**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

(СПбГУТ)

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ СЕТЕЙ И СИСТЕМ **(ИКСС)**

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ **(ПИ И ВТ)**

Дисциплина: «ООП»

Лабораторная работа №1.

**Тема: «Классы»**

Вариант №16

Выполнил: Студент группы ИКПИ-95

Новиков С.C.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Приняла:

Петрова О.Б.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

2020г

А. *Постановка задачи*

Цель настоящей работы состоит в ознакомлении студента с правилами организации классов, принятыми при программировании на языке С++. В процессе выполнения настоящей работы каждый студент должен разработать два класса и написать тестовые программы для демонстрации их работоспособности.

*Б. Задача 1 (5)*

Необходимо предусмотреть выполнение следующих операций над строками разрабатываемого класса:

- int Length() – определение длины строки,

- void Copy(const String& str) – скопировать строку str,

- void Copy(const char\* str) – скопировать строку str,

- String Substr(int index, int count) – выделение подстроки, начиная с индекса index; count – длина, выделяемой подстроки,

- void Remove(int index, int count) – удаление подстроки, начиная с индекса index; count – длина удаляемой подстроки.

- void Insert(char\* s, int index) - вставка строки в стиле языка C в строку, для которой вызывается метод Insert; index – позиция, перед которой выполняется вставка.

- void trim() – удалить из строки ведущие и завершающие пробелы,

- void read() – ввести строку с клавиатуры,

- void print() – метод для вывода строки на экран дисплея.

Требования к реализации. Строка должна располагаться в динамической памяти. Необходимо предусмотреть поле для хранения информации о длине строки.

*В. Задача 2(15)*

В этих задачах требуется разработать класс, обеспечивающий работу с комплексными числами. Задачи отличаются своей реализацией. В задаче 15 реализация должна содержать два поля, определяющие соответственно действительную и мнимую часть комплексного числа. В задаче 16 реализация должна содержать три поля. Первое и втрое поле должны задавать само комплексное число (его действительную и мнимую части), а последнее поле должно содержать модуль комплексного числа.

Разработанный класс(Complex) должен обеспечить выполнение следующих операций:

- сложение,

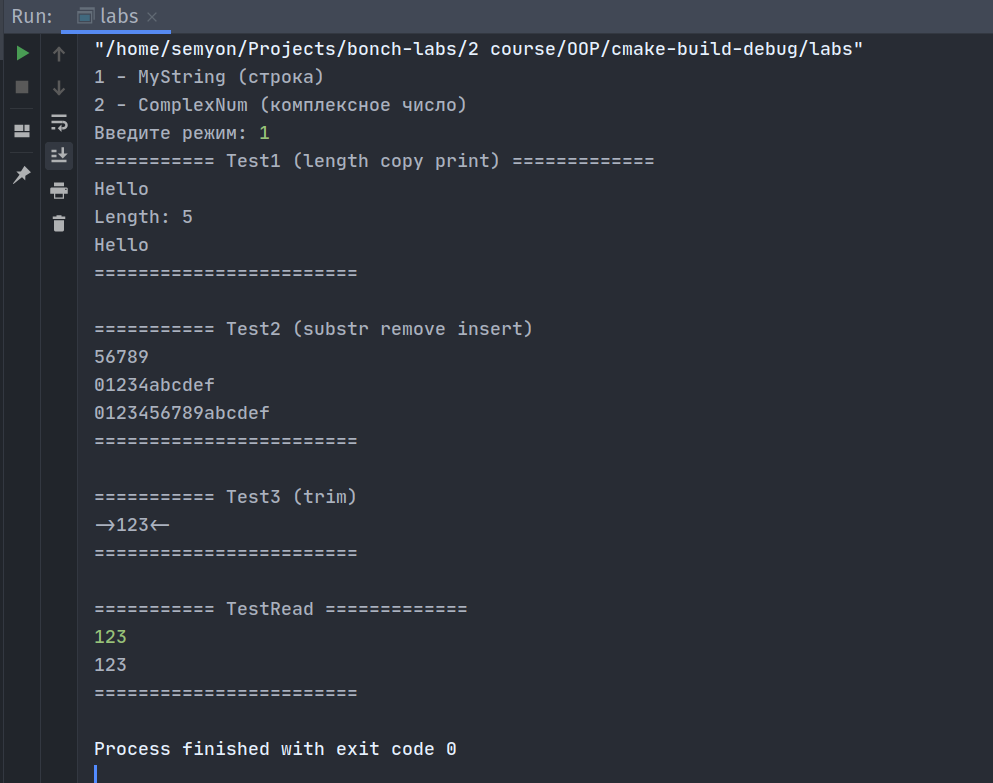
- вычитание,

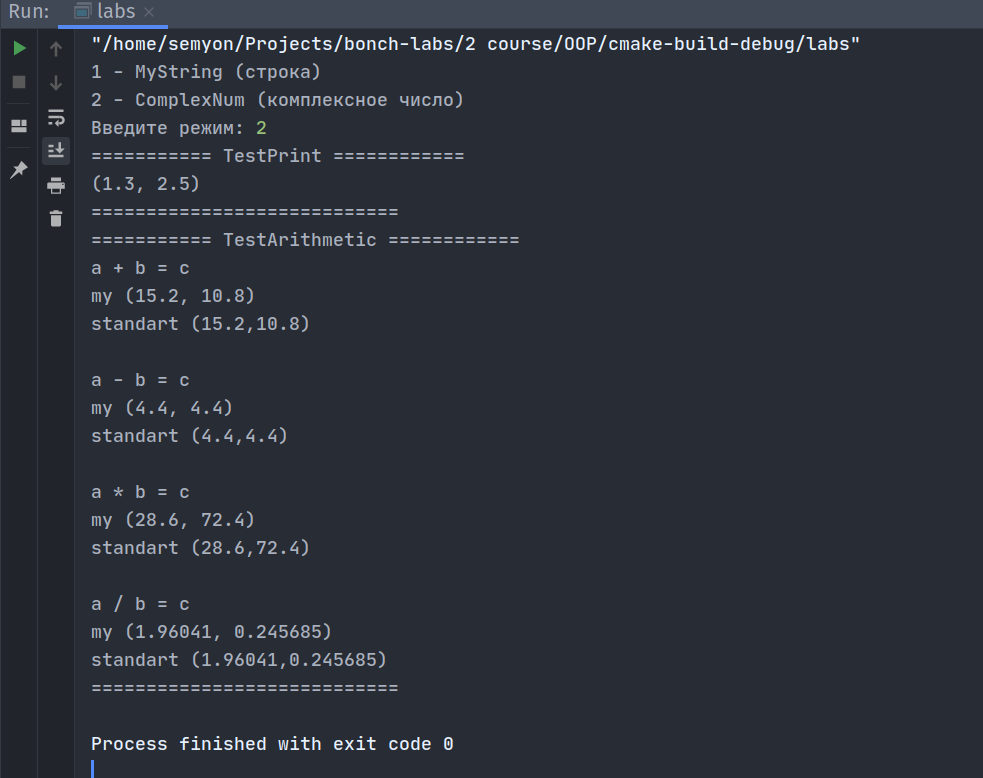
- умножение,

- деление,

- вывод комплексного числа на экран дисплея.

Г. *Результаты работы программы*





E. *Программа на языке* ***c++***

Файл main.cpp

#include <iostream>

#include <complex> // для проверки

#include "String/MyString.hpp"

#include "ComplexNumber/ComplexNumber.hpp"

using namespace std;

namespace TString {

// length copy print

void Test1() {

cout << "=========== Test1 (length copy print) =============" << endl;

MyString str;

str.copy("Hello");

str.print();

cout << endl;

cout << "Length: " << str.length() << endl;

MyString str2;

str2.copy(str);

cout << str2 << endl;

cout << "========================" << endl;

}

// substr remove insert

void Test2() {

cout << "=========== Test2 (substr remove insert)" << endl;

MyString str;

str.copy("0123456789abcdef");

cout << str.subStr(5, 5) << endl;

str.remove(5, 5);

cout << str << endl;

str.insert("56789", 5);

cout << str << endl;

cout << "========================" << endl;

}

// trim

void Test3() {

cout << "=========== Test3 (trim)" << endl;

MyString str;

str.copy(" 123 ");

str.trim();

cout << "->" << str << "<-" << endl;

cout << "========================" << endl;

}

//read

void TestRead() {

cout << "=========== TestRead =============" << endl;

MyString str;

str.read();

str.print();

cout << endl;

cout << "========================" << endl;

}

}

namespace TComplex {

void TestPrint() {

cout << "=========== TestPrint ============" << endl;

ComplexNumD c(1.3, 2.5);

cout << c << endl;

cout << "============================" << endl;

}

void TestArithmetic() {

cout << "=========== TestArithmetic ============" << endl;

ComplexNumD a1(9.8, 7.6), b1(5.4, 3.2), c1;

complex<double> a2(9.8, 7.6), b2(5.4, 3.2), c2;

c1 = a1 + b1;

c2 = a2 + b2;

cout << "a + b = c" << endl;

cout << "my " << c1 << endl;

cout << "standart " << c2 << endl;

cout << endl;

c1 = a1 - b1;

c2 = a2 - b2;

cout << "a - b = c" << endl;

cout << "my " << c1 << endl;

cout << "standart " << c2 << endl;

cout << endl;

c1 = a1 \* b1;

c2 = a2 \* b2;

cout << "a \* b = c" << endl;

cout << "my " << c1 << endl;

cout << "standart " << c2 << endl;

cout << endl;

c1 = a1 / b1;

c2 = a2 / b2;

cout << "a / b = c" << endl;

cout << "my " << c1 << endl;

cout << "standart " << c2 << endl;

cout << "============================" << endl;

}

}

void TestString() {

using namespace TString;

Test1();

cout << endl;

Test2();

cout << endl;

Test3();

cout << endl;

TestRead();

}

void TestComplex() {

using namespace TComplex;

TestPrint();

TestArithmetic();

}

int main() {

setbuf(stdout, nullptr);

cout << "1 - MyString (строка)" << endl;

cout << "2 - ComplexNum (комплексное число)" << endl;

cout << "Введите режим: ";

int mode;

cin >> mode;

if (mode == 1)

TestString();

if (mode == 2)

TestComplex();

return 0;

}

Файл MyString.hpp

#pragma once

class MyString {

public:

MyString();

MyString(const char \*str);

MyString(MyString &str);

MyString(MyString &&str) noexcept;

~MyString();

void reserve(size\_t size);

const char \*c\_str() const;

char operator[](size\_t i) const;

const MyString &operator=(const MyString &str) = delete;

size\_t length() const;

void copy(const MyString &str);

void copy(const char \*str);

MyString subStr(size\_t index, size\_t count) const;

void remove(size\_t index, size\_t count);

void insert(const char \*str, size\_t index);

void read();

void print();

void trim();

private:

char \*str;

size\_t size;

size\_t capacity;

};

std::ostream &operator<<(std::ostream &out, const MyString &str);

Файл MyString.cpp

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

#include "MyString.hpp"

size\_t CStrLength(const char \*str) {

size\_t len = 0;

while (true) {

if (str[len] == '\0')

return ++len;

++len;

}

}

void clearStdin() {

int c;

while ((c = fgetc(stdin)) != EOF && c != '\n');

}

std::ostream &operator<<(std::ostream &out, const MyString &str) {

out << str.c\_str();

return out;

}

MyString::MyString() {

str = new char[5];

capacity = 5;

size = 0;

str[0] = '\0';

}

MyString::MyString(const char \*str) {

size\_t len = CStrLength(str);

this->str = new char[len];

strcat(this->str, str);

size = len - 1;

capacity = len;

}

MyString::MyString(MyString &str) {

this->str = new char[str.size + 1];

strcat(this->str, str.str);

size = str.size;

capacity = str.size + 1;

}

MyString::MyString(MyString &&str) noexcept {

this->str = str.str;

str.str = nullptr;

this->size = str.size;

this->capacity = str.capacity;

}

MyString::~MyString() {

delete[] str;

}

void MyString::reserve(size\_t size) {

if (capacity >= size)

return;

capacity = size;

char \*sub\_str = new char[size];

for (size\_t i = 0; i < this->size + 1; i++) {

sub\_str[i] = str[i];

}

delete[] str;

str = sub\_str;

}

const char \*MyString::c\_str() const {

return str;

}

char MyString::operator[](size\_t i) const {

return str[i];

}

size\_t MyString::length() const {

return size;

}

void MyString::copy(const MyString &str) {

size\_t len = str.size + 1;

if (capacity < len) {

delete[] this->str;

this->str = new char[len];

capacity = len;

}

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

this->str[i] = str[i];

size = len - 1;

}

void MyString::copy(const char \*str) {

size\_t len = CStrLength(str);

if (capacity < len) {

delete[] this->str;

this->str = new char[len];

capacity = len;

}

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

this->str[i] = str[i];

}

size = len - 1;

}

MyString MyString::subStr(size\_t index, size\_t count) const {

MyString string;

string.reserve(count + 1);

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

string.str[i] = str[index + i];

}

string.str[count] = '\0';

string.capacity = count + 1;

string.size = count;

return string;

}

void MyString::remove(size\_t index, size\_t count) {

for (size\_t i = index; i + count < size + 1; i++) {

str[i] = str[i + count];

}

size -= count;

}

void MyString::insert(const char \*str, size\_t index) {

size\_t len = CStrLength(str) - 1;

reserve(size + len + 1);

for (size\_t i = size; i >= index; i--) {

this->str[i + len] = this->str[i];

}

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

this->str[i + index] = str[i];

}

size = size + len + 1;

}

void MyString::trim() {

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

if (str[i] != ' ') {

remove(0, i);

break;

}

}

for (size\_t i = size - 1; i != -1; i--) {

if (str[i] != ' ')

break;

size--;

str[i] = '\0';

}

}

void MyString::read() {

constexpr size\_t buf\_size = 10;

char \*text = reinterpret\_cast<char \*>(calloc(1, 1)), buffer[buf\_size];

size\_t text\_size;

bool final = false;

clearStdin();

while (fgets(buffer, buf\_size, stdin)) {

text\_size = strlen(text) + 1 + strlen(buffer);

void \*r\_m = realloc(text, text\_size);

if (r\_m == nullptr) {

std::cerr << "Error realoc memory\n";

exit(-1);

}

text = reinterpret\_cast<char \*>(r\_m);

if (!text)

throw "Error: fgets";

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

if (buffer[i] == '\n') {

buffer[i] = '\0';

final = true;

break;

}

strcat(text, buffer);

if (final)

break;

}

copy(text);

}

void MyString::print() {

printf("%s", str);

}

Файл ComplexNumber.hpp

#pragma once

#include <iostream>

template<typename RealNum, typename ImaginaryNum>

class ComplexNum {

public:

ComplexNum();

ComplexNum(RealNum real, ImaginaryNum imaginary);

ComplexNum(const ComplexNum<RealNum, ImaginaryNum> &r);

ComplexNum(const ComplexNum<RealNum, ImaginaryNum> &&r) noexcept;

~ComplexNum();

RealNum getReal() const;

ImaginaryNum getImaginary() const;

// сложение комплексных чисел

ComplexNum operator+(const ComplexNum &r) const;

// вычитание комплексных чисел

ComplexNum operator-(const ComplexNum &r) const;

// умножение комплексных чисел

ComplexNum operator\*(const ComplexNum &r) const;

// деление комплексных чисел

ComplexNum operator/(const ComplexNum &r) const;

// оператор присваивания

ComplexNum &operator=(const ComplexNum &r);

// операторы сравнения

bool operator==(const ComplexNum<RealNum, ImaginaryNum> &r) const;

bool operator!=(const ComplexNum<RealNum, ImaginaryNum> &r) const;

RealNum real;

ImaginaryNum imaginary;

};

template<typename RealNum, typename ImaginaryNum>

std::ostream &operator<<(std::ostream &out, const ComplexNum<RealNum, ImaginaryNum> &r);

typedef ComplexNum<float, float> ComplexNumF;

typedef ComplexNum<double, double> ComplexNumD;

#include "ComplexNumber.inl"

Ж. *Выводы*

В результате проведенной работы изучены классы в c++.