МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

# Лабораторная работа №8

по дисциплине: Объектно-ориентированное программирование

тема: «**Создание шаблонов классов в C++**»

Выполнил: ст. группы ПВ-233

Ситников Алексей Павлович

Проверил:

Белгород 2025 г.

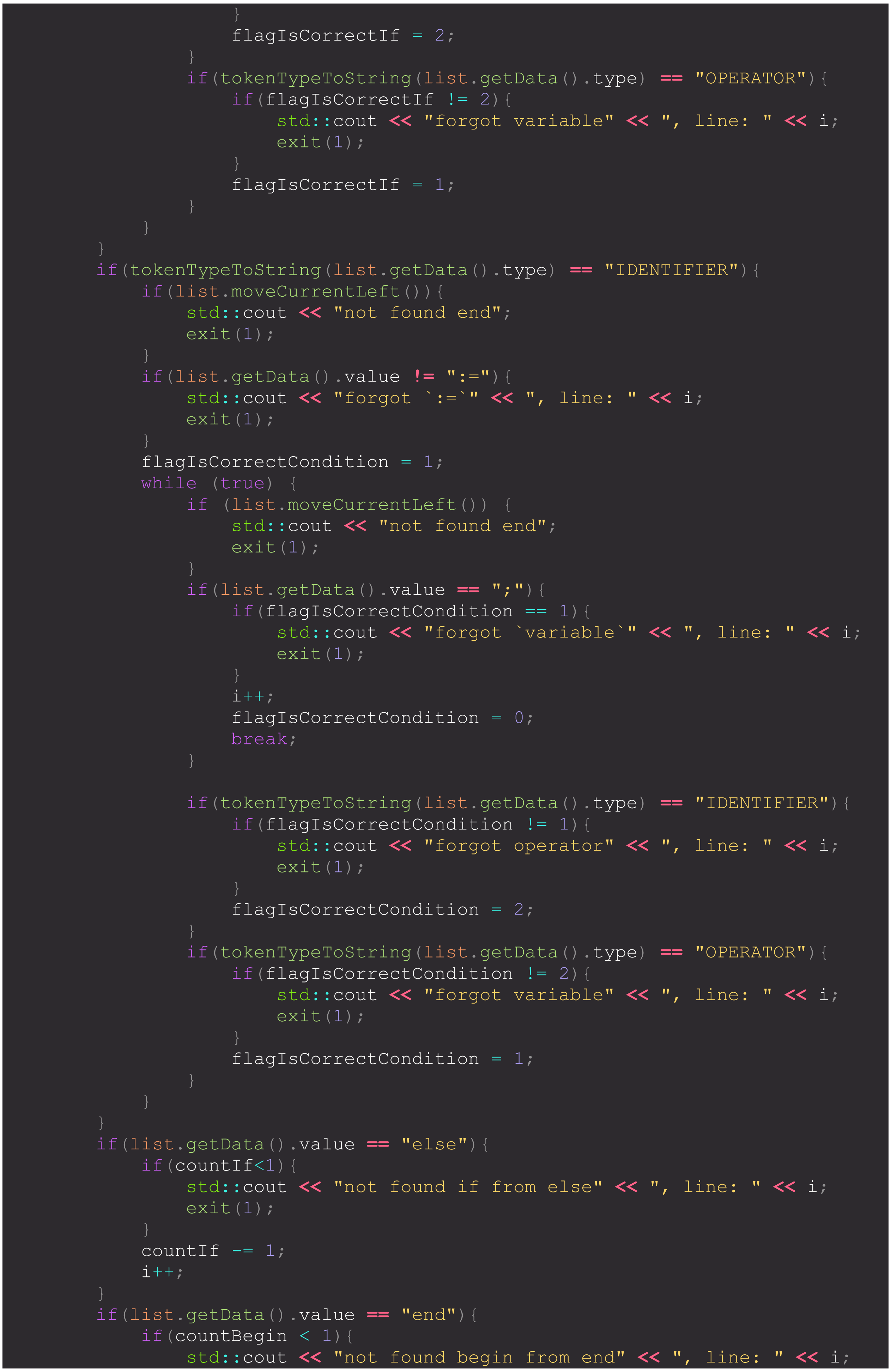
Вариант 3 (13)

**Цель работы:** Получение теоретических знаний о шаблонах классов в C++. Получение практических навыков по созданию классов-шаблонов C++.

Двусвязный список:

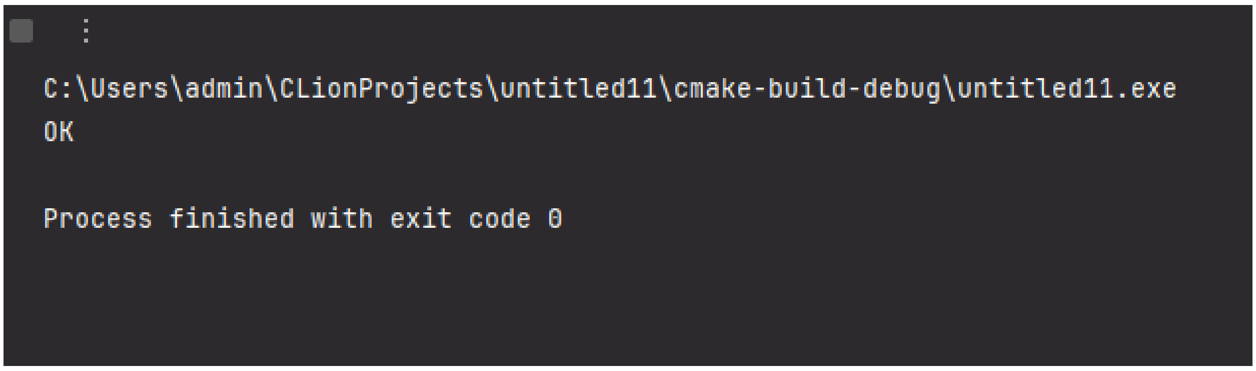
```
#include <iostream>
#include <sstream>
#include <windows.h>

#include "Dlist.h"

enum TokenType_ {
    KEYWORD,
    IDENTIFIER,
    NUMBER,
    OPERATOR,
    DELIMITERS,
    STRINGLITERALS,
    COMMENTS,
    SEMICOLON,
    TYPE
};

struct Token {
    TokenType_ type;
    std::string value;
};


int dataInArray(std::string value, Dlist<std::string> &arr){
    arr.setRight();
    while (true){
        if(value == arr.getData()){
            return 1;
        }
        if(arr.moveCurrentLeft()){
            return 0;
        }
    }

}


void lex(const std::string& code, Dlist<Token> &list) {
    std::istringstream stream(code);
    std::string word;
    std::string keywords[] = {"program", "var", "begin", "end", "if", "then",
"else", "while", "do", "for", "to", "downto", "procedure", "function",
"array", "record", "case", "of", "repeat", "until", "with", "not", "and",
"or"};
    std::string operators[] = {"+", "-", "*", "/", ":=", "=", "<", ">", "<=",
">=", "<>", "and", "or", "not"};
    std::string limiters[] = {";", ",", ".", "(", ")", "[", "]"};
    std::string type[] = {"integer", "real", "char", "boolean", "string",
"array", "record", "file", "pointer", "set", "variant", "enumerated"};
    Dlist<std::string> keywordsDlist;
    Dlist<std::string> operatorsDlist;
```

```cpp
    Dlist<std::string> limitersDlist;
    Dlist<std::string> typeDlist;
    keywordsDlist.creatFromArray(keywords, 24);
    operatorsDlist.creatFromArray(operators, 14);
    limitersDlist.creatFromArray(limiters, 7);
    typeDlist.creatFromArray(type, 12);

    while (stream >> word) {
        Token token;
        if (dataInArray(word, keywordsDlist)) {
            token.type = KEYWORD;
        } else if (std::isdigit(word[0])) {
            token.type = NUMBER;
        } else if (dataInArray(word, operatorsDlist)) {
            token.type = OPERATOR;
        } else if (dataInArray(word, limitersDlist)) {
            token.type = DELIMITERS;
        } else if (word == ";") {
            token.type = SEMICOLON;
        } else if(word[0] == '\'' ){
            std::string temp;
            while (true){
                stream >> temp;
                word+= ' ' + temp;
                if(word[word.size()-1] == '\''){
                    break;
                }
            }
            token.type = STRINGLITERALS;
        }
        else if(dataInArray(word,typeDlist)){
            token.type = TYPE;
        }
        else if(word == "//"){
            token.type = COMMENTS;
            std::string temp;
            while (true){
                if(word == "\n"){
                    break;
                }
                stream >> temp;
                word+=temp;
            }
        }
        else if(word == "{"){
            token.type = COMMENTS;
            std::string temp;
            while (true){
                stream >> temp;
                word+=temp;
                if(word == "}"){
                    break;
                }
            }
        }
        else if(word == "(*"){
            token.type = COMMENTS;
            std::string temp;
            while (true){
                stream >> temp;
                word+=temp;
                if(word == "*)"){
                    break;
                }
            }
        }
        else {
            token.type = IDENTIFIER;
        }
        token.value = word;
        list.appendLeft(token);
    }
}



std::string tokenTypeToString(TokenType_ type) {
    switch (type) {
        case KEYWORD: return "KEYWORD";
        case IDENTIFIER: return "IDENTIFIER";
        case NUMBER: return "NUMBER";
        case OPERATOR: return "OPERATOR";
        case DELIMITERS: return "DELIMITERS";
        case STRINGLITERALS: return "STRINGLITERALS";
        case COMMENTS: return "COMMENTS";
        case SEMICOLON: return "SEMICOLON";
        case TYPE: return "TYPE";
        default: return "UNKNOWN";
    }
}

void parser(Dlist<Token> &list){
    list.setRight();
    int i = 1;
    int countIf = 0;
    int countBegin = 0;
    while (true){
        int flagDeclaration;
        int FlagInit;

        if(list.getData().value == "var"){
            std::string t4 = list.getData().value;
            if(list.moveCurrentLeft()){
                std::cout << "not found end";
                exit(1);
            }
            std::string t = list.getData().value;
            if(list.moveCurrentLeft()){
                std::cout << "not found end";
                exit(1);
            }
            flagDeclaration = 2;
            if(list.getData().value == ":="){
                FlagInit = 1;
                while (true) {
                    if (list.moveCurrentLeft()) {
                        std::cout << "not found end";
                        exit(1);
                    }
                    if (list.getData().value == ";") {
                        if (FlagInit != 4) {
                            std::cout << "forgot `variable`" << ", line: " <<
i;

                            exit(1);
                        }
                        flagDeclaration = 0;
                        FlagInit = 0;
                        i++;
                        break;
                    }
                    if(list.getData().value == "("){
                        FlagInit = 3;
                    }
                    if(list.getData().value == ")") {
                        FlagInit = 4;
                    }
                    if(tokenTypeToString(list.getData().type) ==
"IDENTIFIER"){
                        FlagInit = 4;
                    }
                }
            }
            else {
                while (true) {
                    if (list.getData().value == ";") {
                        if (flagDeclaration != 3) {
                            std::cout << "forgot `type`" << ", line: " << i;
                            exit(1);
                        }
                        flagDeclaration = 0;
                        i++;
                        break;
                    }

                    if (list.getData().value == ":") {
                        if (flagDeclaration == 1) {
                            std::cout << "forgot `variable`" << ", line: " <<
i;

                            exit(1);
                        }
                        if (list.moveCurrentLeft()) {
                            std::cout << "not found end";
                            exit(1);
                        }
                        if (tokenTypeToString(list.getData().type) != "TYPE")
{
                            std::cout << "forgot `type`" << ", line: " << i;
                            exit(1);
                        }
                        flagDeclaration = 3;
                    } else if (list.getData().value == ",") {
                        if (flagDeclaration == 1) {
                            std::cout << "forgot `variable`" << ", line: " <<
i;

                            exit(1);
                        }
                        flagDeclaration = 1;
                    } else if (tokenTypeToString(list.getData().type) ==
"IDENTIFIER") {
                        if (flagDeclaration == 2) {
                            std::cout << "forgot `,`" << ", line: " << i;
                            exit(1);
                        }
                        flagDeclaration = 2;
                    }
                    if (list.moveCurrentLeft()) {
                        std::cout << "not found end";
                        exit(1);
                    }
                }
            }
        }
        if(list.getData().value == "const"){
            if(list.moveCurrentLeft()){
                std::cout << "not found end";
                exit(1);
            }
            if(tokenTypeToString(list.getData().type) != "IDENTIFIER"){
                std::cout << "forgot `variable`" << ", line: " << i;
                exit(1);
            }
            if(list.moveCurrentLeft()){
                std::cout << "not found end";
                exit(1);
            }
            if(list.getData().value != "="){
                std::cout << "forgot `=`" << ", line: " << i;
                exit(1);
            }
            if(list.moveCurrentLeft()){
                std::cout << "not found end";
                exit(1);
            }
            if(tokenTypeToString(list.getData().type) != "IDENTIFIER" &&
tokenTypeToString(list.getData().type) != "NUMBER"){
                std::cout << "forgot `variable`" << ", line: " << i;
                exit(1);
            }
            if(list.moveCurrentLeft()){
                std::cout << "not found end";
                exit(1);
            }
            if(list.getData().value != ";"){
                std::cout << "forgot `;`" << ", line: " << i;
                exit(1);
            }
            i++;
        }
        if(list.getData().value == "begin"){
            countBegin += 1;
            i++;
        }
        int flagIsAssert = 0;
        int flagIsWriteOrReading = 0;
        if(list.getData().value == "write" || list.getData().value ==
"writeln" || list.getData().value == "readln" || list.getData().value ==
"assert"){
            if(list.getData().value == "assert"){
                flagIsAssert = 1;
            }
            else{
                flagIsWriteOrReading = 1;
            }
            if(list.moveCurrentLeft()){
                exit(1);
            }
            if(list.getData().value != "("){
                std::cout << "forgot `(`" << ", line: " << i;
                exit(1);
            }


            while (true){
                if(list.moveCurrentLeft()){
                    std::cout << "not found end";
                    exit(1);
                }
                if(list.getData().value == ";"){
                    if(flagIsWriteOrReading != 3){
                        std::cout << "forgot `variable`" << ", line: " << i;
                        exit(1);
                    }
                    flagIsWriteOrReading = 0;
                    flagIsAssert = 0;
                    i++;
                    break;
                }

                if(tokenTypeToString(list.getData().type) == "IDENTIFIER" ||
tokenTypeToString(list.getData().type) == "STRINGLITERALS" ||
tokenTypeToString(list.getData().type) == "NUMBER"){
                    if(flagIsWriteOrReading == 2){
                        std::cout << "forgot `,`" << ", line: " << i;
                        exit(1);
                    }
                    flagIsWriteOrReading = 2;
                }
                else if(tokenTypeToString(list.getData().type) == "OPERATOR"
|| list.getData().value == ","){
                    if(flagIsWriteOrReading == 1){
                        std::cout << "forgot `Variable`" << ", line: " << i;
                        exit(1);
                    }
                    if(flagIsAssert == 1 && list.getData().value == ","){
                        std::cout << "Cannot use `,`" << ", line: " << i;
                        exit(1);
                    }
                    flagIsWriteOrReading = 1;
                }
                else if(list.getData().value == ")"){
                    if(flagIsWriteOrReading != 2){
                        std::cout << "forgot `Variable`" << ", line: " << i;
                        exit(1);
                    }
                    flagIsWriteOrReading = 3;
                }

            }
        }
        int flagIsWhile = 0;
        int flagIsDo = 0;
        if(list.getData().value == "while"){

            flagIsWhile = 1;
            while (true){
                if(list.moveCurrentLeft()){
                    std::cout << "not found end";
                    exit(1);
                }
                if(list.getData().value == "do"){
                    if(flagIsDo == 1 || flagIsWhile != 2){
                        std::cout << "bad condition" << ", line: " << i;
                        exit(1);
                    }
                    flagIsWhile = 0;
                    flagIsDo = 0;
                    i++;
                    break;
                }
                if(list.getData().value == "("){
                    flagIsDo = 1;
                }
                if(list.getData().value == ")"){
                    if(flagIsDo == 0){
                        std::cout << "forgot `(`" << ", line: " << i;
                        exit(1);
                    }
                    flagIsDo = 0;
                }
                if(tokenTypeToString(list.getData().type) == "IDENTIFIER" ||
tokenTypeToString(list.getData().type) == "NUMBER"){
                    if(flagIsWhile != 1){
                        std::cout << "forgot operator" << ", line: " << i;
                        exit(1);
                    }
                    flagIsWhile = 2;
                }
                if(tokenTypeToString(list.getData().type) == "OPERATOR"){
                    if(flagIsWhile != 2){
                        std::cout << "forgot variable" << ", line: " << i;
                        exit(1);
                    }
                    flagIsWhile = 1;
                }
            }
        }

        int flagIsCorrectIf = 0;
        int flagIsCorrectCondition = 0;
        if(list.getData().value == "if"){
            countIf += 1;
            flagIsCorrectIf = 1;
            while (true){
                if(list.moveCurrentLeft()){
                    std::cout << "not found end";
                    exit(1);
                }
                if(list.getData().value == "then"){
                    if(flagIsCorrectCondition == 1 || flagIsCorrectIf != 2){
                        std::cout << "bad condition" << ", line: " << i;
                        exit(1);
                    }
                    flagIsCorrectIf = 0;
                    flagIsCorrectCondition = 0;
                    i++;
                    break;
                }

                if(list.getData().value == "("){
                    flagIsCorrectCondition = 1;
                }
                if(list.getData().value == ")"){
                    if(flagIsCorrectCondition == 0){
                        std::cout << "forgot `(`" << ", line: " << i;
                        exit(1);
                    }
                    flagIsCorrectCondition = 0;
                }
                if(tokenTypeToString(list.getData().type) == "IDENTIFIER" ||
tokenTypeToString(list.getData().type) == "NUMBER"){
                    if(flagIsCorrectIf != 1){
                        std::cout << "forgot operator" << ", line: " << i;
                        exit(1);
```



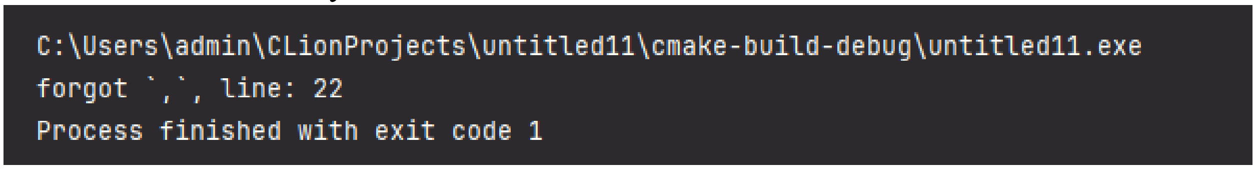
```
                    }
                    flagIsCorrectIf = 2;
                }
                if(tokenTypeToString(list.getData().type) == "OPERATOR"){
                    if(flagIsCorrectIf != 2){
                        std::cout << "forgot variable" << ", line: " << i;
                        exit(1);
                    }
                    flagIsCorrectIf = 1;
                }
            }
        }
        if(tokenTypeToString(list.getData().type) == "IDENTIFIER"){
            if(list.moveCurrentLeft()){
                std::cout << "not found end";
                exit(1);
            }
            if(list.getData().value != ":="){
                std::cout << "forgot `:=`" << ", line: " << i;
                exit(1);
            }
            flagIsCorrectCondition = 1;
            while (true) {
                if (list.moveCurrentLeft()) {
                    std::cout << "not found end";
                    exit(1);
                }
                if(list.getData().value == ";"){
                    if(flagIsCorrectCondition == 1){
                        std::cout << "forgot `variable`" << ", line: " << i;
                        exit(1);
                    }
                    i++;
                    flagIsCorrectCondition = 0;
                    break;
                }

                if(tokenTypeToString(list.getData().type) == "IDENTIFIER"){
                    if(flagIsCorrectCondition != 1){
                        std::cout << "forgot operator" << ", line: " << i;
                        exit(1);
                    }
                    flagIsCorrectCondition = 2;
                }
                if(tokenTypeToString(list.getData().type) == "OPERATOR"){
                    if(flagIsCorrectCondition != 2){
                        std::cout << "forgot variable" << ", line: " << i;
                        exit(1);
                    }
                    flagIsCorrectCondition = 1;
                }
            }
        }
        if(list.getData().value == "else"){
            if(countIf<1){
                std::cout << "not found if from else" << ", line: " << i;
                exit(1);
            }
            countIf -= 1;
            i++;
        }
        if(list.getData().value == "end"){
            if(countBegin < 1){
                std::cout << "not found begin from end" << ", line: " << i;
                exit(1);
            }
            if (list.moveCurrentLeft()) {
                std::cout << "not found end";
                exit(1);
            }
            if(list.getData().value != ";" &&  list.getData().value != "."){
                std::cout << "not found ;" << ", line: " << i;
                exit(1);
            }
            countBegin--;
            i++;
        }
        if(list.moveCurrentLeft()){
            break;
        }
    }
}



int main() {
    SetConsoleOutputCP(CP_UTF8);
    Dlist<Token> list;
    std::string code = R"(
        const eps = 0.0001 ;

var a , b : real ;
begin
  write ( ' Введите числа a и b (a<b) : ' ) ;
  readln ( a , b ) ;
  assert ( a < b ) ;

  var fa := sin ( a ) ;
  var fb := sin ( b ) ;
  assert ( fb * fa < 0 ) ;

  while ( b - a ) > eps do
  begin
    var x := ( b + a ) / 2 ;
    var fx := sin ( x ) ;
    if fa * fx <= 0 then
      b := x ;
    else
    begin
      a := x ;
      fa := fx ;
    end ;
  end ;
  writeln ( ' Корень функции на [a,b] равен ' , ( b + a ) / 2 ) ;
end .
    )";

    lex(code, list);
    parser(list);
    std::cout << "OK" << std::endl;

    return 0;
}
```

Вывод программы:

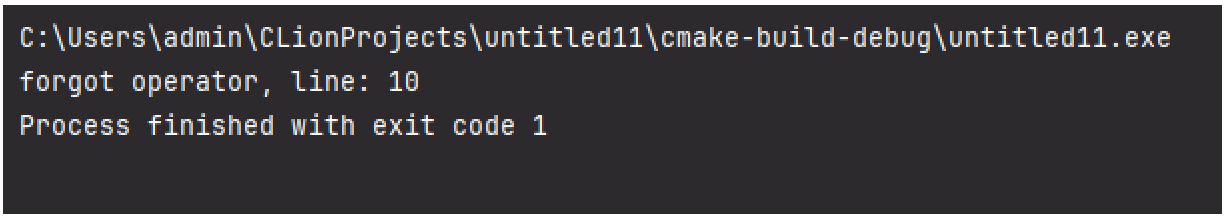


```
C:\Users\admin\CLionProjects\untitled11\cmake-build-debug\untitled11.exe
OK

Process finished with exit code 0
```

Сделаем ошибку в коде:



```
C:\Users\admin\CLionProjects\untitled11\cmake-build-debug\untitled11.exe
forgot `,`, line: 22
Process finished with exit code 1
```



```
C:\Users\admin\CLionProjects\untitled11\cmake-build-debug\untitled11.exe
forgot operator, line: 10
Process finished with exit code 1
```

**Вывод:** в ходе лабораторной работы я научился создавать шаблонные классы.