ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:

“Программирование”

КУРСОВАЯ РАБОТА

Выполнил:
Студент гр. 3147 бакалавриата факультета ФБИТ
Елисеев Леонид

Проверил:
Безруков В.А.

Санкт-Петербург
2023г.

Содержание:

Определения

В настоящем отчёте о НИР применяют следующие термины с соответствующими определениями:

- Си Пи Пи -- Язык программирования C++ за авторством Бьёрна Страуструпа
- ООП -- Объектно Ориентированное Программирование
- ТЗ -- Техническое Задание

Введение

Каждый начинающий разработчик на C++ (си пи пи), который решает попробовать ООП, сталкивается с проблемой при создании эффективной реализации иерархии классов строк. Здесь будет приведено одно из возможных решений. Также в данном решении я постарался точно соблюсти поставленное ТЗ.

Техническое Задание

Описать базовый класс СТРОКА.
Обязательные члены класса:
* указатель на char - хранит адрес динамически выделенной памяти
для размещения символов строки;
* значение типа int - хранит длину строки в байтах.

Обязательные методы должны выполнять следующие действия:

* конструктор без параметров;
* конструктор, принимающий в качестве параметра
Си-строку (заканчивается нулевым байтом);
* конструктор, принимающий в качестве параметра символ (char).
* конструктор копирования;
* деструктор.

* получение длины строки;

Производный от СТРОКА класс СТРОКА_ИДЕНТИФИКАТОР
Строки данного класса строятся по правилам записи идентификаторов
в СИ, и могут включать в себя только те символы, которые могут входить
в состав Си-идентификаторов. Если исходные данные противоречат правилам
записи идентификатора, то создается пустая СТРОКА_ИДЕНТИФИКАТОР.

Обязательные методы:
* конструктор без параметров;
* конструктор, принимающий в качестве параметра Си-строку
(заканчивается нулевым байтом);
* конструктор копирования;
* деструктор.

* перевод всех символов строки (кроме цифр) в верхний регистр;

Переопределить следующие операции:
* присваивание (=);
* оператор < - проверка на меньше. Строка считается меньше другой,
если код символа первой строки в i-й позиции
(i изменяется от 0 до n-1, где n - длина более
короткой строки) меньше кода символа в той же позиции
кода символа в той же позиции во второй строке,
длины строк могут не совпадать.

Производный от СТРОКА класс ДЕСЯТИЧНАЯ_СТРОКА.
Строки данного класса могут содержать только символы десятичных
цифр и символы - и +, задающие знак числа. Символы - или + могут
находиться только в первой позиции числа, причем символ + может
отсутствовать, в этом случае число считается положительным. Если
в составе инициализирующей строки будут встречены любые символы,
отличные от допустимых, ДЕСЯТИЧНАЯ_СТРОКА принимает нулевое значение.

Содержимое данных строк рассматривается как десятичное число.

Обязательные методы:

* конструктор без параметров;
* конструктор, принимающий в качестве параметра
  Си-строку (заканчивается нулевым байтом);
* конструктор копирования;
* деструктор;

* определяющий, можно ли представить данное число в формате char;

Переопределить следующие операции :

* присваивание (=);
* вычитание - - арифметическая разность строк;
* операция < - проверка на меньше (по значению).

Разработчик вправе вводить любое (с обоснованием необходимости)
число дополнительных членов и методов.

Основная часть

Исходный код программы: структура

Моё решение разбито на 7 файлов:
Stroka.h – файл заголовка для класса Stroka
IdentStr.h – файл заголовка для класса IdentStr
DecStr.h – файл заголовка для класса DecStr
Stroka.cpp – реализация методов класса Stroka
IdentStr.cpp – реализация методов класса IdentStr
DecStr.cpp – реализация методов класса DecStr
main.cpp – главный файл

Исходный код программы: содержание файлов

Stroka.h:

```
#pragma once
#ifndef STROKA_H
#define STROKA_H

class Stroka {
public:
    int len;
    char* pCh;
    Stroka(int = 0);
    Stroka(char);
    Stroka(const char*);
    Stroka(const Stroka&);
    ~Stroka();
    char* GetStr(void) const {
        return pCh;
    }
    int GetLen(void) const {
        return len;
    }
    void Show(void);
};

#endif
```

IdentStr.h:

```
#pragma once
#include <iostream>
#include "Stroka.h"

using namespace std;

class IdentStr :public Stroka {
public:
    IdentStr(int = 0);
    IdentStr(char);
    IdentStr(const char*);
    IdentStr(const IdentStr&);
    ~IdentStr();
    void upper(void) const {
        for (int i = 0; i < len; i++) {
            if (pCh[i] >= 'a' && pCh[i] <= 'z') {
                pCh[i] -= 32;
            }
        }
        cout << "the letters are bigger" << endl;
    }
    IdentStr& operator = (const IdentStr&);
    char& operator[] (int);
    IdentStr operator ~();
    friend IdentStr operator+(const IdentStr&, const IdentStr&);
    friend IdentStr operator+(const IdentStr&, const char*);
    friend IdentStr operator+(const char*, const IdentStr&);
    friend bool operator<(const IdentStr&, const IdentStr&);
};
```

DecStr.h:

```h
#pragma once
#include <iostream>
#include "Stroka.h"

using namespace std;

class DecStr :public Stroka {
public:
    DecStr(int = 0);
    //DecStr(char);
    DecStr(const char*);
    DecStr(const DecStr&);
    ~DecStr();
    DecStr& operator = (const DecStr&);
    friend DecStr operator + (const DecStr&, const DecStr&);
/*дружественная функция имеет доступ ко всем частям класса*/
    friend DecStr operator + (const DecStr&, const int);
    friend DecStr operator + (const int, const DecStr&);
    friend DecStr operator - (const DecStr&, const DecStr&);
    friend DecStr operator - (const DecStr&, const int);
    friend DecStr operator - (const int, const DecStr&);
    friend bool operator<(const DecStr&, const DecStr&);
};
```

Stroka.cpp:

```h
#include "Stroka.h"
#include <iostream>
```

```
using namespace std;

Stroka::Stroka(int val) : len(val), pCh(new char[len + 1])
{
    if (val == 0) pCh[0] = '\0';
    //cout << "Stroka::Stroka(int val) : len(val), pCh(new char[len + 1])"
<< endl;
}

Stroka::Stroka(char ch) : len(1), pCh(new char[len + 1])
{
    pCh[0] = ch;
    pCh[1] = '\0';
    //cout << "Stroka::Stroka(char ch) : len(1), pCh(new char[len + 1])"
<< endl;
}

Stroka::~Stroka()
{
    if (pCh) delete[]pCh;
    //cout << "Stroka::~Stroka()" << endl;
}

void Stroka::Show(void)
{
    cout << "pCh = " << pCh << endl;
    cout << "len = " << len << endl;
}

Stroka::Stroka(const char* S) : len(strlen(S)), pCh(new char[len + 1])
{
    strcpy_s(pCh, len + 1, S);
    //cout << "Stroka::Stroka(const char* S) : len(strlen(S)), pCh(new
char[len + 1])" << endl;
}

Stroka::Stroka(const Stroka& from) : len(strlen(from.pCh)), pCh(new
char[from.len + 1])
{
    strcpy_s(pCh, len + 1, from.pCh);
    //cout << "Stroka::Stroka(const Stroka& from) : len(strlen(from.pCh)),
pCh(new char[from.len + 1])" << endl;
}
```

IdentStr.cpp:

```
#include "IdentStr.h"
#include <iostream>
```

```
using namespace std;


IdentStr::IdentStr(int val) :Stroka(val)
{
    //cout << "IdentStr::IdentStr(int val) :Stroka(val) " << val << endl;
}

IdentStr::IdentStr(char ch) :Stroka(ch)
{
    if (!((pCh[0] >= 'a' && pCh[0] <= 'z') || (pCh[0] >= 'A' && pCh[0] <=
'Z') || (pCh[0] == '_'))) {
        cout << "Bad simvol, pCh[0] = " << pCh[0] << endl;
        if (pCh) delete[]pCh;
        len = 0;
        pCh = new char[len + 1];
        pCh[0] = '\0';
        return;
    }
    //cout << pCh << endl;
}

IdentStr::IdentStr(const char* Str) :Stroka(Str)
{
    const char* keyword[] = { "alignas", "alignof", "andB", "and_eqB",
"asma", "auto", "bitandB", "bitorB", "bool", "break", "case",
        "catch", "char", "char8_tc", "char16_t", "char32_t", "class",
"complB", "conceptc", "const", "const_cast", "constevalc", "constexpr",
        "constinitc", "continue", "co_awaitc", "co_returnc", "co_yieldc",
"decltype", "default", "delete", "do", "double", "dynamic_cast",
        "else", "enum", "explicit", "exportc", "extern", "false", "float",
"for", "friend", "goto", "if", "inline", "", "int", "long", "mutable",
        "namespace", "new", "noexcept", "notB", "not_eqB", "nullptr",
"operator", "orB", "or_eqB", "private", "protected", "public",
        "register reinterpret_cast", "requiresc", "return", "short",
"signed", "sizeof", "static", "static_assert", "static_cast", "struct",
        "switch", "template", "this", "thread_local", "throw", "true",
"try", "typedef", "typeid", "typename", "union", "unsigned", "virtual",
        "void", "volatile", "wchar_t", "while", "xorB", "xor_eq", }; //90
ключевых слов
    for (int i = 0; i < 89; i++) {
        if (strcmp(pCh, keyword[i]) == 0) {
            cout << "Error, delete '" << keyword[i] << "'" << endl;
            len = 0;
            pCh = new char[len + 1];
            pCh[0] = '\0';
            return;
        }
    }
    if (!((pCh[0] >= 'a' && pCh[0] <= 'z') || (pCh[0] >= 'A' && pCh[0] <=
'Z') || (pCh[0] == '_'))) {
```

```
        cout << "Bad simvol, pCh[0] = " << pCh[0] << endl;
        if (pCh) delete[]pCh;
        len = 0;
        pCh = new char[len + 1];
        pCh[0] = '\0';
        return;
    }
    for (int i = 1; i < len; i++) {
        if (!((pCh[i] >= 'a' && pCh[i] <= 'z') || (pCh[i] >= 'A' && pCh[i]
<= 'Z') || (pCh[i] == '_') || (pCh[i] >= '0' && pCh[i] <= '9'))) {
            cout << "Bad simvol, pCh[" << i << "] = " << pCh[i] << endl;
            if (pCh) delete[]pCh;
            len = 0;
            pCh = new char[len + 1];
            pCh[0] = '\0';
            return;
        }
    }
    //cout << "IdentStr::IdentStr(const char* Str) :Stroka(Str)" << endl;
}

IdentStr::IdentStr(const IdentStr& from) :Stroka(from)
{
    //cout << "IdentStr::IdentStr(const IdentStr& from) :Stroka(from)" <<
endl;
}

IdentStr::~IdentStr()
{
    //cout << "IdentStr::~IdentStr()" << endl;
}

IdentStr& IdentStr::operator=(const IdentStr& S)
{
    if (&S != this) {
        delete[]pCh;
        len = strlen(S.pCh);
        pCh = new char[len + 1];
        strcpy_s(pCh, len + 1, S.pCh);
    }
    //cout << "IdentStr& IdentStr::operator=(const IdentStr& S)" << endl;
    return *this;
}

IdentStr operator+(const IdentStr& one, const IdentStr& other) {
    int newSize = one.len + other.len;
    char* newData = new char[newSize + 1];

    strcpy_s(newData, newSize + 1, one.pCh);
    strcat_s(newData, newSize + 1, other.pCh);
```

```
    IdentStr result(newData);
    delete[] newData;
    return result;
}

IdentStr operator+(const IdentStr& one, const char* other) {
    int newSize = one.len + strlen(other);
    char* newData = new char[newSize + 1];

    strcpy_s(newData, newSize + 1, one.pCh);
    strcat_s(newData, newSize + 1, other);

    IdentStr result(newData);
    delete[] newData;
    return result;
}

IdentStr operator+(const char* other, const IdentStr& one) {
    int newSize = one.len + strlen(other);
    char* newData = new char[newSize + 1];

    strcpy_s(newData, newSize + 1, other);
    strcat_s(newData, newSize + 1, one.pCh);

    IdentStr result(newData);
    delete[] newData;
    return result;
}

bool operator<(const IdentStr& one, const IdentStr& other) {
    int minLength = min(one.len, other.len);

    for (int i = 0; i < minLength; ++i) {
        if (one.pCh[i] < other.pCh[i]) {
            return true;
        }
        else if (one.pCh[i] > other.pCh[i]) {
            return false;
        }
    }

    return one.len < other.len;
}

char& IdentStr::operator[](int index)
{
    if (index >= 0 && index < len) {
        //cout << "char& IdentStr::operator[](int index)" << endl;
        return pCh[index];
    }
    return pCh[0];
```

```
}

IdentStr IdentStr::operator ~()
{
    int i = 0, j = len - 1;
    char tmp;
    if (!(pCh[j] >= '0' && pCh[j] <= '9')) {
        for (i = 0, j = len - 1; i < len / 2; i++, j--)
        {
            tmp = pCh[i];
            pCh[i] = pCh[j];
            pCh[j] = tmp;
        }
    }
    else {
        cout << "reverse not possible, last simvol is bad" << endl;
    }
    //cout << "IdentStr IdentStr::operator ~()" << endl;
    return *this;
}
```

DecStr.cpp:

```
#include "DecStr.h"
#include <iostream>
#include <cstring>
#include <string>

using namespace std;

DecStr::DecStr(int val) :Stroka(val)
```

```
{
    //cout << "DecStr::DecStr(int val) :Stroka(val), val = " << val <<
endl;
}

DecStr::DecStr(const char* Str) :Stroka(Str)
{
    if ((pCh[0] == '+' || pCh[0] == '-') && len == 1) { pCh[0] = '0'; }
    if (!((pCh[0] >= '1' && pCh[0] <= '9') || (pCh[0] == '+') || (pCh[0]
== '-') || (len == 1 && pCh[0] == '0'))) {
        cout << "Bad simvol, pCh[0] = " << pCh[0] << endl;
        if (pCh) delete[]pCh;
        len = 0;
        pCh = new char[len + 1];
        pCh[0] = '\0';
        return;
    }
    if (len > 1) {
        if (((pCh[0] == '+') || (pCh[0] == '-')) && (!(pCh[1] >= '1' &&
pCh[1] <= '9'))) {
            cout << "Bad simvol, pCh[1] = " << pCh[1] << endl;
            if (pCh) delete[]pCh;
            len = 0;
            pCh = new char[len + 1];
            pCh[0] = '\0';
            return;
        }
    }
    for (int i = 1; i < len; i++) {
        if (!(pCh[i] >= '0' && pCh[i] <= '9')) {
            cout << "Bad simvol, pCh[" << i << "] = " << pCh[i] << endl;
            if (pCh) delete[]pCh;
            len = 0;
            pCh = new char[len + 1];
            pCh[0] = '\0';
            return;
        }
    }
    //cout << "DecStr::DecStr(const char* Str) :Stroka(Str)" << endl;
}

DecStr::DecStr(const DecStr& from) :Stroka(from)
{
    //cout << "DecStr::DecStr(const DecStr& from) :Stroka(from)" << endl;
}

DecStr::~DecStr()
{
    //cout << "DecStr::~DecStr()" << endl;
}
```

```
bool operator<(const DecStr& one, const DecStr& other) {
    if (one.pCh[0] == '-' && other.pCh[0] != '-') { return true; }
    else if (one.pCh[0] != '-' && other.pCh[0] == '-') { return false; }
    else {
        int i = 0, j = 0;
        if (one.pCh[0] == '-' || one.pCh[0] == '+') { i++; }
        if (other.pCh[0] == '-' || other.pCh[0] == '+') { j++; }
        if (one.len - i != other.len - j) {
            if (one.pCh[0] == '-' && other.pCh[0] == '-') {
                if (one.len - i > other.len - j) { return true; }
                else { return false; }
            }
            else {
                if (one.len - i < other.len - j) { return true; }
                else { return false; }
            }
        }
        else {
            for (i; i < one.len; i++, j++) {
                if (one.pCh[i] < other.pCh[j]) { return true; }
                if (one.pCh[i] > other.pCh[j]) { return false; }
            }
            return false;
        }
    }
}

DecStr& DecStr::operator = (const DecStr& Ds)
{
    if (&Ds != this) {
        delete[]pCh;
        len = strlen(Ds.pCh);
        pCh = new char[len + 1];
        strcpy_s(pCh, len + 1, Ds.pCh);
    }
    //cout << "DecStr& DecStr::operator = (const DecStr& Ds)" << endl;
    return *this;
}

DecStr operator + (const DecStr& pobg1, const DecStr& pobg2)
{
    int num1, num2;
    num1 = atoi(pobg1.GetStr()); //переводит строку в число
    num2 = atoi(pobg2.GetStr());
    int A = num1 + num2;
    cout << num1 << " + " << num2 << " = " << A << endl;
    string b = to_string(A);
    const char* c = b.c_str();
    int newSize = strlen(c);
    char* newData = new char[newSize + 1];
    strcpy_s(newData, newSize + 1, c);
```

```
    DecStr result(newData);
    delete[] newData;
    return result;
}

DecStr operator + (const DecStr& pobg1, const int pobg2)
{
    int num1;
    num1 = atoi(pobg1.GetStr()); //переводит строку в число
    int A = num1 + pobg2;
    cout << num1 << " + " << pobg2 << " = " << A << endl;
    string b = to_string(A);
    const char* c = b.c_str();
    int newSize = strlen(c);
    char* newData = new char[newSize + 1];
    strcpy_s(newData, newSize + 1, c);
    DecStr result(newData);
    delete[] newData;
    return result;
}

DecStr operator + (const int pobg2, const DecStr& pobg1)
{
    int num1;
    num1 = atoi(pobg1.GetStr()); //переводит строку в число
    int A = num1 + pobg2;
    cout << pobg2 << " + " << num1 << " = " << A << endl;
    string b = to_string(A);
    const char* c = b.c_str();
    int newSize = strlen(c);
    char* newData = new char[newSize + 1];
    strcpy_s(newData, newSize + 1, c);
    DecStr result(newData);
    delete[] newData;
    return result;
}

DecStr operator - (const DecStr& pobg1, const DecStr& pobg2)
{
    int num1, num2;
    num1 = atoi(pobg1.GetStr()); //переводит строку в число
    num2 = atoi(pobg2.GetStr());
    int A = num1 - num2;

    cout << num1 << " - " << num2 << " = " << A << endl;
    string b = to_string(A);
    const char* c = b.c_str();
    int newSize = strlen(c);
    char* newData = new char[newSize + 1];
    strcpy_s(newData, newSize + 1, c);
    DecStr result(newData);
```

```
    delete[] newData;
    return result;
}

DecStr operator - (const DecStr& pobg1, const int pobg2)
{
    int num1;
    num1 = atoi(pobg1.GetStr()); //переводит строку в число
    int A = num1 - pobg2;

    cout << num1 << " - " << pobg2 << " = " << A << endl;
    string b = to_string(A);
    const char* c = b.c_str();
    int newSize = strlen(c);
    char* newData = new char[newSize + 1];
    strcpy_s(newData, newSize + 1, c);
    DecStr result(newData);
    delete[] newData;
    return result;
}

DecStr operator - (const int pobg2, const DecStr& pobg1)
{
    int num1;
    num1 = atoi(pobg1.GetStr()); //переводит строку в число
    int A = pobg2 - num1;

    cout << pobg2 << " - " << num1 << " = " << A << endl;
    string b = to_string(A);
    const char* c = b.c_str();
    int newSize = strlen(c);
    char* newData = new char[newSize + 1];
    strcpy_s(newData, newSize + 1, c);
    DecStr result(newData);
    delete[] newData;
    return result;
}
```

main.cpp:

```
#include "DecStr.h"
#include "IdentStr.h"
#include "Stroka.h"
#include <iostream>

using namespace std;

void Stroka_test() {
    Stroka strobj1;
    Stroka strobj2("Hello");
    Stroka strobj3("itmo");
```

```
    Stroka strobj4('p');
    Stroka strobj5(strobj2);
    Stroka strobj6("56");
    cout << endl;

    strobj1.Show();
    strobj2.Show();
    strobj3.Show();
    strobj4.Show();
    strobj5.Show();
    strobj6.Show();
    cout << endl;
    cout << "strobj3 len = " << strobj3.GetLen() << endl;
    cout << "strobj3 str = " << strobj3.GetStr() << endl;
}

void IdentStr_test() {
    cout << endl << "-------------------------------------------------
-----";
    cout << endl;
    IdentStr isobj1;
    IdentStr isobj2('p');
    IdentStr isobj3("itmo");
    IdentStr isobj4("fbit");
    IdentStr isobj5("654");
    cout << endl;

    cout << "isobj1 = " << isobj1.GetStr() << endl;
    cout << "isobj2 = " << isobj2.GetStr() << endl;
    cout << "isobj3 = " << isobj3.GetStr() << endl;
    cout << "isobj4 = " << isobj4.GetStr() << endl;
    cout << "isobj5 = " << isobj5.GetStr() << endl; //bad simvol
    cout << endl;

    IdentStr isobj6;
    isobj6 = isobj3;
    if (isobj6 < isobj4) { cout << "isobj6 < isobj4" << endl; }
    else { cout << "isobj6 !< isobj4" << endl; }
    cout << endl;

    cout << "isobj3 = " << isobj3.GetStr() << endl;
    isobj3.upper();
    cout << "isobj3 = " << isobj3.GetStr() << endl;
    cout << endl;

    IdentStr isobj7("_forrevers");
    cout << "isobj7 = " << isobj7.GetStr() << endl;
    IdentStr isobj8("_forrevers8");
    cout << "isobj8 = " << isobj8.GetStr() << endl;
    ~isobj7;
    cout << "isobj7 = " << isobj7.GetStr() << endl;
```

```
    ~isobj8;
    cout << "isobj8 = " << isobj8.GetStr() << endl;
    cout << endl;

    IdentStr sumstr;
    sumstr = isobj3 + isobj4;
    cout << "sumstr = " << sumstr.GetStr() << endl;
    sumstr = sumstr + "isbetter";
    cout << "sumstr = " << sumstr.GetStr() << endl;

}

void DecStr_test() {
    cout << endl << "-------------------------------------------------------
-----" << endl;
    cout << endl;
    DecStr dcobj;
    DecStr dcobj1("456");
    DecStr dcobj2("+123");
    DecStr dcobj3("-85");
    DecStr dcobj4("qwerty");
    cout << endl;

    cout << "dcobj = " << dcobj.GetStr() << endl;
    cout << "dcobj1 = " << dcobj1.GetStr() << endl;
    cout << "dcobj2 = " << dcobj2.GetStr() << endl;
    cout << "dcobj3 = " << dcobj3.GetStr() << endl;
    cout << "dcobj4 = " << dcobj4.GetStr() << endl; //bad simvol
    cout << endl;

    DecStr dcobj5;
    dcobj5 = dcobj3;
    cout << endl;

    if (dcobj5 < dcobj2) { cout << "dcobj5 < dcobj2" << endl; }
    else { cout << "dcobj5 !< dcobj2" << endl; }
    if (dcobj1 < dcobj2) { cout << "dcobj1 < dcobj2" << endl; }
    else { cout << "dcobj1 !< dcobj2" << endl; }
    cout << endl;

    dcobj2 + dcobj3;
    dcobj5 - 32;
    74 + dcobj1;
    dcobj2 - dcobj3;

}

int main() {
    cout << endl;
    Stroka_test();
    IdentStr_test();
```

```
    DecStr_test();
    cout << endl;

     return 0;
}
```

Выводы:

У меня получилось разработать эффективную иерархию классов строк на языке C++. В этом мне помогли мои знания, полученные на практиках и лекциях по программированию, также я пользовался дополнительными источниками в виде книг по программированию. Делая этот проект, я освоил и закрепил такой навык как ООП. Это безусловно пригодится мне в дальнейшем. Результат моей работы меня полностью устраивает.

Список использованных источников

[1] Richard L. (2019) *Fundamentals of Programming C++*.
[2] Березин, Б. И., & Березин, С. Б. (1998). *Начальный курс C и C++*. Диалог МИФИ Москва.
[3] Stroustrup, B. (2013). *The C++ Programming Language*. Pearson Education.
[4] Роберт Лафоре. (2016). *Объектно-ориентированное программирование в C++ 4-е издание.*
[5] Мейерс Скотт. (2019) *Эффективный и современный C++: 42 рекомендации по использованию C++11 и C++14.*
[6] Васильев А. Н. (2016) *Программирование на C++ в примерах и задачах.*