МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | Факультет прикладной
математики и информатики

Кафедра теоретической и прикладной информатики

Лабораторная работа № 4

по дисциплине «Языки программирования и методы трансляции»

РАЗРАБОТКА И РЕАЛИЗАЦИЯ БЛОКА ГЕНЕРАЦИИ КОДА

Бригада ГОЛУБЬ АНДРЕЙ

БУДАНЦЕВ ДМИТРИЙ

Группа ПМ-13

Вариант 8

Преподаватель ДВОРЕЦКАЯ ВИКТОРИЯ КОНСТАНТИНОВНА

Новосибирск,2024

1. Цель работы

Изучить методы генерации кода с учётом различных промежуточных форм представления программы. Изучить методы управления памятью и особенности их использования на этапе генерации кода. Научиться проектировать генератор кода.

2. Задачи

Подмножество языка C++ включает:

- данные типа **int, float, struct** из элементов указанных типов**;**
- инструкции описания переменных;
- операторы присваивания в любой последовательности;
- операции +, – , <, >, побитовые 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, полученых при изучении курса “Языки программирования и методы трансляции (часть 1)”.

3. Входные и выходные данные

Постоянные таблицы заполняются с помощью текстовых файлов, хранящие ключевые слова, допустимые символы, числа, операции, разделители. На файлы не накладываются дополнительные ограничения, кроме аппаратных ограничений компьютера.

Переменные таблицы заполняются в ходе работы программы, используя в качестве ключа идентификатор элемента или значение константы. Есть возможность изменять данные в процессе работы.

На количество элементов в таблицах накладываются только аппаратные ограничения.

Выходными файлами являются файлы: файл программы на языке Ассемблера, файл ошибок.

4. *Представление выражений на языке ASM*

Выражение на C++	***Выражение на ASM***	***Комментарий***

`struct People {`
`int age = 100 - 32 - 50;`
`float val = 98.23 - 32;`
`}`
	People struct
	age dd 18
	val real4 66.03
	People ends
		Объявление структуры

`int age;`	age dd ?	Объявление переменных
`float val;`	val real4 ?	
`age`	fild age	При участии идентификатора в арифметическом выражении целого типа
`val`	fld val	При участии идентификатора в арифметическом выражении вещественного типа
`age = val`	fld val fistp age	Присваивание при age целого типа val вещественного
`val = age`	fild age fstp val	
`age + val`	fild age fld val fadd	Сложение при age целого типа, val вещественного типа
`age + 1.4`	fild age fld constname1 fadd	
Вычитание аналогично		
`age >> 4`	mov ecx, constname2 shr age, cl	Сдвиг вправо на 4, при age целое
`age << 4`	mov ecx, constname2 shl age, cl	Сдвиг влево на 4, при age целое
Аналогично для выражения с операцией сдвига вправо/влево, где оба операнда целые		
age | num	mov eax, num or age, eax	Побитовый оператор “или” при age и num целым
age & num	mov eax, num and age, eax	Побитовый оператор “и” при age и num целым
age > val	fild age fld val fcomip st(0), st(1) ja greater jb less greater: fild age jmp mark_block_end	Сравнение переменных age целого и val вещественного

	less: fld val mark_block_end:	
val < age	fld val fild age fcomip st(0), st(1) ja greater jb less greater: fild age jmp mark_block_end less: fld val mark_block_end:	
0.1	constname1 dd 0.1	Объявление константы

5. Тесты

Исходная программа	Программа на ASM

struct People {
float strench;
int age = 100;
float val = 98.23;
}
// Hello
void main()
{
int a = 0;
float b = 12.45;
int c;
int d;
c = b >> 4;
d = b | a;
}
struct Point {
float x;
float y;
float z;
}
	.386
	.MODEL FLAT, STDCALL
	OPTION CASEMAP: NONE
	
	EXTRN ExitProcess@4:NEAR
	
	People struct
	strench real4 ?
	age dd 100
	val real4 98.23
	People ends
	
	Point struct
	x real4 ?
	y real4 ?
	z real4 ?
	Point ends
	
	.DATA
	TEMP_VAR1 dd ?
	TEMP_VAR2 dd ?
	TEMP_VAR real4 ?
	a dd ?
	c dd ?
	b real4 ?
	d dd ?

	.CONST
	constname25 dd 100
	constname20 real4 98.23
	constname10 real4 12.45
	constname18 dd 0
	constname22 dd 4
	
	.CODE
	START:
	finit
	fild constname18
	fistp a
	fld constname10
	fstp b
	fld b
	fistp TEMP_VAR1
	fild constname22
	fistp TEMP_VAR2
	mov ecx, TEMP_VAR2
	SHR TEMP_VAR1, cl
	fild TEMP_VAR1
	fistp c
	fld b
	fistp TEMP_VAR1
	fild a
	fistp TEMP_VAR2
	mov eax, TEMP_VAR2
	OR TEMP_VAR1, eax
	fild TEMP_VAR1
	fistp d
	CALL ExitProcess@4
	END START

Файл ошибок: Warning: Using an operation that is performed between integers. In this case, real ones are transmitted. Warning: Using an operation that is performed between integers. In this case, real ones are transmitted.	

struct People {
float strench;
int age = 100;
float val = 98.23;
}
void main()
{
float st = 32 + 54 & 12; // Cooment
int al = 12.1234 << 2;
st = 54 + al;
}
	.386
	.MODEL FLAT, STDCALL
	OPTION CASEMAP: NONE
	
	EXTRN ExitProcess@4:NEAR
	
	People struct
	strench real4 ?
	age dd 100
	val real4 98.23
	People ends

	.DATA
	TEMP_VAR1 dd ?
	TEMP_VAR2 dd ?
	TEMP_VAR real4 ?
	st real4 ?
	al dd ?
	
	.CONST
	constname17 real4 12.1234
	constname9 dd 12
	constname25 dd 100
	constname20 real4 98.23
	constname11 dd 32
	constname15 dd 54
	constname21 dd 2
	
	.CODE
	START:
	finit
	fild constname11
	fild constname15
	fadd
	fistp TEMP_VAR1
	fild constname9
	fistp TEMP_VAR2
	mov eax, TEMP_VAR2
	AND TEMP_VAR1, eax
	fild TEMP_VAR1
	fstp st
	fld constname17
	fistp TEMP_VAR1
	fild constname21
	fistp TEMP_VAR2
	mov ecx, TEMP_VAR2
	SHL TEMP_VAR1, cl
	fild TEMP_VAR1
	fistp al
	fild constname15
	fild al
	fadd
	fstp st
	CALL ExitProcess@4
	END START

Файл ошибок: Warning: Using an operation that is performed between integers. In this case, real ones are transmitted.	

`struct People {`
`float strench;`
`int age = 100;`
`float val = 98.23;`
`}`
	Error: Lack of main structure.

`void main () {`
`struct lambda;`
`lambda = 2 + 3;`
`}`
	Error: In this version of the program there is no processing of the structure type in this constructor!

7. Программа

Translator.h

```
#pragma once
#include "ConstTable.h"
#include "VariableHashTable.h"
#include "Token.h"
#include "Error.h"
#define KEYWORDFILENAME "input_files/Types.txt"
#define SEPARATORFILENAME "input_files/Separator.txt"
#define NUMFILENAME "input_files/Numbers.txt"
#define OPERATIONFILENAME "input_files/Operation.txt"
#define LETTERSFILENAME "input_files/Letters.txt"
#define PARSERFILENAME "input_files/parse_table.txt"
#define NAMEFILEERRORS "out_files/error.txt"
#define NAMEFILETOKEN "out_files/token.txt"
#define NAMEFILEIDENTIFICATORS "out_files/identificators.txt"
#define NAMEFILECONSTANS "out_files/constans.txt"
#define NAMEFILECONSTTABLE "out_files/consttables.txt"
#define NAMEFILEPOSTFIX "out_files/postfix.txt"
#define NAMEASSEMBLERCODE "out_files/asm_code.asm"

#define OPERATION 0
#define NUMBERS 1
#define KEYWORD 2
#define SEPARATORS 3
#define LETTERS 4
#define IDENTIFICATORS 5
#define CONSTANS 6

#ifndef TRANSLATOR

class Translator {
private:
    ConstantTable<string> operation; // 0
    ConstantTable<char> numbers; // 1
    ConstantTable<string> keyword; // 2
    ConstantTable<char> separators; // 3
    ConstantTable<char> letters; // 4
```

```
    VariableHashTable identificators; // 5
    VariableHashTable constans; // 6
    queue<Errors> QErrors;
    queue<Token> QToken;

    int num_line = 0;

    struct table_parser_elem {
        vector<string> terminal;
        int jump = 0;
        bool accept = false;
        bool stack_ = false;
        bool return_ = false;
        bool err = false;
    };
    vector<table_parser_elem> parser_table;
    vector<Token> postfix_buffer;

    void ltrim(string& str) {
        str.erase(0, str.find_first_not_of("\t\n\v\f\r "));
    }
    void rtrim(string& str) {
        str.erase(str.find_last_not_of("\t\n\v\f\r ") + 1);
    }


    void CheckingComments(string& line, bool& continuecheck, bool& nextcheck) {
        size_t ind3 = line.find("*/");
        if (continuecheck) {
            if (ind3 != -1) {
                line.erase(0, ind3 + 2);
                ind3 = line.find("*/");
            }
            else {
                line = "";
                continuecheck = true;
                return;
            }
        }
        size_t ind1 = line.find("//"); size_t ind2 = line.find("/*");
        while (ind1 != -1 || ind2 != -1 || ind3 != -1) {
            if (ind1 != -1 && (ind1 < ind2 || ind2 == -1) && (ind1 < ind3 || ind3
== -1)) {
                line.erase(ind1);
                rtrim(line);
                continuecheck = false;
                return;
            }
            else if (ind2 != -1) {
                if (ind3 == -1) {
                    line.erase(ind2);
```

```
                    rtrim(line);
                    nextcheck = true;
                    continuecheck = false;
                    return;
                }
                else if (ind2 < ind3) {
                    line.erase(ind2, ind3 - ind2 + 2);
                    ind1 = line.find("//"); ind2 = line.find("/*"); ind3 =
line.find("*/");
                    continue;
                }
                else {
                    QErrors.push(Errors(num_line, ERR_BLOCKCOMMENT));
                    line = "";
                    continuecheck = false;
                    return;
                }
            }
            else {
                QErrors.push(Errors(num_line, ERR_BLOCKCOMMENT));
                line = "";
                continuecheck = false;
                return;
            }
        }
        ltrim(line);
        rtrim(line);
        continuecheck = false;
        return;
    }

    bool is_number(const string& s)
    {
        if (s == "-") return false;
        return !s.empty() && (s.find_first_not_of("-0123456789") == s.npos);
    }
    bool is_float(const string& num) {
        return num.find(".") != string::npos;
    }

    bool fill_parser_table() {
        ifstream ParserFile(PARSERFILENAME, ios::in);
        if (!ParserFile.is_open()) return false;
        string temp;
        int num;
        getline(ParserFile, temp);
        while (!ParserFile.eof()) {
            table_parser_elem el;
            ParserFile >> temp;
            while (!is_number(temp)) {
                el.terminal.push_back(temp);
```

```
            ParserFile >> temp;
        }
        el.jump = stoi(temp) - 1;
        ParserFile >> temp;
        el.accept = stoi(temp) == 1;
        ParserFile >> temp;
        el.stack_ = stoi(temp) == 1;
        ParserFile >> temp;
        el.return_ = stoi(temp) == 1;
        ParserFile >> temp;
        el.err = stoi(temp) == 1;
        parser_table.push_back(el);
    }
    ParserFile.close();
}

string buffer_to_string(queue<char> buffer) {
    stringstream ss;
    while (!buffer.empty()) {
        ss << buffer.front();
        buffer.pop();
    }
    return ss.str();
}
void clear_buffer(queue<char>& buffer) {
    while (!buffer.empty()) buffer.pop();
}
Code_Errors LexicalLineScanWithoutComments(string& line) {
    ltrim(line);
    queue<char> buffer;
    /*
    0 - не выбрано
    1 - число
    2 - ключевое слово
    3 - операция
    4 - идентификатор
    */

    /*
    0 - не выбрано
    1 - число
    2 - ключевое слово
    3 - операция
    */
    int temp_mode = 0, num_dot = 0;
    string temp_str;
    char temp_char;
    for (char ch : line) {
        if (ch == ' ' || ch == ';') {
            switch (temp_mode)
            {
```

```
                case 0:
                    if (ch == ';')
                        QToken.push(Token(3, separators.getIndex(ch)));
                    continue;
                case 1:
                    temp_str = buffer_to_string(buffer);
                    if (!constans.Contains(temp_str)) {
                        if (!constans.Add(temp_str, "const")) return
ERR_FAILED_ADD_TABLE;
                    }
                    QToken.push(Token(6, constans.getIndex(temp_str)));
                    if (ch == ';') QToken.push(Token(3, separators.getIndex(ch)));
                    temp_mode = 0;
                    num_dot = 0;
                    clear_buffer(buffer);
                    continue;
                case 2:
                    temp_str = buffer_to_string(buffer);
                    if (keyword.Contains(temp_str)) {
                        keyword.Add(temp_str);
                        QToken.push(Token(2, keyword.getIndex(temp_str)));
                        if (ch == ';') QToken.push(Token(3,
separators.getIndex(ch)));
                        temp_mode = 0;
                        clear_buffer(buffer);
                        continue;
                    }
                    else {
                        if (!identificators.Contains(temp_str)) {
                                if (!identificators.Add(temp_str)) return
ERR_FAILED_ADD_TABLE;
                        }
                        QToken.push(Token(5, identificators.getIndex(temp_str)));
                        if (ch == ';') QToken.push(Token(3,
separators.getIndex(ch)));
                        temp_mode = 0;
                        clear_buffer(buffer);
                        continue;
                    }
                case 3:
                    if (ch == ' ') {
                        temp_str = buffer_to_string(buffer);
                        if (operation.Contains(temp_str)) {
                            operation.Add(temp_str);
                            QToken.push(Token(0, operation.getIndex(temp_str)));
                            clear_buffer(buffer);
                            temp_mode = 0;
                            continue;
                        }
                    }
                    return ERR_INVALID_OPERATION;
```

```
                default:
                    break;
                }
            }
            else if (separators.Contains(ch)) {
                if (temp_mode == 2) {
                    temp_str = buffer_to_string(buffer);
                    if (keyword.Contains(temp_str)) {
                        keyword.Add(temp_str);
                        QToken.push(Token(2, keyword.getIndex(temp_str)));
                        QToken.push(Token(3, separators.getIndex(ch)));
                        temp_mode = 0;
                        clear_buffer(buffer);
                        continue;
                    }
                    else {
                        if (!identificators.Contains(temp_str)) {
                            if (!identificators.Add(temp_str)) return
ERR_FAILED_ADD_TABLE;
                        }
                        QToken.push(Token(5, identificators.getIndex(temp_str)));
                        QToken.push(Token(3, separators.getIndex(ch)));
                        temp_mode = 0;
                        clear_buffer(buffer);
                        continue;
                    }
                }
                separators.Add(ch);
                QToken.push(Token(3, separators.getIndex(ch)));
                continue;

            }
            else if (numbers.Contains(ch)) {
                switch (temp_mode)
                {
                case 0:
                    buffer.push(ch);
                    temp_mode = 1;
                    continue;
                case 1:
                    if (buffer.size() == 1) {
                        temp_char = buffer.front();
                        if (temp_char != '0') {
                            numbers.Add(ch);
                            buffer.push(ch);
                            continue;
                        }
                        else {
                            return ERR_INVALID_INT_NUM;
                        }
                    }
```

```
            else {
                numbers.Add(ch);
                buffer.push(ch);
                continue;
            }
        case 2:
            numbers.Add(ch);
            buffer.push(ch);
            continue;
        case 3:
            return ERR_INVALID_SYMBOL;
        default:
            break;
        }
    }
    else if (letters.Contains(ch)) {
        letters.Add(ch);
        switch (temp_mode)
        {
        case 0:
            buffer.push(ch);
            temp_mode = 2;
            continue;
        case 1:
            return ERR_INVALID_INT_NUM;
        case 2:
            buffer.push(ch);
            continue;
        case 3:
            return ERR_INVALID_OPERATION;
        default:
            break;
        }
    }
    else {
        if (temp_mode == 0) {
            if (operation.Contains(to_string(ch))) {
                operation.Add(to_string(ch));
                QToken.push(Token(0, operation.getIndex(to_string(ch))));
                continue;
            }
            temp_mode = 3;
            buffer.push(ch);
            continue;
        }
        else {
            if (temp_mode != 3) {
                if (ch == '.' && temp_mode == 1) {
                    if (num_dot == 0) {
                        buffer.push(ch);
                        num_dot++;
```

```
                        continue;
                    }
                    else {
                        return ERR_INVALID_FLOAT_NUM;
                    }
                }
                return ERR_INVALID_NAME;
            }
            else {
                buffer.push(ch);
                temp_str = buffer_to_string(buffer);
                if (operation.Contains(temp_str)) {
                    operation.Add(temp_str);
                    QToken.push(Token(0, operation.getIndex(temp_str)));
                    clear_buffer(buffer);
                    temp_mode = 0;
                    continue;
                }
                else {
                    return ERR_INVALID_OPERATION;
                }
            }
        }
    }
    return NO_ERROR;
}

string get_token_info(Token& val) {
    switch (val.NUM_TABLE)
    {
    case 0:
        return operation.getElembyIndex(val.INDEX).first;
    case 1:
        return numbers.getElembyIndex(val.INDEX).first + string();
    case 2:
        return keyword.getElembyIndex(val.INDEX).first;
    case 3:
        return separators.getElembyIndex(val.INDEX).first + string();
    case 4:
        return letters.getElembyIndex(val.INDEX).first + string();
    case 5:
        return identificators.getElemIndex(val.INDEX).name;
    case 6:
        return constans.getElemIndex(val.INDEX).name;
    default:
        return "";
    }
}

Errors SentenceAnalysis() {
```

```
        Token fir_token, sec_token;
        stack<int> parser_stack;
        int row = 0;
        string type;
        vector<Token> infix_expr;
        bool postfix = false;
        bool have_type = false;
        queue<Token> CQToken = QToken;
        string name_token;
        bool flag_err = false;

        fir_token = CQToken.front(); CQToken.pop();
        bool end_flag = CQToken.empty();
        sec_token = CQToken.front(); CQToken.pop();
        while (!flag_err) {
            if (row == 0)
                parser_stack.push(0);
            name_token = get_token_info(fir_token);
            if (fir_token.NUM_TABLE == 5 && name_token != "main") name_token =
"var";
            if (fir_token.NUM_TABLE == 6) name_token = "const";

            bool is_terminal = false;
            for (int i = 0; i < parser_table[row].terminal.size() && !is_terminal;
i++) {
                if (parser_table[row].terminal[i] == name_token) is_terminal =
true;
            }
            if (is_terminal) {
                if (end_flag && parser_table[row].accept) break;
                if (parser_table[row].stack_)
                    parser_stack.push(row + 1);
                if (parser_table[row].accept) {
                    if (name_token == "var" && (get_token_info(sec_token) == "=" ||
row == 54)) {
                        postfix = true;
                        if (get_token_info(sec_token) == "=")
                            identificators.SetValue(get_token_info(fir_token),
true);
                    }
                    if (!have_type && name_token == "var" &&
identificators.getElemIndex(fir_token.INDEX).type == "") {
                        return Errors(ERR_UNKNOWN_VARIABLE,
parser_table[row].terminal, get_token_info(fir_token));
                    }
                    if (postfix)
                        infix_expr.push_back(fir_token);
                    if (name_token == ";" || name_token == ",") {
                        if (!make_postfix(infix_expr))
                            flag_err = true;
                        infix_expr.clear();
```

```
                    postfix = false;
                }
                if (name_token == ";")
                    have_type = false;
                if (name_token == "int" || name_token == "float" || name_token
== "struct") {
                    have_type = true;
                    type = name_token;
                }
                if (name_token == "var" && have_type && (row == 54 || row == 31
|| row == 18))
                    identificators.SetType(get_token_info(fir_token), type);
                if (name_token == "var" && row == 17)
                    postfix_buffer.push_back(fir_token);
                if (name_token == "main" && row == 6)
                    postfix_buffer.push_back(fir_token);
                if ((name_token == "{" || name_token == "}") && (row == 10 ||
row == 12 || row == 18 || row == 20))
                    postfix_buffer.push_back(fir_token);
                end_flag = CQToken.empty();
                fir_token = sec_token;
                if (!end_flag) {
                    sec_token = CQToken.front(); CQToken.pop();
                }
            }
            if (parser_table[row].return_) {
                if (!parser_stack.empty()) {
                    row = parser_stack.top();
                    parser_stack.pop();
                }
                else {
                    return Errors(ERR_PARSESTACK);
                }
            }
            else {
                row = parser_table[row].jump;
            }
        }
        else {
            if (parser_table[row].err) {
                return Errors(ERR_UNKNOWN_TERMINAL, parser_table[row].terminal,
get_token_info(fir_token));
            }
            else {
                row++;
            }
        }

    }
    if (!flag_err && parser_stack.size() > 1) {
        return Errors(ERR_PARSESTACK_NO_EMPTY, parser_stack);
```

```
        }
        if (!flag_err && name_token == "}") {
            postfix_buffer.push_back(fir_token);
        }
        return Errors(NO_ERROR);
    }
    bool make_postfix(vector<Token> v) {
        stack<Token> stack_temp;
        bool error_flag = false;
        int index = 0;
        while (index < v.size() && !error_flag) {
            int i;
            for (i = index; i < v.size() && !error_flag && get_token_info(v[i]) !=
";" && get_token_info(v[i]) != ","; i++) {
                string token_text = get_token_info(v[i]);
                if (v[i].NUM_TABLE == 5 || v[i].NUM_TABLE == 6) {
                    postfix_buffer.push_back(v[i]);
                }
                else if (token_text == "(") {
                    stack_temp.push(v[i]);
                }
                else if (token_text == ")")
                {
                    while (!stack_temp.empty() && get_token_info(stack_temp.top())
!= "("
                    {
                        Token tmpstr = stack_temp.top();
                        postfix_buffer.push_back(tmpstr);
                        stack_temp.pop();
                    }
                    if (stack_temp.empty())
                    {
                        // Syntax Error: Unexpected ")"
                        QErrors.push(Errors(ERR_INVALID_SYMBOL));
                        error_flag = true;
                    }
                    else
                    {
                        stack_temp.pop();
                    }
                }
                else if (v[i].NUM_TABLE == 0)
                {
                    while (!stack_temp.empty() && priority_le(token_text,
get_token_info(stack_temp.top())))
                    {
                        Token tmpstr = stack_temp.top();
                        postfix_buffer.push_back(tmpstr);
                        stack_temp.pop();
                    }
                    stack_temp.push(v[i]);
```

```
                }
            }
            if (error_flag)
            {
                postfix_buffer.clear();
                return false;
            }
            else
            {
                while (!stack_temp.empty() &&
                    get_token_info(stack_temp.top()) != "(" &&
get_token_info(stack_temp.top()) != ")")
                {
                    Token tmpstr = stack_temp.top();
                    postfix_buffer.push_back(tmpstr);
                    stack_temp.pop();
                }
                if (!stack_temp.empty())
                {
                    //Syntax Error: Brackets balance error!
                    QErrors.push(Errors(ERR_BRACKET));
                    error_flag = true;
                }
            }
            if (error_flag)
            {
                postfix_buffer.clear();
                return false;
            }
            index = i + 1;
            postfix_buffer.push_back(Token(3, separators.getIndex(';')));
        }
        return true;
    }

    // Печать постфиксной записи в файл и на экран
    void postfix_print(const string& file_tree)
    {
        ofstream out(file_tree);
        for (int i = 0; i < postfix_buffer.size(); i++)
        {
            out << get_token_info(postfix_buffer[i]) << " ";
        }
        out.close();
    }
    bool priority_le(string what, string with_what)
    {
        int pw = 0, pww = 0;
        if (what == "=" || what == "+=" || what == "-=") pw = 10;
        else if (what == "|") pw = 20;
        else if (what == "&") pw = 30;
```

```
        else if (what == "<" || what == ">") pw = 40;
        else if (what == "<<" || what == ">>") pw = 50;
        else if (what == "+" || what == "-") pw = 60;
        else pw = 70;
        if (with_what == "=" || with_what == "+=" || with_what == "-=") pww = 10;
        else if (with_what == "|") pww = 20;
        else if (with_what == "&") pww = 30;
        else if (with_what == "<" || with_what == ">") pww = 40;
        else if (with_what == "<<" || with_what == ">>") pww = 50;
        else if (with_what == "+" || with_what == "-") pww = 60;
        else pww = 70;
        if (pw <= pww) return true;
        return false;
    };

    Errors GenerateCodeAssembler() {
        /*
        .386
        .MODEL FLAT, STDCALL
        OPTION CASEMAP: NONE
        EXTRN ExitProcess@4:NEAR
        <structs>
        .DATA
        TEMP_VAR dd ?
        <data>
        .CONST
        <constants>
        .CODE
            START:
            <code>
            CALL ExitProcess@4
            END START
        */
        ofstream AsmFile(NAMEASSEMBLERCODE);
        if (!AsmFile.is_open())
            return Errors(FATALERR_FAILLEDREADFILE);
        AsmFile << ".386" << endl << ".MODEL FLAT, STDCALL" << endl << "OPTION
CASEMAP: NONE" << endl << endl << "EXTRN ExitProcess@4:NEAR" << endl << endl;
        vector<Token> temp_postfix_buffer = postfix_buffer;
        bool mark_continue = false;
        bool mark_is_structure = false;
        string token_text_1, token_text_2, token_text_3, name_structure;
        int start_structure = 0, ind, start_new_struct;
        size_t temp;
        Token tok1, tok2, tok3;
        lexeme temp_lex;
        Errors temp_err;
        for (ind = 0; ind < postfix_buffer.size(); ind++) {
            token_text_1 = get_token_info(postfix_buffer[ind]);
            if (token_text_1 == "}") {
                mark_continue = false;
```

```
                mark_is_structure = false;
                continue;
            }
            if (mark_continue)
                continue;
            if (!mark_is_structure) {
                if (token_text_1 != "main") {
                    mark_is_structure = true;
                    name_structure = token_text_1;
                    start_structure = ind;
                    ind++;
                }
                else
                    mark_continue = true;
            }
            else {
                AsmFile << name_structure << " struct" << endl;
                tok1 = postfix_buffer[ind];
                token_text_1 = get_token_info(tok1);
                while (token_text_1 != "}" && (ind + 2) < postfix_buffer.size()) {
                    tok2 = postfix_buffer[ind + 1]; tok3 = postfix_buffer[ind + 2];

                    token_text_2 = get_token_info(tok2);
                    if (token_text_2 == ";") {
                        if (tok1.NUM_TABLE == IDENTIFICATORS) {
                            temp_lex = identificators.getElement(token_text_1);
                            if (temp_lex.type == "int") {
                                AsmFile << "\t" << token_text_1 << " dd ?" << endl;
                                identificators.SetStruct(token_text_1, true);
                            }
                            else if (temp_lex.type == "float") {
                                AsmFile << "\t" << token_text_1 << " real4 ?" <<
endl;

                                identificators.SetStruct(token_text_1, true);
                            }
                            else {
                                AsmFile.close();
                                return Errors(ERR_TYPE_STRUCT);
                            }
                            ind = ind + 2;
                            tok1 = postfix_buffer[ind];
                            token_text_1 = get_token_info(tok1);
                            continue;
                        }
                        else {
                            AsmFile.close();
                            return Errors(ERR_INVALID_PREFIX);
                        }
                    }
                    token_text_3 = get_token_info(tok3);
                    if (token_text_3 == "=") {
```

```
                        if (tok1.NUM_TABLE == IDENTIFICATORS) {
                            temp_lex = identificators.getElement(token_text_1);
                            if (temp_lex.type == "int") {
                                if (tok2.NUM_TABLE == CONSTANS) {
                                    if ((temp = token_text_2.find(';')) ==
string::npos) {
                                        AsmFile << "\t" << token_text_1 << " dd "
<< token_text_2 << endl;
                                        identificators.SetStruct(token_text_1,
true);
                                    }
                                    else {
                                        QErrors.push(Errors(WARNING_FLOAT_TO_INT));
                                        token_text_2.erase(token_text_2.begin() +
temp, token_text_2.end());
                                        AsmFile << "\t" << token_text_1 << " dd "
<< token_text_2 << endl;
                                        identificators.SetStruct(token_text_1,
true);
                                    }
                                }
                                else
                                {
                                    AsmFile.close(); return
Errors(ERR_STRUCT_NO_EXPR);
                                }
                            }
                            else if (temp_lex.type == "float") {
                                if (tok2.NUM_TABLE == CONSTANS) {
                                    AsmFile << "\t" << token_text_1 << " real4 " <<
token_text_2 << endl;
                                    identificators.SetStruct(token_text_1, true);
                                }
                                else
                                {
                                    AsmFile.close(); return
Errors(ERR_STRUCT_NO_EXPR);
                                }
                            }
                            else {
                                AsmFile.close();
                                return Errors(ERR_TYPE_STRUCT);
                            }
                        }
                        else {
                            AsmFile.close();
                            return Errors(ERR_INVALID_PREFIX);
                        }
                        if (get_token_info(postfix_buffer[ind + 3]) != ";")
                        {
                            AsmFile.close(); return Errors(ERR_POSTFIX_EXPR_END);
```

```
                    }
                    ind += 4;
                    tok1 = postfix_buffer[ind];
                    token_text_1 = get_token_info(tok1);
                    continue;
                }
                else {
                    AsmFile.close();
                    return Errors(ERR_MULTIPLY_EXPR);
                }
            }
            AsmFile << name_structure << " ends" << endl << endl;
            start_new_struct = find(temp_postfix_buffer.begin(),
temp_postfix_buffer.end(), postfix_buffer[start_structure]) -
temp_postfix_buffer.begin();
            temp_postfix_buffer.erase(temp_postfix_buffer.begin() +
start_new_struct, temp_postfix_buffer.begin() + (ind-start_structure) +
start_new_struct + 1);
            mark_is_structure = false;
        }
    }
    AsmFile << endl << ".DATA" << endl << "\tTEMP_VAR1 dd ?" << endl <<
"\tTEMP_VAR2 dd ?" << endl << "\tTEMP_VAR real4 ?" << endl;
    for (lexeme var : identificators.values()) {
        if (identificators.getValue(var.name) &&
!identificators.getStruct(var.name)) {
            if (var.type == "int") {
                AsmFile << "\t" << var.name << " dd ?" << endl;
            }
            else if (var.type == "float") {
                AsmFile << "\t" << var.name << " real4 ?" << endl;
            }
            else if (var.type == "struct") {
                AsmFile.close();
                return Errors(ERR_TYPE_STRUCT);
            }
            else {
                AsmFile.close();
                return Errors(ERR_INVALID_TYPE_VAR);
            }
        }
    }
    AsmFile << endl << ".CONST" << endl;
    for (lexeme num : constans.values()) {
        if (is_float(num.name))
            AsmFile << "\tconstname" << to_string(constans.getIndex(num.name))
<< " real4 " << num.name << endl;
        else
            AsmFile << "\tconstname" << to_string(constans.getIndex(num.name))
<< " dd " << num.name << endl;
    }
```

```
        AsmFile << endl << ".CODE" << endl << "\tSTART:" << endl << "\tfinit" <<
endl;
        if (temp_postfix_buffer.size() > 2) {
            for (ind = 2, tok3 = temp_postfix_buffer[ind], token_text_3 =
get_token_info(tok3); ind+1 < temp_postfix_buffer.size(); ind++) {
                if (tok3.NUM_TABLE == OPERATION) {
                    if (ind - 2 < 2) {
                        AsmFile.close(); return Errors(ERR_INVALID_PREFIX);
                    }
                    tok1 = temp_postfix_buffer[ind - 2]; tok2 =
temp_postfix_buffer[ind - 1];
                    if (token_text_3 == "+") {
                        if ((temp_err = float_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) {
                            AsmFile.close(); return temp_err;
                        }
                        AsmFile << "\tfadd" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                    else if (token_text_3 == "-") {
                        if ((temp_err = float_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) {
                            AsmFile.close(); return temp_err;
                        }
                        AsmFile << "\tfsub" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                    else if (token_text_3 == "|") {
                        if ((temp_err = int_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) {
                            AsmFile.close(); return temp_err;
                        }
                        AsmFile << "\tmov eax, TEMP_VAR2" << endl << "\tOR
TEMP_VAR1, eax" << endl << "\tfild TEMP_VAR1" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
```

```
                    else if (token_text_3 == "&") {
                        if ((temp_err = int_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) {
                            AsmFile.close(); {
                                AsmFile.close(); return temp_err;
                            }
                        }
                        AsmFile << "\tmov eax, TEMP_VAR2" << endl << "\tAND
TEMP_VAR1, eax" << endl << "\tfild TEMP_VAR1" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                    else if (token_text_3 == ">>") {
                        if ((temp_err = int_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) { AsmFile.close(); return temp_err; }
                        AsmFile << "\tmov ecx, TEMP_VAR2" << endl << "\tSHR
TEMP_VAR1, cl" << endl << "\tfild TEMP_VAR1" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                    else if (token_text_3 == "<<") {
                        if ((temp_err = int_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) {
                            AsmFile.close(); return temp_err;
                        }
                        AsmFile << "\tmov ecx, TEMP_VAR2" << endl << "\tSHL
TEMP_VAR1, cl" << endl << "\tfild TEMP_VAR1" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                    else if (token_text_3 == "<") {
                        if ((temp_err = float_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) {
                            AsmFile.close(); return temp_err;
                        }
                        AsmFile << "\tfcomip st(0), st(1)" << endl << "\tja
greater" << endl << "\tjb less" << endl << "\tgreater:" << endl << "\tfild 0" <<
endl <<
```

```
                            "\tjmp mark_block_end" << endl << "\tless:" << endl <<
"\tfild 1" << endl << "\tmark_block_end:" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                    else if (token_text_3 == ">") {
                        if ((temp_err = float_oper(AsmFile, tok1, tok2)).CODE !=
NO_ERROR) {
                            AsmFile.close();
                            return temp_err;
                        }
                        AsmFile << "\tfcomip st(0), st(1)" << endl << "\tja
greater" << endl << "\tjb less" << endl << "\tgreater:" << endl << "\tfild 1" <<
endl <<
                            "\tjmp mark_block_end" << endl << "\tless:" << endl <<
"\tfild 0" << endl << "\tmark_block_end:" << endl;
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 1, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        temp_postfix_buffer[ind].NUM_TABLE = NUMBERS;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                    else if (token_text_3 == "=") {
                        if ((temp_err = float_var_asm(name_structure, tok2)).CODE
!= NO_ERROR) return temp_err;
                        AsmFile << name_structure;
                        if (tok1.NUM_TABLE != IDENTIFICATORS) {
                            AsmFile.close();
                            return Errors(ERR_INVALID_PREFIX);
                        }
                        temp_lex = identificators.getElemIndex(tok1.INDEX);
                        if (temp_lex.type == "float") {
                            AsmFile << "\tfstp " << temp_lex.name << endl;
                        }
                        else if (temp_lex.type == "int") {
                            AsmFile << "\tfistp " << temp_lex.name << endl;
                        }
                        else if (temp_lex.type == "struct") {
                            AsmFile.close();
                            return Errors(ERR_TYPE_STRUCT);
                        }
                        else {
                            AsmFile.close();
                            return
Errors(ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR);
                        }
```

```
                        temp_postfix_buffer.erase(temp_postfix_buffer.begin() + ind
- 2, temp_postfix_buffer.begin() + ind + 1);
                        ind -= 2;
                        tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                    }
                }
                else {
                    tok3 = temp_postfix_buffer[ind + 1]; token_text_3 =
get_token_info(tok3);
                }
            }
            AsmFile << "\tCALL ExitProcess@4" << endl;
            AsmFile << "\tEND START" << endl;
            AsmFile.close();
            return Errors(NO_ERROR);
        }
        else {
            AsmFile.close();
            return Errors(ERR_LACK_MAIN);
        }

    }

    Errors float_oper(ofstream & AsmFile, Token tok1, Token tok2) {
        lexeme temp_lex;
        string token_text_1, token_text_2;
        Errors temp;
        if ((tok1.NUM_TABLE == IDENTIFICATORS || tok1.NUM_TABLE == CONSTANS) &&
(tok2.NUM_TABLE == IDENTIFICATORS || tok2.NUM_TABLE == CONSTANS)) {
            if ((temp = float_var_asm(token_text_1, tok1)).CODE != NO_ERROR) return
temp;
            AsmFile << token_text_1;
            if ((temp = float_var_asm(token_text_1, tok2)).CODE != NO_ERROR) return
temp;
            AsmFile << token_text_1;
        }
        else if (tok1.NUM_TABLE == NUMBERS && (tok2.NUM_TABLE == IDENTIFICATORS ||
tok2.NUM_TABLE == CONSTANS)) {
            if ((temp = float_var_asm(token_text_1, tok2)).CODE != NO_ERROR) return
temp;
            AsmFile << token_text_1;
        }
        else if ((tok1.NUM_TABLE == IDENTIFICATORS || tok1.NUM_TABLE == CONSTANS)
&& tok2.NUM_TABLE == NUMBERS) {
            AsmFile << "\tfstp TEMP_VAR" << endl;
            if ((temp = float_var_asm(token_text_1, tok1)).CODE != NO_ERROR) return
temp;
            AsmFile << token_text_1;
            AsmFile << "\tfld TEMP_VAR" << endl;
        }
```

```
        return Errors(NO_ERROR);
    }
    Errors int_oper(ofstream& AsmFile, Token tok1, Token tok2) {
        lexeme temp_lex;
        string token_text_1, token_text_2;
        Errors temp;
        if ((tok1.NUM_TABLE == IDENTIFICATORS || tok1.NUM_TABLE == CONSTANS) &&
(tok2.NUM_TABLE == IDENTIFICATORS || tok2.NUM_TABLE == CONSTANS)) {
            if ((temp = int_var_asm(token_text_1, tok1, "TEMP_VAR1")).CODE !=
NO_ERROR) return temp;
            AsmFile << token_text_1;
            if ((temp = int_var_asm(token_text_1, tok2, "TEMP_VAR2")).CODE !=
NO_ERROR) return temp;
            AsmFile << token_text_1;
        }
        else if (tok1.NUM_TABLE == NUMBERS && (tok2.NUM_TABLE == IDENTIFICATORS ||
tok2.NUM_TABLE == CONSTANS)) {
            AsmFile << "\tfistp TEMP_VAR1" << endl;
            if ((temp = int_var_asm(token_text_1, tok2, "TEMP_VAR2")).CODE !=
NO_ERROR) return temp;
            AsmFile << token_text_1;
        }
        else if ((tok1.NUM_TABLE == IDENTIFICATORS || tok1.NUM_TABLE == CONSTANS)
&& tok2.NUM_TABLE == NUMBERS) {
            AsmFile << "\tfistp TEMP_VAR2" << endl;
            if ((temp = int_var_asm(token_text_1, tok1, "TEMP_VAR1")).CODE !=
NO_ERROR) return temp;
            AsmFile << token_text_1;
        }
        else if (tok1.NUM_TABLE == NUMBERS && tok2.NUM_TABLE == NUMBERS) {
            AsmFile << "\tfistp TEMP_VAR2" << endl;
            AsmFile << "\tfistp TEMP_VAR1" << endl;
        }
        return Errors(NO_ERROR);
    }
    Errors float_var_asm(string& str, Token tok1) {
        lexeme temp_lex;
        stringstream ss;
        if (tok1.NUM_TABLE == IDENTIFICATORS) {
            temp_lex = identificators.getElemIndex(tok1.INDEX);
            if (identificators.getValueIndex(tok1.INDEX)) {
                temp_lex.value = true;
            }
            if (temp_lex.value) {
                if (temp_lex.type == "int") {
                    ss << "\tfild " << temp_lex.name << endl;
                }
                else if (temp_lex.type == "float") {
                    ss << "\tfld " << temp_lex.name << endl;
                }
                else if (temp_lex.type == "struct")
```

```
                    return Errors(ERR_TYPE_STRUCT);
                else return Errors(ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR);
            }
            else
                return Errors(ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR);
        }
        else if (tok1.NUM_TABLE == CONSTANS) {
            string temp = get_token_info(tok1);
            if (is_float(temp)) {
                ss << "\tfld " << "constname" << to_string(tok1.INDEX) << endl;
            }
            else {
                ss << "\tfild " << "constname" << to_string(tok1.INDEX) << endl;
            }
        }
        str = ss.str();
        return Errors(NO_ERROR);
    }
    Errors int_var_asm(string& str, Token tok1, string to) {
        lexeme temp_lex;
        stringstream ss;
        if (tok1.NUM_TABLE == IDENTIFICATORS) {
            temp_lex = identificators.getElemIndex(tok1.INDEX);
            if (identificators.getValueIndex(tok1.INDEX)) {
                if (temp_lex.type == "int") {
                    ss << "\tfild " << temp_lex.name << endl << "\tfistp "<< to <<
endl;
                }
                else if (temp_lex.type == "float") {
                    QErrors.push(Errors(WARNING_FLOAT_TO_INT_PERFORMING));
                    ss << "\tfld " << temp_lex.name << endl << "\tfistp " << to <<
endl;
                }
                else if (temp_lex.type == "struct")
                    return Errors(ERR_TYPE_STRUCT);
                else return Errors(ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR);
            }
            else
                return Errors(ERR_PERFORMING_OPERATION_UNINITIALIZED_VAR);
        }
        else if (tok1.NUM_TABLE == CONSTANS) {
            string temp = get_token_info(tok1);
            if (is_float(temp)) {
                QErrors.push(Errors(WARNING_FLOAT_TO_INT_PERFORMING));
                ss << "\tfld " << "constname" << to_string(tok1.INDEX) << endl <<
"\tfistp " << to << endl;
            }
            else {
                ss << "\tfild " << "constname" << to_string(tok1.INDEX) << endl <<
"\tfistp " << to << endl;
            }
```

```
        }
        str = ss.str();
        return Errors(NO_ERROR);
    }

public:
    Translator() {
        if (!operation.ReadKeysFile(OPERATIONFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!numbers.ReadKeysFile(NUMFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!keyword.ReadKeysFile(KEYWORDFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!separators.ReadKeysFile(SEPARATORFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!letters.ReadKeysFile(LETTERSFILENAME)) QErrors.push(Errors(0,
FATALERR_FAILLEDREADFILE));
        if (!fill_parser_table()) QErrors.push(Errors(FATALERR_FAILLEDREADFILE));
    }

    void PrintToken() {
        cout << "TABLE" << "\t" << "NUM_TABLE" << endl;
        queue<Token> copy_token = QToken;
        Token temp_token;
        while (!copy_token.empty()) {
            temp_token = copy_token.front();
            copy_token.pop();
            cout << temp_token.NUM_TABLE << "\t" << temp_token.INDEX << endl;
        }
    }
    void PrintErrors() {
        cout << "NUM LINE" << "\t" << "CODE ERROR" << endl;
        queue<Errors> copy_Error = QErrors;
        Errors temp_error;
        while (!copy_Error.empty()) {
            temp_error = copy_Error.front();
            copy_Error.pop();
            cout << temp_error.NUM_LINE << "\t" << temp_error.CODE << endl;
        }
    }
    void ScanningFile(const string& NameFile) {
        if (operation.empty() || numbers.empty() || keyword.empty() ||
letters.empty() || separators.empty() || parser_table.empty()) return;

        ifstream ScanFile(NameFile, ios::in);
        if (!ScanFile.is_open()) {
            QErrors.push(Errors(0, FATALERR_FAILLEDREADSCANFILE));
            return;
        }

        string line;
```

```
bool flag_err = false;
bool check = false, nextcheck = false;
Code_Errors code;
while (getline(ScanFile, line)) {
    if (nextcheck) {
        check = true;
        nextcheck = false;
    }
    CheckingComments(line, check, nextcheck);
    if (check == true || line.empty()) {
        num_line++;
        continue;
    }
    if ((code = LexicalLineScanWithoutComments(line)) != NO_ERROR) {
        QErrors.push(Errors(num_line, code));
        flag_err = true;
        break;
    }
    num_line++;
}
ScanFile.close();
if (!flag_err) {
    Errors err = SentenceAnalysis();
    if (err.CODE == NO_ERROR) {
        postfix_print(NAMEFILEPOSTFIX);
    }
    else {
        QErrors.push(err);
    }
}
if (!flag_err) {
    Errors err = GenerateCodeAssembler();
    if (err.CODE != NO_ERROR) {
        QErrors.push(err);
    }
}
if (check || nextcheck) QErrors.push(Errors(num_line, ERR_BLOCKCOMMENT));
if (!outFileToken()) {
    cerr << NAMEFILETOKEN << " file not opened!";
}
if (!outFileErrors()) {
    cerr << NAMEFILEERRORS << " file not opened!";
    return ;
}
if (!outFileIC()) {
    cerr << NAMEFILEIDENTIFICATORS << " file not opened!";
    return ;
}
if (!outFileC()) {
    cerr << NAMEFILECONSTANS << " file not opened!";
    return ;
```

```
    }
    if (!outConstTable()) {
        cerr << NAMEFILECONSTTABLE << " file not opened!";
        return ;
    }
}



bool outFileToken() {
    ofstream FileOut(NAMEFILETOKEN);
    if (!FileOut.is_open()) {
        cerr << "Error! File not opened!" << endl;
        return false;
    }
    queue<Token> CQtoken = QToken;
    Token t_token;
    while (!CQtoken.empty()) {
        t_token = CQtoken.front();
        CQtoken.pop();
        FileOut << t_token.NUM_TABLE << "\t" << t_token.INDEX << endl;
    }
    FileOut.close();
    return true;
}
bool outFileErrors() {
    ofstream FileOut(NAMEFILEERRORS);
    if (!FileOut.is_open()) {
        cerr << "Error! File not opened!" << endl;
        return false;
    }
    queue<Errors> CQErrors = QErrors;
    Errors t_error;
    while (!CQErrors.empty()) {
        t_error = CQErrors.front();
        CQErrors.pop();
        FileOut << t_error.info() << endl;
    }
    FileOut.close();
    return true;
}
bool outFileIC() {
    ofstream FileOut(NAMEFILEIDENTIFICATORS);
    return identificators.outFile(FileOut);
    FileOut.close();
}
bool outFileC() {
    ofstream FileOut(NAMEFILECONSTANS);
    return constans.outFile(FileOut);
    FileOut.close();
}
```

```
    bool outConstTable() {
        ofstream FileOut(NAMEFILECONSTTABLE);
        if (!FileOut.is_open()) {
            cerr << "Error! File not opened!" << endl;
            return false;
        }
        letters.outfile(FileOut);
        keyword.outfile(FileOut);
        operation.outfile(FileOut);
        separators.outfile(FileOut);
        numbers.outfile(FileOut);
        FileOut.close();
        return true;
    }
};

#endif // !TRANSLATOR
```