  b 3 =  a 3 13 + 4 2 \* - = | **“code.asm”**  .386  .model FLAT, C  .data  a dd ?  b dd 3  const\_3 dd 3  const\_13 dd 13  const\_4 dd 4  const\_2 dd 2  .code  main proc  fild const\_3  fild const\_13  fadd  fild const\_4  fild const\_2  fmul  fsub  fistp a  mov eax, 0  ret  main endp  end main | Инициализация;  Присваивание;  Выражение с операторами разного приоритета |
| 2 | **“prog.txt”**  int main()  {  int a = 5, b;  if (a == 5)  {  b = a;  }  return 0;  }  **“postfixSimple.txt”**  a 5 =  a 5 == m1 CJF b a = m1: | **“code.asm”**  .386  .model FLAT, C  .data  a dd 5  b dd ?  const\_5 dd 5  .code  main proc  fild a  fild const\_5  fcomp  fstsw ax  sahf  jne m1  fild a  fistp b  m1:  mov eax, 0  ret  main endp  end main | if без else |
| 3 | **“prog.txt”**  int main()  {  int a = 2, b;  if (a == 5)  {  b = a;  }  else  {  b = 5;  }  return 0;  }  **“postfixSimple.txt”**  a 2 =  a 5 == m1 CJF b a = m2 UJ m1: b 5 = m2: | **“code.asm”**  .386  .model FLAT, C  .data  a dd 2  b dd ?  const\_5 dd 5  .code  main proc  fild a  fild const\_5  fcomp  fstsw ax  sahf  jne m1  fild a  fistp b  jmp m2  m1:  fild const\_5  fistp b  m2:  mov eax, 0  ret  main endp  end main | if с else |
| 4 | **“prog.txt”**  int main()  {  int a = 5, b = 5;  if (a == 5)  {  if (b == 3)  {  b = a;  }  else  {  b = 3;  }  }  else  {  a = b;  }  return 0;  }  **“postfixSimple.txt”**  a 5 =  b 5 =  a 5 == m1 CJF b 3 == m2 CJF b a = m3 UJ m2: b 3 = m3:  m4 UJ m1: a b = m4: | **“code.asm”**  .386  .model FLAT, C  .data  a dd 5  b dd 5  const\_5 dd 5  const\_3 dd 3  .code  main proc  fild a  fild const\_5  fcomp  fstsw ax  sahf  jne m1  fild b  fild const\_3  fcomp  fstsw ax  sahf  jne m2  fild a  fistp b  jmp m3  m2:  fild const\_3  fistp b  m3:  jmp m4  m1:  fild b  fistp a  m4:  mov eax, 0  ret  main endp  end main | Вложенный if |

1. **Текст программы**

***Файл “VarTableRow.h”***

#pragma once

#include <string>

using namespace std;

class VarTableRow

{

public:

int value;

string name;

bool is\_set;

VarTableRow() {};

VarTableRow(const int& t\_value, const string& t\_name, const bool t\_is\_set) :

value(t\_value), name(t\_name), is\_set(t\_is\_set) {};

bool operator == (VarTableRow lhs)

{

return value == lhs.value && name == lhs.name && is\_set == lhs.is\_set;

}

};

***Файл “VarTable.h”***

#pragma once

#include <vector>

#include <string>

#include <iomanip>

#include "VarTableRow.h"

using namespace std;

class VarTable

{

public:

vector<VarTableRow> table;

// Создание пустой таблицы

VarTable()

{

table = vector<VarTableRow>(0);

}

// Функция поиска номера строки таблицы по идентификатору,

// возращает -1 в случае отсутствия строки с таким идентификатором в таблице

int GetRowIndex(const VarTableRow& t\_row)

{

for(size\_t i = 0; i < table.size(); i++)

if(table[i] == t\_row)

return i;

return -1;

}

// Функция добавления строки в таблицу, если такого вхождения нет,

// возвращает номер строки

int AddRow(const VarTableRow& t\_row)

{

int index = GetRowIndex(t\_row);

if(index == -1)

{

table.push\_back(t\_row);

return table.size() - 1;

}

else

return index;

}

// Функция, возвращающая

VarTableRow GetRow(const int& t\_index)

{

if(t\_index < table.size())

return table[t\_index];

else

printf\_s("Error!");

}

void Output(const string& OUT\_FILE)

{

ofstream fout(OUT\_FILE);

fout << "i value name is set" << endl;

for (size\_t i = 0; i < table.size(); i++)

{

fout << i << setw(5) << table[i].value;

fout << setw(8) << table[i].name;

fout << setw(5) << table[i].is\_set;

fout << endl;

}

fout.close();

}

// Получение значений атрибутов

int GetValue(const int& t\_index) { return table[t\_index].value; }

string GetName(const int& t\_index) { return table[t\_index].name; }

bool GetIsSet(const int& t\_index) { return table[t\_index].is\_set; }

int GetIndexByName(const string& t\_name)

{

for (int i = 0; i < table.size(); i++)

if (table[i].name == t\_name)

return i;

return -1;

}

// Установление значений атрибутов

void SetValue(const int& t\_index, const int& t\_value) {table[t\_index].value = t\_value;}

void SetName(const int& t\_index, const string& t\_name) { table[t\_index].name = t\_name; }

void SetIsSet(const int& t\_index, const bool t\_is\_set) {table[t\_index].is\_set=t\_is\_set;}

};

***Файл “ConstTableRow.h”***

#pragma once

#include <string>

using namespace std;

class ConstTableRow

{

public:

string name;

ConstTableRow() {};

ConstTableRow(const string& t\_name) :

name(t\_name) {};

bool operator == (const ConstTableRow& lhs)

{

return name == lhs.name;

}

};

***Файл “ConstTable.h”***

#pragma once

#include <vector>

#include <string>

#include <iomanip>

#include "ConstTableRow.h"

using namespace std;

class ConstTable

{

public:

vector<ConstTableRow> table;

// Создание пустой таблицы

ConstTable()

{

table = vector<ConstTableRow>(0);

}

// Создание таблицы с ключевыми словами

void FillKeyWords()

{

const int k = 5;

table.resize(k);

string key\_words[k] = {"if", "else", "main", "return", "int"};

for(size\_t i = 0; i < k; i++)

table[i] = ConstTableRow(key\_words[i]);

}

// Создание таблицы с операторами

void FillOperators()

{

const int k = 14;

table.resize(k);

string operators[k] = { "=", "+", "-", "\*" , "/", "==", "!=", "<", "(", ")", "{", "}",

",", ";"};

for(size\_t i = 0; i < k; i++)

table[i] = ConstTableRow(operators[i]);

}

// Создание таблицы со всеми символами алфавита языка

void FillAplhabet()

{

const int k = 17;

table.resize(k + 26 + 26 + 10);

string operators[k] = { "=", "+", "-", "\*", "/", "=", "!", "<", "(", ")", "{", "}",

",", ";", "\_", "!=", "==" };

for(size\_t i = 0; i < k; i++)

table[i] = ConstTableRow(operators[i]);

for(int i = 0; i < 26; i++)

table[i + k] = ConstTableRow(string(1, (char)('a' + i)));

for(int i = 0; i < 26; i++)

table[i + k + 26] = ConstTableRow(string(1, (char)('A' + i)));

for (int i = 0; i < 10; i++)

table[i + k + 26 + 26] = ConstTableRow(string(1, (char)('0' + i)));

}

// Создание таблицы со всеми символами алфавита языка с которых

// могут начинаться идентификаторы

void FillIdentName()

{

table.resize(1 + 26 + 26);

for (int i = 0; i < 26; i++)

table[i] = ConstTableRow(string(1, (char)('a' + i)));

for (int i = 0; i < 26; i++)

table[i + 26] = ConstTableRow(string(1, (char)('A' + i)));

table[52] = ConstTableRow(string(1, (char)('\_')));

}

// Создание таблицы со всеми цифрами алфавита языка

void FillNumbers()

{

table.resize(10);

for(int i = 0; i < 10; i++)

table[i] = ConstTableRow(string(1, (char)('0' + i)));

}

// Функция поиска номера строки таблицы по идентификатору,

// возращает -1 в случае отсутствия строки с таким идентификатором в таблице

int GetRowIndex(const ConstTableRow& t\_row)

{

for(int i = 0; i < table.size(); i++)

if(table[i] == t\_row)

return i;

return -1;

}

string GetRow(const int& index) { return table[index].name; }

void Output(const string& OUT\_FILE)

{

ofstream fout(OUT\_FILE);

fout << "i name" << endl;

for (size\_t i = 0; i < table.size(); i++)

{

fout << setw(2) << i;

fout << setw(10) << table[i].name;

fout << endl;

}

fout.close();

}

};

***Файл “LexicalAnalyzer.h”***

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include "VarTable.h"

#include "ConstTable.h"

enum class WordType

{

Blank,

Operator,

Word,

Constant

};

enum class SymbolType

{

Separator,

Operator,

Letter,

Number,

Error

};

class LexicalAnalyzer

{

public:

ConstTable alphabet, key\_words, operators, numbers, ident\_name;

VarTable var\_table, const\_table;

LexicalAnalyzer()

{

alphabet.FillAplhabet();

key\_words.FillKeyWords();

operators.FillOperators();

numbers.FillNumbers();

ident\_name.FillIdentName();

}

// Определение типа символа и получение его индекса

// в соответствующей таблице

SymbolType GetSymbolType(const string& s, int& place)

{

// Разделитель

if(s == " " || s == "\n" || s == "\t")

return SymbolType::Separator;

// Ошибка

place = alphabet.GetRowIndex(ConstTableRow(s));

if(place == -1)

return SymbolType::Error;

// Оператор

place = operators.GetRowIndex(ConstTableRow(s));

if(place != -1)

return SymbolType::Operator;

// Символ с которого может начинаться имя переменной

place = ident\_name.GetRowIndex(ConstTableRow(s));

if(place != -1)

return SymbolType::Letter;

// Цифра

place = numbers.GetRowIndex(ConstTableRow(s));

if(place != -1)

return SymbolType::Number;

}

// Печать всех таблиц

void PrintAllTables(const string& directory)

{

alphabet.Output(directory + "/aplhabet.txt");

key\_words.Output(directory + "/keyWords.txt");

operators.Output(directory + "/operators.txt");

numbers.Output(directory + "/numbers.txt");

ident\_name.Output(directory + "/ident\_name.txt");

const\_table.Output(directory + "/const.txt");

var\_table.Output(directory + "/var.txt");

}

void MakeTokens(const string& in\_filename, const string& out\_filename)

{

ifstream fin(in\_filename);

ofstream fout(out\_filename);

int symbol\_n = 0, line\_n = 1;

char c;

string word = "", symbol;

// Тип передыдущего слова

WordType word\_type = WordType::Blank;

// Тип символа

SymbolType symbol\_type;

// Место символа в соответствующей таблице

int place = 0;

// Место предыдущего символа в соответствующей таблице

int prev\_place = 0;

// Если комментирование оператором \*/

bool is\_op\_comment = false;

string prev\_symbol;

// Если комментирование оператором //

bool is\_line\_comment = false;

while(fin.get(c))

{

symbol = c;

symbol\_n++;

symbol\_type = GetSymbolType(symbol, place);

if(symbol\_type == SymbolType::Error)

{

cout << "Error at line " << line\_n << " pos " << symbol\_n;

cout << ": Invalid symbol! ";

exit(2);

}

if(is\_op\_comment)

{

string temp\_s = prev\_symbol + symbol;

if(temp\_s == "\*/")

is\_op\_comment = false;

else

prev\_symbol = symbol;

}

else if(is\_line\_comment)

{

if(symbol == "\n")

is\_line\_comment = false;

}

else

switch(word\_type)

{

// Слово не задано

case WordType::Blank:

{

switch(symbol\_type)

{

case SymbolType::Separator:

{

if(symbol == "\n")

{

symbol\_n = 0;

line\_n++;

fout << endl;

}

word\_type = WordType::Blank;

break;

}

case SymbolType::Operator:

{

prev\_place = place;

word = symbol;

word\_type = WordType::Operator;

break;

}

case SymbolType::Letter:

{

word = symbol;

word\_type = WordType::Word;

break;

}

case SymbolType::Number:

{

word = symbol;

word\_type = WordType::Constant;

break;

}

}

break;

}

// Слово - оператор

case WordType::Operator:

{

switch(symbol\_type)

{

// Символ - разделитель

case SymbolType::Separator:

{

fout << "(20," << place << ")";

word\_type = WordType::Blank;

word = "";

if(symbol == "\n")

{

symbol\_n = 0;

fout << endl;

line\_n++;

}

break;

}

// Символ - оператор

case SymbolType::Operator:

{

string temp\_op = word + symbol;

if(temp\_op == "/\*")

{

word\_type = WordType::Blank;

word = "";

is\_op\_comment = true;

break;

}

if(temp\_op == "//")

{

word\_type = WordType::Blank;

word = "";

is\_line\_comment = true;

break;

}

int temp\_place = operators.GetRowIndex(ConstTableRow(temp\_op));

// Если оператор - "==" или "!="

if(temp\_place != -1)

{

fout << "(20," << temp\_place << ")";

word\_type = WordType::Blank;

word = "";

}

else if(temp\_op == "()")

{

fout << "(20," << prev\_place << ")";

word\_type = WordType::Operator;

word = symbol;

prev\_place = place;

}

else

{

cout << "Error at line " << line\_n << " pos " << symbol\_n;

cout << ": Invalid operator! ";

exit(2);

break;

}

break;

}

// Символ - буква

case SymbolType::Letter:

{

fout << "(20," << prev\_place << ")";

word\_type = WordType::Word;

word = symbol;

prev\_place = place;

break;

}

// Символ - цифра

case SymbolType::Number:

{

fout << "(20," << prev\_place << ")";

word\_type = WordType::Constant;

word = symbol;

prev\_place = place;

break;

}

}

break;

}

// Слово - слово

case WordType::Word:

{

switch(symbol\_type)

{

// Символ - разделитель

case SymbolType::Separator:

{

int kw\_place = key\_words.GetRowIndex(ConstTableRow(word));

// Если слово - ключеваое слово

if(kw\_place != -1)

fout << "(10," << kw\_place << ")";

else

fout<<"(30,"<< var\_table.AddRow(VarTableRow(-1,word,false))<<")";

word\_type = WordType::Blank;

word = "";

if(symbol == "\n")

{

symbol\_n = 0;

line\_n++;

fout << endl;

}

break;

}

// Символ - оператор

case SymbolType::Operator:

{

int kw\_place = key\_words.GetRowIndex(ConstTableRow(word));

// Если слово - ключеваое слово

if(kw\_place != -1)

fout << "(10," << kw\_place << ")";

else

fout<<"(30,"<< var\_table.AddRow(VarTableRow(-1,word,false))<<")";

word\_type = WordType::Operator;

prev\_place = place;

word = symbol;

break;

}

// Символ - буква

case SymbolType::Letter:

{

word += symbol;

break;

}

// Символ - цифра

case SymbolType::Number:

{

word += symbol;

break;

}

}

break;

}

// Слово - константа

case WordType::Constant:

{

switch(symbol\_type)

{

// Символ - разделитель

case SymbolType::Separator:

{

fout<<"(40,"<< const\_table.AddRow(VarTableRow(0,word,false)) << ")";

word\_type = WordType::Blank;

word = "";

if(symbol == "\n")

{

symbol\_n = 0;

line\_n++;

fout << endl;

}

break;

}

// Символ - оператор

case SymbolType::Operator:

{

fout<<"(40,"<< const\_table.AddRow(VarTableRow(0 word,false)) << ")";

word\_type = WordType::Operator;

word = symbol;

prev\_place = place;

break;

}

// Символ - буква

case SymbolType::Letter:

{

cout << "Error at line " << line\_n << " pos " << symbol\_n;

cout << ": Invalid constant (identifier)! ";

exit(2);

break;

}

// Символ - цифра

case SymbolType::Number:

{

word += symbol;

break;

}

}

break;

}

}

}

if(is\_op\_comment)

{

cout << "Unclosed comment!";

exit(2);

}

fout.close();

fin.close();

}

};

***Файл “SyntaxAnalyzer.h”***

#pragma once

#include "LexicalAnalyzer.h"

#include <sstream>

#include <stack>

#include <deque>

#include <map>

using namespace std;

class SyntaxlAnalyzer

{

public:

SyntaxlAnalyzer()

{

};

struct token

{

int tableNum, index;

char buf;

string readToken(ifstream& ftoken, LexicalAnalyzer la)

{

ftoken >> buf >> tableNum;

ftoken >> buf >> index >> buf;

switch (tableNum)

{

case 10:

return la.key\_words.GetRow(index);

case 20:

return la.operators.GetRow(index);

case 30:

return "v\_name";

case 40:

if (la.const\_table.GetName(index) == "0") return "0";

return "const";

default:

break;

}

}

string nameByToken(LexicalAnalyzer& la)

{

switch (tableNum)

{

case 10:

return la.key\_words.GetRow(index);

case 20:

return la.operators.GetRow(index);

case 30:

return la.var\_table.GetName(index);

case 40:

return la.const\_table.GetName(index);

default:

break;

}

}

};

struct parsingTableRow

{

vector<string> terminals;

int jump, accept, stack, retrn, error;

parsingTableRow(vector<string> \_terminals, int \_jump, int \_accept, int \_stack,

int \_retrn, int \_error)

{

terminals = \_terminals;

jump = \_jump;

accept = \_accept;

stack = \_stack;

retrn = \_retrn;

error = \_error;

}

bool isExist(const string& terminal)

{

for (size\_t i = 0; i < terminals.size(); i++)

if (terminals[i] == terminal)

return true;

return false;

}

};

vector<parsingTableRow> parsingTable;

map<string, int> priority = { {"+", 2}, {"-",2}, {"\*",3},{"=",0},{"==",1},{"!=",1 },

{"<",1}, {"/",3},{",",0} };

void readParseTable(const string& ParseTableFile)

{

ifstream fin(ParseTableFile);

string line, temp;

while (getline(fin, line))

{

vector<string> str;

vector<string> terminals;

stringstream ss(line);

while (ss >> temp)

str.push\_back(temp);

int i = 0;

for (i; i < str.size() - 5; i++)

terminals.push\_back(str[i]);

parsingTableRow row(terminals, stoi(str[i]), stoi(str[i+1]), stoi(str[i+2]),

stoi(str[i+3]), stoi(str[i+4]));

parsingTable.push\_back(row);

}

fin.close();

}

struct tree

{

string elem;

tree\* left, \* right;

tree(string \_elem = "@", tree\* \_left = NULL, tree\* \_right = NULL)

: elem(\_elem), left(\_left), right(\_right) {}

};

tree\* buildTree(deque<string>& postfix)

{

string c;

tree\* t;

c = postfix.back();

postfix.pop\_back();

if (c == "\*" || c == "/" || c == "+" || c == "-" || c == "=")

{

t = new tree(c);

t->right = buildTree(postfix);

t->left = buildTree(postfix);

return t;

}

else

{

t = new tree(c);

return t;

}

}

bool equalTrees(tree\* t1, tree\* t2)

{

if (t1 == NULL && t2 == NULL)

return true;

if (t1->elem == t2->elem)

{

if(equalTrees(t1->left, t2->left) && equalTrees(t1->right, t2->right) ||

equalTrees(t1->left, t2->right) && equalTrees(t1->right, t2->left) &&

t1->elem == "+")

return true;

else return false;

}

else return false;

}

void standIn(tree\*\* t)

{

tree\* d = \*t, \* f1, \* f2, \* temp;

bool isProcessed;

stack<tree\*> s;

stack<bool> was;

stack<int> count;

do

{

while (d)

{

s.push(d);

was.push(false);

d = d->left;

}

if (!s.empty())

{

do

{

d = s.top();

s.pop();

isProcessed = was.top();

was.pop();

if (isProcessed)

{

if (d->elem == "/" && d->left->elem == "/")

{

d->right = new tree("\*", d->left->right, d->right);

d->left = d->left->left;

}

if (d->elem == "/" && d->right->elem == "/")

{

d->left = new tree("\*", d->left, d->right->right);

d->right = d->right->left;

}

if (d->elem == "+" || d->elem == "-")

{

if (d->left->elem == "\*" && d->right->elem == "\*")

{

f1 = d->left;

f2 = d->right;

if (equalTrees(f1->left, f2->left))

{

d->left = f1->left;

d->right = new tree(d->elem, f1->right, f2->right);

d->elem = "\*";

}

else

if (equalTrees(f1->left, f2->right))

{

d->left = f1->left;

d->right = new tree(d->elem, f1->right, f2->left);

d->elem = "\*";

}

else

if (equalTrees(f1->right, f2->left))

{

d->left = f1->right;

d->right = new tree(d->elem, f1->left, f2->right);

d->elem = "\*";

}

else

if (equalTrees(f1->right, f2->right))

{

d->left = f1->right;

d->right = new tree(d->elem, f1->left, f2->left);

d->elem = "\*";

}

}

if (d->left->elem == "/" && d->right->elem == "/")

{

f1 = d->left;

f2 = d->right;

if (equalTrees(f1->right, f2->right))

{

d->right = f1->right;

d->left = new tree(d->elem, f1->left, f2->left);

d->elem = "/";

}

}

}

}

} while (isProcessed && !s.empty());

if (!isProcessed)

{

s.push(d);

was.push(true);

d = d->right;

}

}

} while (!s.empty());

}

void PostOrder(tree\* t, deque<string>& postfix)

{

stack<tree\*> s;

stack<bool> was;

bool isProcessed;

do

{

while (t)

{

s.push(t);

was.push(false);

t = t->left;

}

if (!s.empty())

{

do

{

t = s.top();

s.pop();

isProcessed = was.top();

was.pop();

if (isProcessed)

postfix.push\_back(t->elem);

} while (isProcessed && !s.empty());

if (!isProcessed)

{

s.push(t);

was.push(true);

t = t->right;

}

}

} while (!s.empty());

};

void simplification(deque<string>& postfix)

{

tree\* t = buildTree(postfix);

standIn(&t);

PostOrder(t, postfix);

}

void postfix(ofstream& postfixFile, ofstream& postfixSimpl, vector<token>& infix,

LexicalAnalyzer& la)

{

stack<string> op;

string tmpStr, sCur;

token tknCur;

deque<string> postfix, postfixSimple;

for (int i = 0; i < infix.size(); i++)

{

tknCur = infix[i];

if (tknCur.tableNum == 30 || tknCur.tableNum == 40)

postfix.push\_back(tknCur.nameByToken(la));

else

{

sCur = tknCur.nameByToken(la);

if (sCur == "(")

op.push(sCur);

else

if (sCur == ")")

{

while (op.top() != "(")

{

postfix.push\_back(op.top());

op.pop();

}

op.pop();

}

else

if (op.empty() || priority[sCur] == 0 || priority[sCur] >

priority[op.top()])

op.push(sCur);

else

{

while (priority[op.top()] >= priority[sCur])

{

postfix.push\_back(op.top());

op.pop();

}

op.push(sCur);

}

}

}

while (!op.empty())

{

postfix.push\_back(op.top());

op.pop();

}

while (!postfix.empty())

{

if (postfix.front() != ";")

{

postfixFile << postfix.front() << " ";

postfixSimple.push\_back(postfix.front());

}

postfix.pop\_front();

}

simplification(postfixSimple);

while (!postfixSimple.empty())

{

if (postfixSimple.front() != ";")

postfixSimpl << postfixSimple.front() << " ";

postfixSimple.pop\_front();

}

}

bool LL1(const string& tokenFile, const string& postfixFile, const string& postfixSimple,

LexicalAnalyzer& la)

{

ifstream ftoken(tokenFile);

ofstream fpostfix(postfixFile);

ofstream fsimple(postfixSimple);

token tknCur, tknNext;

stack<int> states, m1, m2;

int currState = 0, index = 0, if\_count = 0;

string sCur, sNext;

vector<token> infix;

bool OPZ = false;

if (ftoken.peek() == EOF) { return true; }

sCur = tknCur.readToken(ftoken, la);

do

{

if (parsingTable[currState].isExist(sCur))

{

if (parsingTable[currState].accept)

{

//объявление или идентификатор в присваивании

if (currState == 15 || currState == 48)

{

sCur = la.var\_table.GetName(tknCur.index);

if (currState == 15) //идентификатор в присваивании

{

//если не задан тип переменной

if (!la.var\_table.GetIsSet(tknCur.index))

{

cout << "Error: Unknown identifier '" << sCur << "'!";

return false;

}

}

else //объявление

{

//если тип переменной уже задан

if (la.var\_table.GetIsSet(tknCur.index))

{

cout << "Error: redescribing the type of a variable '" <<sCur<<"'!";

return false;

}

else { la.var\_table.SetIsSet(tknCur.index, 1); }

}

if (ftoken.peek() != EOF)

{

sNext = tknNext.readToken(ftoken, la);

if (sNext == "=") //идентификатор слева от =

{

la.var\_table.SetValue(tknCur.index, -2);

infix.push\_back(tknCur);

OPZ = true;

}

sCur = sNext;

tknCur = tknNext;

sNext = "";

}

}

else

{

if (currState == 25) // идентификатор в выражении

{

sCur = la.var\_table.GetName(tknCur.index);

//если не задан тип идентификатора

if (!la.var\_table.GetIsSet(tknCur.index))

{

cout << "Error: Unknown identifier '" << sCur << "'!";

return false;

}

//если не задано значение идентификатора

if (la.var\_table.GetValue(tknCur.index) == -1)

{

cout << "Error: Value of the variable '" << sCur << "' is not set!";

return false;

}

}

if (currState == 58) //объявление нескольких переменных

if (OPZ)

{

postfix(fpostfix, fsimple, infix, la);

fpostfix << endl;

fsimple << endl;

OPZ = false;

infix.clear();

}

//if

if (currState == 66) // ) в условии

{

if (OPZ)

{

postfix(fpostfix, fsimple, infix, la);

OPZ = false;

infix.clear();

index++;

m1.push(index);

fpostfix << "m" << index << " CJF ";

fsimple << "m" << index << " CJF ";

}

}

if (currState == 69) // } в if

{

if (OPZ)

{

postfix(fpostfix, fsimple, infix, la);

OPZ = false;

infix.clear();

}

}

if (currState == 85) // } в else

{

if (OPZ)

{

index++;

m2.push(index);

fpostfix << "m" << index << " UJ ";

fpostfix << "m" << m1.top() << ": ";

fsimple << "m" << index << " UJ ";

fsimple << "m" << m1.top() << ": ";

m1.pop();

postfix(fpostfix, fsimple, infix, la);

OPZ = false;

infix.clear();

fpostfix << "m" << m2.top() << ": " << endl;

fsimple << "m" << m2.top() << ": " << endl;

m2.pop();

if\_count--;

}

}

if (OPZ) { infix.push\_back(tknCur); }

if (ftoken.peek() != EOF)

{

sNext = tknNext.readToken(ftoken, la);

//идентификатор слева от == != <

if (sNext == "==" || sNext == "!=" || sNext == "<")

{

infix.push\_back(tknCur);

OPZ = true;

if\_count++;

}

sCur = sNext;

tknCur = tknNext;

sNext = "";

}

}

}

if (currState == 81) // нет else

{

fpostfix << "m" << m1.top() << ": " << endl;

fsimple << "m" << m1.top() << ": " << endl;

m1.pop();

if\_count--;

}

if (parsingTable[currState].stack)

states.push(currState + 1);

if (parsingTable[currState].jump > 0)

currState = parsingTable[currState].jump - 1;

else

{

if (parsingTable[currState].retrn)

{

if (!states.empty())

{

currState = states.top();

states.pop();

if (currState == 18 || currState == 46) //;

if (OPZ && if\_count == 0)

{

postfix(fpostfix, fsimple, infix, la);

fpostfix << endl;

fsimple << endl;

OPZ = false;

infix.clear();

}

}

else

{

if (currState != 8)

{

cout << "Syntax error: Stack is empty!";

return false;

}

}

}

}

}

else //если символа нет в столбце terminal

{

if (parsingTable[currState].error)

{

cout << "Error: Unexpected symbol! Possible symbols: ";

for (size\_t i = 0; i < parsingTable[currState].terminals.size(); i++)

cout << "'" << parsingTable[currState].terminals[i] << "' ";

return false;

}

else

currState++;

}

} while (ftoken.peek() != EOF);

if (currState == 8 && la.operators.GetRow(tknCur.index) == "}")

cout << "Success!";

else

{

cout << "Error: Incorrect end of the program! Expected '}'";

return false;

}

fpostfix.close();

fsimple.close();

ftoken.close();

return true;

}

};

***Файл “CodeGenerator.h”***

#pragma once

#include "LexicalAnalyzer.h"

#include <stack>

class CodeGenerator

{

public:

CodeGenerator()

{

};

void CodeToAsm(const string& asmFile, const string& postfixFile, LexicalAnalyzer& la)

{

ifstream fpostfix(postfixFile);

ofstream fasm(asmFile);

stack<string> Stack;

stringstream asmCode;

string line, temp;

vector<string> variableVector;

int index = -1;

fasm << ".386\n.model FLAT, C\n\n";

while(getline(fpostfix, line))

{

vector<string> postfix;

stringstream ss(line);

while (ss >> temp)

postfix.push\_back(temp);

if (postfix.size() == 3 &&

la.GetSymbolType(postfix[0], index) == SymbolType::Letter &&

postfix[2] == "=" &&

la.GetSymbolType(postfix[1], index) != SymbolType::Letter &&

la.GetSymbolType(postfix[1], index) != SymbolType::Operator)

{

index = la.var\_table.GetIndexByName(postfix[0]);

int a = stoi(postfix[1]);

la.var\_table.SetValue(index, a);

la.const\_table.SetIsSet(la.const\_table.GetIndexByName(postfix[1]), 1);

}

else

for (int i = 0; i < postfix.size(); i++)

{

if (postfix[i].find("m") != string::npos)

{

if (postfix[i].find(":") != string::npos)

{

asmCode << postfix[i] << endl;

continue;

}

else

{

asmCode << "\t" << "jmp " << postfix[i] << endl;

continue;

}

}

if (postfix[i] == "CJF" || postfix[i] == "UJ") { continue; }

if (la.GetSymbolType(postfix[i], index) != SymbolType::Operator)

{

Stack.push(postfix[i]);

bool added = false;

for (int j = 0; !added && j < (int)variableVector.size(); j++)

{

if (variableVector[j] == postfix[i])

added = true;

}

if (!added)

variableVector.push\_back(postfix[i]);

}

else

{

string oper1p, oper2p;

int type1 = 0, type2 = 0;

oper2p = Stack.top();

Stack.pop();

oper1p = Stack.top();

Stack.pop();

if (la.GetSymbolType(oper1p, index) == SymbolType::Letter)

{

if (postfix[i] != "=")

asmCode << "\tfild\t" << oper1p << "\n";

}

else

if (postfix[i] != "=" && oper1p != "last")

{

la.const\_table.SetIsSet(la.const\_table.GetIndexByName(oper1p), 0);

asmCode << "\tfild\tconst\_" << oper1p << "\n";

}

if (la.GetSymbolType(oper2p, index) == SymbolType::Letter)

asmCode << "\tfild\t" << oper2p << "\n";

else if (oper2p != "last")

{

la.const\_table.SetIsSet(la.const\_table.GetIndexByName(oper2p), 0);

asmCode << "\tfild\tconst\_" << oper2p << "\n";

}

if (postfix[i] == "+")

asmCode << "\tfadd\n";

else if (postfix[i] == "-")

{

if (oper2p == "last" && oper1p != "last")

asmCode << "\tfsubr\n";

else

asmCode << "\tfsub\n";

}

else if (postfix[i] == "\*")

{

asmCode << "\tfmul\n";

}

else if (postfix[i] == "/")

{

if (oper2p == "last" && oper1p != "last")

asmCode << "\tfdivr\n";

else

asmCode << "\tfdiv\n";

}

else if (postfix[i] == "==")

{

asmCode << "\tfcomp\n";

asmCode << "\tfstsw\tax\n\tsahf\n";

asmCode << "\tjne " << postfix[i + 1] << "\n";

i += 2;

}

else if (postfix[i] == "!=")

{

asmCode << "\tfcomp\n";

asmCode << "\tfstsw\tax\n\tsahf\n";

asmCode << "\tje " << postfix[i + 1] << "\n";

i += 2;

}

else if (postfix[i] == "<")

{

asmCode << "\tfcomp\n";

asmCode << "\tfstsw\tax\n\tsahf\n";

if (oper2p == "last" && oper1p != "last")

asmCode << "\tjae " << postfix[i + 1] << "\n";

else

asmCode << "\tjbe " << postfix[i + 1] << "\n";

i += 2;

}

else if (postfix[i] == "=")

asmCode << "\tfistp\t" << oper1p << "\n";

Stack.push("last");

}

}

while (!Stack.empty()) { Stack.pop(); }

}

fpostfix.close();

fasm << ".data\n";

for (int i = 0; i < la.var\_table.table.size(); i++)

{

if (la.var\_table.table[i].value == -1 || la.var\_table.table[i].value == -2)

fasm << "\t" << la.var\_table.table[i].name << "\t\tdd\t?\n";

else

fasm << "\t" << la.var\_table.table[i].name << "\t\tdd\t" <<

la.var\_table.table[i].value << "\n";

}

for(int i = 0; i < la.const\_table.table.size() - 1; i++)

if (la.const\_table.table[i].is\_set == false)

fasm << "\tconst\_" << la.const\_table.table[i].name << "\tdd\t" <<

la.const\_table.table[i].name << "\n";

fasm << "\n.code\nmain proc\n";

asmCode << "\tmov\t\teax, 0\n\tret\n";

asmCode << "main endp\n\nend main";

fasm << asmCode.str();

fasm.close();

}

};

***Файл “main.cpp”***

#include <iostream>

#include "SyntaxAnalyzer.h"

#include "CodeGenerator.h"

using namespace std;

int main()

{

LexicalAnalyzer la = LexicalAnalyzer();

la.MakeTokens("prog.txt", "tokens.txt");

la.PrintAllTables("tables");

SyntaxlAnalyzer sa = SyntaxlAnalyzer();

sa.readParseTable("parsingTable.txt");

if (sa.LL1("tokens.txt", "postfix.txt", "postfixSimple.txt", la))

{

la.PrintAllTables("tables");

CodeGenerator cg = CodeGenerator();

cg.CodeToAsm("code.asm", "postfixSimple.txt", la);

la.PrintAllTables("tables");

}

}