МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»

Институт радиоэлектроники и информационных технологий

Кафедра «Прикладная математика»

Дисциплина: «Теория компиляции»

Курсовая работа

Проектирование и реализация систем лексического и

синтаксического анализа программного кода

Выполнил:

Студент группы 20-ПМ-2

Сидягин Ю.В.

Проверил:

Доцент кафедры «Вычислительные

системы и технологии»

Жевнерчук Дмитрий Валерьевич

Нижний Новгород

2022

**Содержание**

[Введение 3](#_Toc104739378)

[Формальная модель задачи 5](#_Toc104739379)

[Спецификация основных процедур и функций 10](#_Toc104739380)

[Реализация 11](#_Toc104739381)

[Описание программы 12](#_Toc104739382)

[Результат 25](#_Toc104739383)

[Заключение 30](#_Toc104739385)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Введение

Любой язык программирования является средством описания алгоритмов для людей и машин. Однако после запуска программы она не начинает выполняться сиюминутно. Выполнение программы компьютером является сложным процессом, включающим в себя синтаксическую проверку программы, перевод ее в машинный код и само исполнение инструкций.

Программные средства, выполняющие трансляцию кода с языка программирования в машинный код, называются компиляторами.

В течении семестра мы узнали, как разрабатывать и реализовывать компиляторы. Мы узнали, что для построения трансляторов широкого диапазона языков и машин могут использоваться всего несколько базовых идей, которые и были реализованы в данной работе

Постановка задачи

1. Составить формальное описание модельного языка программирования с помощью:

 РБНФ;

 диаграмм Вирта;

 формальных грамматик.

. Написать пять примеров программ.

. Составить таблицы лексем и диаграмму состояний с действиями для распознавания и формирования лексем языка.

. По диаграмме с действиями написать функцию сканирования текста входной программы на модельном языке.

. Разработать программное средство, реализующее лексический анализ текста программы на входном языке.

. Реализовать синтаксический анализатор текста программы на модельном языке.

Формальная модель задачи

**Определение понятия «идентификатор» с использованием РБНФ имеет вид**

<идентификатор> ::= <буква> | <идентификатор> <цифра>

<буква>:: =A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |X |Y | Z |a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра>:: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<целое>::= <двоичное> | <восьмеричное> | <десятичное> |

<шестнадцатеричное>

<двоичное>::= {/ 0 | 1 /} (B | b)

<восьмеричное>::= {/ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 /} (O | o)

<десятичное>::= {/ <цифра> /} [D | d]

<шестнадцатеричное>::= <цифра> {<цифра> | A | B | C | D | E | F | a | b || d | e | f} (H | h)

**В соответствии с данными правилами синтаксис модельного языка будет выглядеть следующим образом**

<буква>:: =A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |X |Y | Z |a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<идентификатор>::= <буква> { <буква> | <цифра> }

<число> ::= {/< цифра> /}

<ключевое\_слово>::= begin | for | end | to | %|| ! ||$ | if | then | else | while | readln | writeln | step

<программа>::= «{»{/(<описание>\<оператор>); /} «}»

<описание> ::= <тип><индефикатор>{,<индефикатор>}

<составной>::= «{» <оператор> { ; <оператор> } «}»

<присваивания> ::= <идентификатор> ass <выражение>

<условный> ::= if (<выражение>)<оператор>[else<оператор>]

<фиксированного\_цикла>::= for ( [<выражение>] ; [<выражение>] ;

[<выражение>] ) <оператор>

<условного\_цикла>::= do while <выражение> <оператор> loop

<ввода>::= input (<идентификатор> {пробел <идентификатор>})

<вывода>::= output (<выражение> { пробел <выражение> })

<выражение>::= <операнд>{<операции\_группы\_отношения> <операнд>} <операнд>::= <слагаемое> {<операции\_группы\_сложения> <слагаемое>}

<слагаемое>::= <множитель> {<операции\_группы\_умножения> <множитель>}

<множитель>::= <идентификатор> | <число> | <логическая\_константа> | <унарная\_операция> <множитель> | (<выражение>) <число>::= <целое> | <действительное> <логическая\_константа>::= true | false

**Формальные грамматики**

P -> D2 B

D2 -> readln D1

D1-> D | D1

> % I1 | ! I1

I1 -> I | I1

B -> main S1 return 0;

S1 -> S | S1

S -> begin S1 end | if E S else S | while E S | readln I | writeln E

E -> E1 | E1 != E1 | E1 == E1 | E1 < E1 | E1 <= E1 | E1 > E1 | E1 >= E1

E1 -> T | T = E1 | T-E1 | T||E1

T -> F| F\*T | F/T | F&&T

F -> I | N | L | !F

L -> true | false

I -> C | IC | IR

N -> R | NR

C -> A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |X |Y | Z |a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

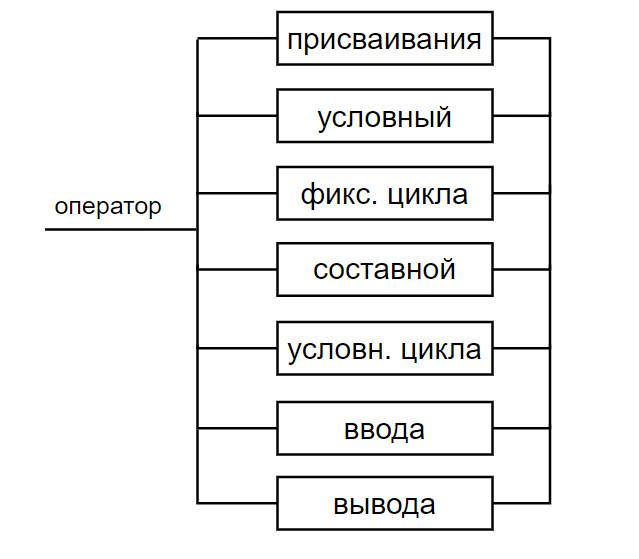
R -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

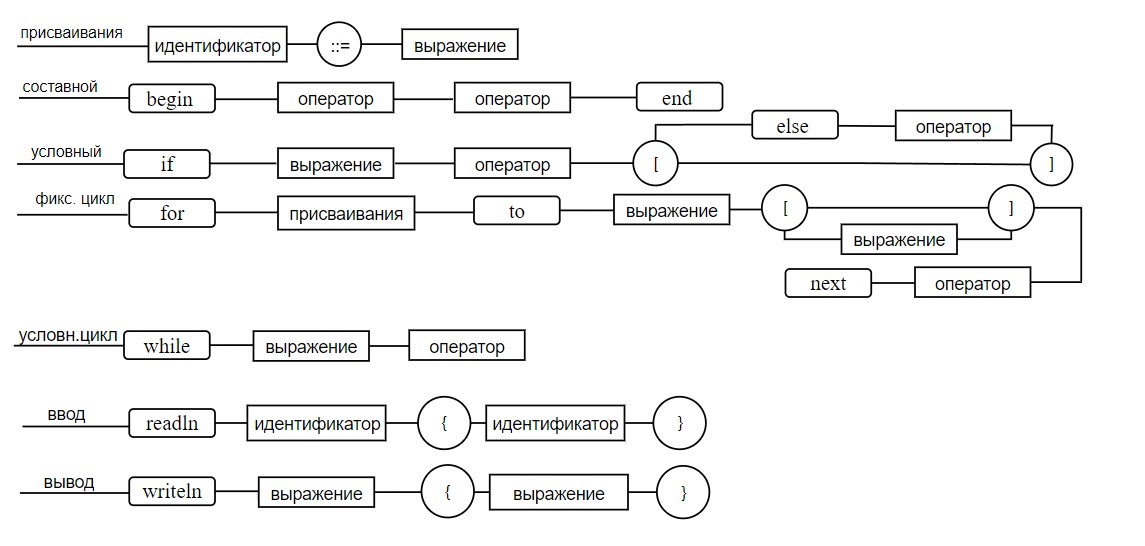
**Описание синтаксиса модельного языка с помощью диаграмм Вирта**











Спецификация основных процедур и функций

программирование компилятор модельный язык

**Лексический анализатор**

Лексический анализатор (ЛА) - это первый этап процесса компиляции, на котором символы, составляющие исходную программу, группируются в отдельные минимальные единицы текста, несущие смысловую нагрузку - лексемы.

**Синтаксический анализатор**

Задача синтаксического анализатора: провести разбор текста программы, сопоставив его с эталоном, данным в описании языка. Для синтаксического разбора используются контекстно-свободные грамматики (КС-грамматика).

Один из эффективных методов синтаксического анализа - метод рекурсивного спуска.

Метод рекурсивного спуска или нисходящий разбор - это один из методов определения принадлежности входной строки к некоторому формальному языку, описанному контекстно-свободной грамматикой (КС-грамматика).

Реализация

Программа была реализована с помощью классов:

Scaner – считывает файл и приводит его к виду строки либо списка

ScanSplitted – обрабатывает считанные данные алфавита языка для дальнейшей работы,

Parser – выполняет разбиение на токены

Compiler – выполняет парсинг путём LL разбора

Описание программы

Далее следует краткое описание принципа работы программы на примере простейшего кода на модельном языке:

if (a GT 5){

output(5);

}

Программа принимает три файла: алфавит языка, грамматика языка и код на модельном языке, который нужно скомпилировать.

Затем файлы, описывающие язык представляются в виде списков, чтобы в дальнейшем легко получать нужные элементы, а файл с кодом в виде строки.

Далее посимвольным сравнением в коде находятся совпадения с операторами, которые выделяются отдельно. Выражения, не попавшие в этот список, помечаются как «разное» - сюда попадают числа и идентификаторы переменных.

wordOperators : [if, output]

operators: [(, GT, ), {, (, ), ;, }]

misc: [a, 5]

Наконец, сортировкой по положению все выражения в коде, включая и операторы, и операнды, получается дерево переходов для кода, после чего находятся несоответствия с правилами грамматики модельного языка. Если все переходы закономерны, программа считается скомпилированной.

Program tree

if

(

a

GT

5

)

{

output

(

5

)

;

}

Compiled

Код

**Classes**

**Word**

#pragma once

#include<string>

using namespace std;

struct Word {

string text;

int index;

string type;

Word(string text, int index, string type) : text(text), index(index), type(type) {

}

};

**Scaner**

#pragma once

#include<stdlib.h>

#include<string>

#include<ctype.h>

#include<iostream>

#include<vector>

#include<fstream>

#include<set>

#include <algorithm>

#include <map>

#include<iomanip>

#pragma warning(disable:4996)

using namespace std;

struct Error {

bool isError;

string error;

};

class Scaner {

private:

vector<string> split(string str, char separator);

Error err;

void resetError();

public:

Scaner();

bool isError();

string getErrorText();

string readFileAsString(string fileName);

vector<vector<string>> readAndSplitFile(string fileName);

};

vector<string> Scaner::split(string str, char separator){

vector<string> result;

str += separator;

int iter = 0;

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

if (str[i] == separator && str != "\n" && str != "") {

result.push\_back(str.substr(iter, i - iter));

iter = i + 1;

}

}

return result;

}

void Scaner::resetError() {

err.isError = false;

err.error = "";

}

Scaner::Scaner() {

this->err.isError = false;

this->err.error = "";

}

bool Scaner::isError() {

return this->err.isError;

}

string Scaner::getErrorText() {

return this->err.error;

}

string Scaner::readFileAsString(string fileName) {

this->resetError();

ifstream file(fileName);

string fileString = "";

if (!file.is\_open()) {

this->err.isError = true;

this->err.error = "error while opening the file\n";

return fileString;

}

char ch;

while (!file.eof()) {

ch = file.get();

if (ch != 9)

fileString += ch;

}

return fileString;

}

vector<vector<string>> Scaner::readAndSplitFile(string fileName) {

this->resetError();

vector<vector<string>> result;

ifstream file(fileName);

if (!file.is\_open()) {

this->err.isError = true;

this->err.error = "error while opening the file\n";

return result;

}

string s;

while (getline(file, s)) {

vector<string>vec = this->split(s, ' ');

result.push\_back(vec);

}

file.close();

return result;

}

**ScanerResult**

#pragma once

#include<stdlib.h>

#include<string>

#include<ctype.h>

#include<iostream>

#include<vector>

#include<fstream>

#include<set>

#include"Source\_Scaner.hpp"

#pragma warning(disable:4996)

class ScanSplitted {

private:

vector<vector<string>> scan;

public:

ScanSplitted(vector<vector<string>> scan);

vector<string> splittedOperators();

vector<string> splittedWordOperators();

};

ScanSplitted::ScanSplitted(vector<vector<string>> scan) : scan(scan) {}

vector<string> ScanSplitted::splittedOperators() {

vector<string> operators;

vector<string> comparisons = scan[0];

vector<string> ass = scan[1];

vector<string> additionGroupOperators = scan[2];

vector<string> multGroupOperators = scan[3];

vector<string> unaryOperators = scan[4];

vector<string> numbers = scan[5];

vector<string> otherOperators = scan[6];

for (string s : comparisons)

operators.push\_back(s);

for (string s : ass)

operators.push\_back(s);

for (string s : additionGroupOperators)

operators.push\_back(s);

for (string s : multGroupOperators)

operators.push\_back(s);

for (string s : unaryOperators)

operators.push\_back(s);

for (string s : otherOperators)

operators.push\_back(s);

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

if (operators[i].size() == 0) {

operators.erase(operators.begin() + i);

i--;

}

}

return operators;

}

vector<string> ScanSplitted::splittedWordOperators() {

vector<string> wordOperators;

vector<string> inOutOperators = scan[7];

vector<string> loopOperators = scan[8];

vector<string> varTypes = scan[9];

for (string s : inOutOperators)

wordOperators.push\_back(s);

for (string s : loopOperators)

wordOperators.push\_back(s);

for (string s : varTypes)

wordOperators.push\_back(s);

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

if (wordOperators[i].size() == 0) {

wordOperators.erase(wordOperators.begin() + i);

i--;

}

}

return wordOperators;

}

**Parser**

#pragma once

#include<stdlib.h>

#include<string>

#include<ctype.h>

#include<iostream>

#include<vector>

#include<fstream>

#include"Source\_Word.hpp"

#pragma warning(disable:4996)

class Parser{

private:

vector<string> operators;

vector<string> wordOperators;

vector<int> isSymbol;

string file;

string getType(string str);

public:

Parser(string file, vector<string> operators, vector<string> wordOperators, vector<int> isSymbol);

vector<Word> operatorsStruct;

vector<Word> wordOperatorsStruct;

vector<Word> miscStruct;

void parseFile();

};

string Parser::getType(string str) {

bool isVariable = true;

bool isNumber = true;

if (str[0] >= '0' && str[0] <= '9') isVariable = false;

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

if (str[i] < '0' || str[i]>'9') isNumber = false;

if (!((str[i] >= '0' && str[i] <= '9') || (str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z'))) isVariable = false;

}

if (isNumber) return "\_number";

if (isVariable) return "\_var";

return str;

}

Parser::Parser(string file, vector<string> operators, vector<string> wordOperators, vector<int> isSymbol) : file(file), operators(operators), wordOperators(wordOperators), isSymbol(isSymbol) {}

void Parser::parseFile() {

for (int j = 0; j < file.size(); j++) {

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

if (file.substr(j, operators[i].size()) == operators[i]) {

operatorsStruct.push\_back(Word(operators[i], j, operators[i]));

for (int k = j; k < j + operators[i].size(); k++) {

isSymbol[k] = 1;

}

j += operators[i].size();

j--;

}

}

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

if (file.substr(j, wordOperators[i].size()) == wordOperators[i]) {

if (!((file[j-1] >= '0' && file[j-1] <= '9') || (file[j-1] >= 'a' && file[j-1] <= 'z') || (file[j-1] >= 'A' && file[j-1] <= 'Z'))){

wordOperatorsStruct.push\_back(Word(wordOperators[i], j, wordOperators[i]));

for (int k = j; k < j + wordOperators[i].size(); k++) {

isSymbol[k] = 1;

}

j += wordOperators[i].size();

j--;

}

}

}

}

string res = "";

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

if (isSymbol[i] == 0 && file[i] != ' ' && file[i] != '\n' && file[i] != '\0' && file[i] != 13) {

res += file[i];

}

else if (res != "") {

miscStruct.push\_back(Word(res, i - res.size(), getType(res)));

res = "";

}

}

if (res != "" && res[0] != -1) {

miscStruct.push\_back(Word(res, isSymbol.size() - res.size(), getType(res)));

}

}

**Compiler**

#pragma once

#include<stdlib.h>

#include<string>

#include<ctype.h>

#include<iostream>

#include<vector>

#include<fstream>

#include <map>

#include"Source\_Word.hpp"

#pragma warning(disable:4996)

class Compiler {

private:

map<string, vector<string>>grammatics;

vector<Word> all;

bool isInclude(const vector<string>& in, const string& str);

public:

Compiler(map<string, vector<string>> grammatics, vector<Word> all);

void compile();

};

bool Compiler::isInclude(const vector<string>& in, const string& str) {

for (const string& sin : in) {

if (sin == str) return true;

}

return false;

}

Compiler::Compiler(map<string, vector<string>> grammatics, vector<Word> all) : grammatics(grammatics), all(all) {}

void Compiler::compile() {

bool isCompiledCorrect = true;

string errorString = "";

for (int i = 1; i < all.size(); i++) {

cout << all[i - 1].text << endl;

string currentOperator = all[i - 1].type;

string nextOperator = all[i].type;

if (!isInclude(grammatics[currentOperator], nextOperator)) {

isCompiledCorrect = false;

errorString = all[i - 1].text + " " + all[i].text;

cout << "ERROR" << endl;

break;

}

}

if (isCompiledCorrect) {

cout << all.back().text << endl;

cout << "Compiled" << endl;

}

else {

cout << "Can't compile on string: " << errorString << endl;

}

}

**Main program**

#include"Source\_Scaner.hpp"

#include"Source\_Word.hpp"

#include"Source\_Parser.hpp"

#include"Source\_ScanSplitted.hpp"

#include"Source\_Compiler.hpp"

#pragma warning(disable:4996)

using namespace std;

#define SEPARATION endl << "=============================="<< endl << endl

int main() {

map<string, vector<string>> grammatics;

Scaner scan;

auto gram = scan.readAndSplitFile("grammatics.txt");

if (scan.isError()) {

cout << scan.getErrorText();

return 1;

}

for (const auto& v : gram) {

grammatics[v[0]] = { v.begin() + 1, v.end() };

}

auto symbols = scan.readAndSplitFile("symbols.txt");

if (scan.isError()) {

cout << scan.getErrorText();

return 1;

}

ScanSplitted scanS(symbols);

vector <string> wordOperators = scanS.splittedWordOperators();

vector <string> operators = scanS.splittedOperators();

auto codeFile = scan.readFileAsString("text.txt");

if (scan.isError()) {

cout << scan.getErrorText();

return 1;

}

vector<int>isSymbol(codeFile.size());

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

isSymbol[i] = 0;

}

cout << "Code from file: " << endl << endl;

Parser parser(codeFile, operators, wordOperators, isSymbol);

parser.parseFile();

vector <Word> operatorsStruct = parser.operatorsStruct;

vector <Word> wordOperatorsStruct = parser.wordOperatorsStruct;

vector <Word> miscStruct = parser.miscStruct;

vector<Word>allExpressions;

cout << codeFile << endl << SEPARATION;

cout << "Word operators: " << endl << endl;

cout << "№" << setw(20) << "OPERATOR" << endl;

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

cout << i+1 << setw(20)<< wordOperatorsStruct[i].text<< endl;

allExpressions.push\_back(wordOperatorsStruct[i]);

}

cout << SEPARATION;

cout << "Operators: " << endl << endl;

cout << "№" << setw(20) << "OPERATOR" << endl;

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

cout << i + 1 << setw(20) << operatorsStruct[i].text<< endl;

allExpressions.push\_back(operatorsStruct[i]);

}

cout << SEPARATION;

cout << "Misc: " << endl << endl;

cout << "№" << setw(20) << "TEXT" << endl;

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

cout << i + 1 << setw(20) << miscStruct[i].text<< endl;

allExpressions.push\_back(miscStruct[i]);

}

cout << SEPARATION;

sort(allExpressions.begin(), allExpressions.end(),

[](const Word& left, const Word& right){

return left.index < right.index;

});

cout << "Programm tree" << endl << endl;

Compiler compilator(grammatics, allExpressions);

compilator.compile();

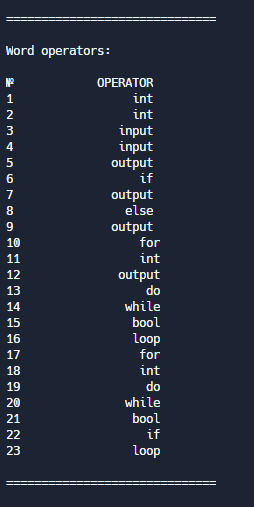
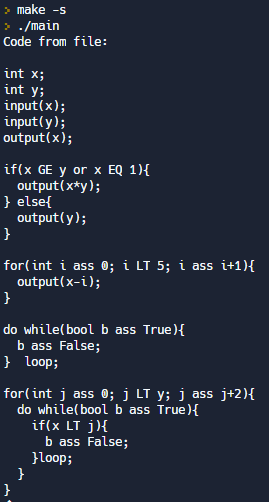
return 0;

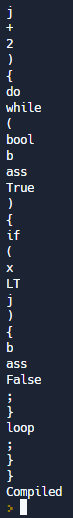
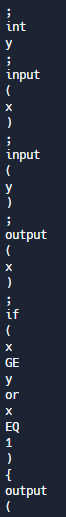
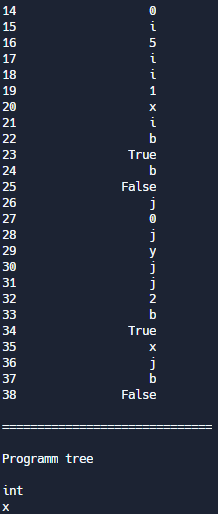
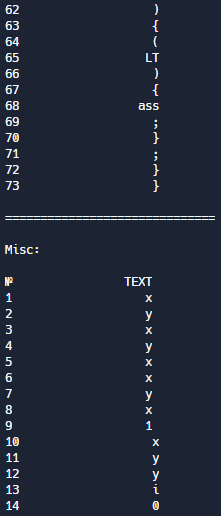
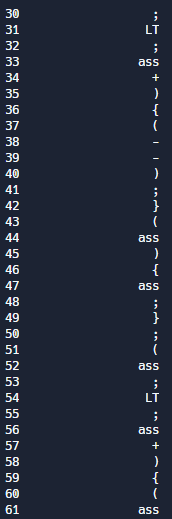
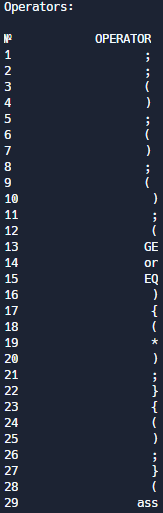
}

##### 

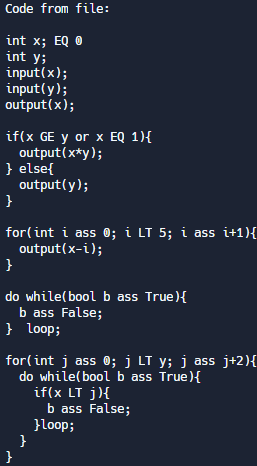
Результат

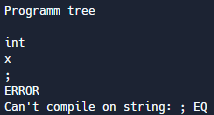
В случае синтаксически правильной программы, поданной на вход, будет выведен ее код, операторы, операнды, а также ее дерево. Успешно пройденная проверка сопровождается строчкой «Compiled».





В случае если в программе присутствуют синтаксические ошибки, дерево программы будет построено лишь до точки ошибки, после чего будет выведено уведомление об ошибке и указана ее точка.





Заключение

Было разработано на языке программирования C++ средство реализующее компилятор модельного языка программирования. Программное средство способно выполнять следующие функции:

ввод и редактирование текста программ, написанных на определенном модельном языке;

производить лексический анализ программ;

выполнять синтаксическую проверку программ;