|  |  |
| --- | --- |
|  | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования"Московский технологический университет"МИРЭА | |
| Факультет информационных технологий (ИТ) | |
| Кафедра практической и прикладной информатики | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ** | |
| **по дисциплине** | |
| **«**Объектно-ориентированное программирование**»** | |
|  | |
| Выполнил студент группы ИНБО-04-18 | Лаптев И.А. |
| Принял ассистент кафедры | Хлебникова В.Л. |

|  |  |  |
| --- | --- | --- |
| Лабораторные работы выполнены | «23» февраля 2019 г. |  |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. |  |

Москва 2019

**Содержание**

[Таблица успеваемости 3](#_Toc2269390)

[Практическая работа №1 4](#_Toc2269391)

[Практическая работа №2 6](#_Toc2269392)

[Практическая работа №3 11](#_Toc2269393)

## Таблица успеваемости

**Лабораторные занятия**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №/р. | страница в отчете. | Студент выполнил:  (подпись) | Преподаватель принял:  (подпись) | Балл  (максимально возможный) | Балл (Фактический) |
| 1. |  |  |  | 2 |  |
| 2. |  |  |  | 2 |  |
| 3. |  |  |  | 4 |  |
| 4. |  |  |  | 6 |  |
| 5. |  |  |  | 6 |  |
| 6. |  |  |  | 8 |  |
| 7. |  |  |  | 8 |  |
| 8. |  |  |  | 8 |  |
|  |  |  | Итоговый балл: | 44 |  |

**Практические занятия**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №/р. | страница в отчете. | Студент выполнил:  (подпись) | Преподаватель принял:  (подпись) | Балл  (максимально возможный) | Балл (Фактический) |
| 1. |  |  |  | 2 | … |
| 2. |  |  |  | 2 |  |
| 3. |  |  |  | 4 |  |
| 4. |  |  |  | 6 |  |
| 5. |  |  |  | 6 |  |
| 6. |  |  |  | 8 |  |
| 7. |  |  |  | 8 |  |
|  |  |  | Итоговый балл: | 36 |  |

## Практическая работа №1

**Изучение синтаксиса языка C++**

**Цель практической работы**

Целью данной практической работы является знакомство с синтаксисом, основными алгоритмическими конструкциями языка С++, а также приобретение практических навыков разработки объектно-ориентированных программ на языке программирования С++.

**Задачи**

1. Реализовать программу, которая считывает имя и здоровается
2. Реализовать программу, производящую операции над числами. (Сложение/вычитание, целые числа/числа с плавающей запятой)

**Ход работы**

1. Реализовать программу, которая считывает имя и здоровается

**#include** <iostream>

**#include** <string>

**using** **namespace** std;

**int** **main**() {

**setlocale**(LC\_ALL, "");

string name;

cout << "Enter your name: ";

cin >> name;

cout << "Hello, " << name << "!" << **endl**;

**system**("pause");

**return** 0;

}

1. Реализовать программу, производящую операции над числами. (Сложение/вычитание, целые числа/числа с плавающей запятой)

**#include** <iostream>

**using** **namespace** std;

**int** **main**() {

**setlocale**(LC\_ALL, "");

**double** a, b;

cout << "Enter two numbers: ";

cin >> a >> b;

cout << "a - b = " << (a - b) << **endl**;

cout << "a + b = " << (a + b) << **endl**;

**system**("pause");

**return** 0;

}

**Вывод**

В ходе выполнения практической работы была использована библиотека стандартного ввода и вывода, операторы ввода, вывода, сложения и вычитания. Были достигнуты результаты, изображённые на рисунках (рис. 1 и 2). Также исходный код программ был помещён на GitHub и доступен по ссылкам:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr1.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr1.2>

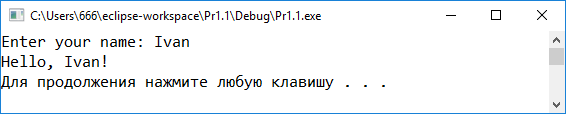


Рис. 1 Результат работы первой программы

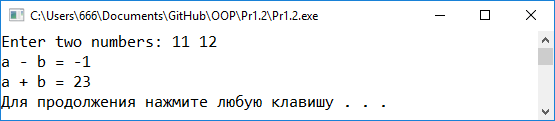


Рис. 2 Результат работы второй программы

## 

## Практическая работа №2

**Решение задач на нелинейные алгоритмические конструкции языка С++**

**Цель практической работы**

Целью данной практической работы является получение практических навыков по разработке нелинейных алгоритмов с использованием конструкции «if-else».

**Ход работы**

1. Вариант 1

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**int** **main**() {

**double** a, b, m, n;

cout << "Enter a and b: ";

cin >> a >> b;

**if** (a >= b) {

n = **cbrt**(a - b);

} **else** {

n = a \* a + (a - b) / (**sin**(a \* b));

}

**if** (n < b) {

m = (n + a) / (-b) + **sqrt**(**sin**(a) \* **sin**(a) - **cos**(n));

} **else** {

**if** (n == b) {

m = b \* b + **tan**(n \* a);

} **else** {

m = b \* b \* b + n \* a \* a;

}

}

**if** (isnan(m) || isnan(n) || isinf(m) || isinf(n)) {

cout << "Error: cannot count result" << **endl**;

} **else** {

cout << "a = " << a << **endl**;

cout << "b = " << b << **endl**;

cout << "m = " << m << **endl**;

cout << "n = " << n << **endl**;

}

**system**("pause");

**return** 0;

}

1. Вариант 2

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**int** **main**() {

**double** a, b, z, t;

cout << "Enter a and b: ";

cin >> a >> b;

**if** (a < b) {

z = **sqrt**(abs(a \* a - b \* b));

} **else** {

z = 1 - 2 \* **cos**(a) \* **sin**(b);

}

**if** (z < b) {

t = **cbrt**(z + a \* a \* b);

} **else** {

**if** (z == b) {

t = 1 - **log10**(z) + **cos**(a \* a \*b);

} **else** {

t = 1 / **cos**(z \* a);

}

}

**if** (isnan(t) || isnan(z) || isinf(z) || isinf(t)) {

cout << "Error: cannot count result" << **endl**;

} **else** {

cout << "a = " << a << **endl**;

cout << "b = " << b << **endl**;

cout << "t = " << t << **endl**;

cout << "z = " << z << **endl**;

}

**system**("pause");

**return** 0;

}

1. Вариант 3

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**int** **main**() {

**double** a, b, y, t;

cout << "Enter a and b: ";

cin >> a >> b;

**if** (a <= b) {

y = (a - b) \* (a + b) / ((a + b) \* (a \* a - a \* b + b \* b));

} **else** {

y = a + pow(**log**(b), 2);

}

**if** (y == b) {

t = (2 \* y + **sqrt**(y \* y - a)) / (2 \* b - **sqrt**(a \* a - y));

} **else** {

**if** (y < b) {

t = **sin**(y) \* **sin**(y) + 1 / **tan**(a - b);

} **else** {

t = **cbrt**(y \* **sin**(a)) + 1 / **sqrt**(y \* **cos**(b));

}

}

**if** (isnan(y) || isnan(t) || isinf(y) || isinf(t)) {

cout << "Error: cannot count result" << **endl**;

} **else** {

cout << "a = " << a << **endl**;

cout << "b = " << b << **endl**;

cout << "y = " << y << **endl**;

cout << "t = " << t << **endl**;

}

**system**("pause");

**return** 0;

}

1. Вариант 4

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**int** **main**() {

**double** a, x, y, t;

cout << "Enter a and x: ";

cin >> a >> x;

**if** (a <= x) {

y = a + **log**(x + a);

} **else** {

y = **sqrt**(**sin**(a \* x));

}

**if** (a > y) {

t = y / (a - x);

} **else** {

**if** (a == y) {

t = y / (a - x) + (a + x) / (y \* y);

} **else** {

t = **tan**(a \* x) + **cos**(2 \* a \* y);

}

}

**if** (isnan(y) || isnan(t) || isinf(y) || isinf(t)) {

cout << "Error: cannot count result" << **endl**;

} **else** {

cout << "a = " << a << **endl**;

cout << "x = " << x << **endl**;

cout << "y = " << y << **endl**;

cout << "t = " << t << **endl**;

}

**system**("pause");

**return** 0;

}

**Вывод**

В ходе выполнения практической работы была использована библиотека cmath, которая позволила вычислять сложные математические функции, а также быстро совершить проверку на ошибки, и конструкция ветвления «if-else». Были достигнуты результаты, изображённые на рисунках (рис. 3-6). Также код программ доступен по ссылкам:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr2.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr2.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Pr2.3>
4. <https://github.com/IvLaptev/OOP/tree/master/Pr2.4>

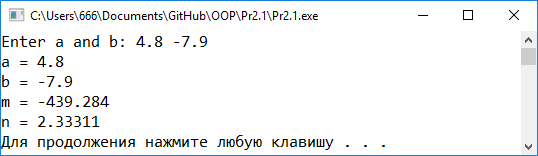


Рис. 3 Результат практической работы №2 вариант 1

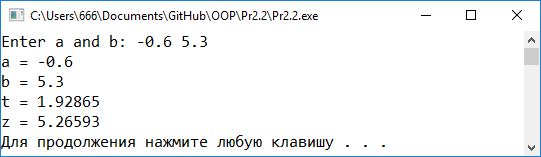


Рис. 4 Результат практической работы №2 вариант 2

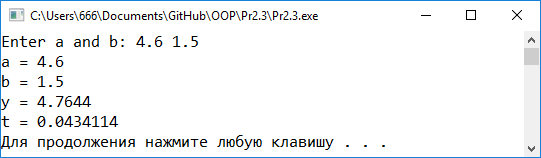


Рис. 5 Результат практической работы №2 вариант 3

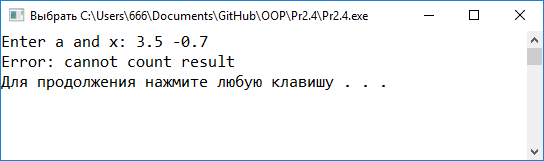


Рис. 6 Результат практической работы №2 вариант 4

## Практическая работа №3

**Классы в языке С++**

**Цель практической работы**

Целью данной практической работы является приобретение практических навыков при разработке классов на языке программирования С++.

**Задачи**

1. Реализовать класс «Книга»
2. Реализовать класс «Группа студентов»
3. Реализовать класс «Автомобиль»
4. Реализовать класс «Вектор»
5. Реализовать класс «Библиотека»

**Ход работы**

1. Реализовать класс «Книга»

* Файл Book.h

**#include <iostream>**

**#include <string>**

**#include <vector>**

**#ifndef BOOK\_H\_**

**#define BOOK\_H\_**

**class Book {**

**int pages; // количество страниц**

**int currentPage; //текущая страница**

**std::vector<std::string> authors; // авторы**

**std::string title; // название**

**std::string genre; // жанр**

**std::string publisher; // издатель**

**public:**

**Book(); // Создаёт пустой экземпляр**

**Book(int, const std::vector<std::string> &, std::string &, // Создаёт экземпляр с**

**const std::string &, const std::string &); // заполненными полями**

**void setNumberOfPages(const int); // Ввод количества страниц в книге**

**int getNumberOfPages(); // Получение количества страниц**

**void setAuthors(const std::vector<std::string> &); // Ввод авторов книги**

**void addAuthor(const std::string &); // Добавление одного автора к уже существующим**

**std::vector<std::string> &getAuthors(); // Получение авторов книги**

**void setPublisher(const std::string &); // Ввод издателя книги**

**std::string &getPublisher(); // Получение издателя книги**

**void setGenre(const std::string &); // Ввод жанра**

**std::string &getGenre(); // Получение жанра**

**void openPage(const int); // Установка определённой текущей страницы**

**void nextPage(); // Открытие следующей страницы**

**int getCurrentPage(); // Получение текущей открытой страницы**

**void setTitle(const std::string &); // Ввод названия книги**

**std::string &getTitle(); // Получение названия книги**

**~Book(); // Деструктор экземпляра**

**};**

**#endif /\* BOOK\_H\_ \*/**

* Файл Book.cpp

**#include** "Book.h"

**Book::Book**() {

**this**->pages = 0;

**this**->currentPage = 0;

**this**->authors.clear();

**this**->genre = "";

**this**->publisher = "";

}

**Book::Book**(**int** pages, **const** std::vector<std::string> &authors, std::string &title,

**const** std::string &genre, **const** std::string &publisher) {

**this**->pages = pages;

**if** (**this**->pages) {

**this**->currentPage = 1;

} **else** {

**this**->currentPage = 1;

}

**this**->authors = authors;

**this**->title = title;

**this**->genre = genre;

**this**->publisher = publisher;

}

**void** **Book::setAuthors**(**const** std::vector<std::string> &authors) {

**this**->authors = authors;

}

**void** **Book::addAuthor**(**const** std::string &author) {

authors.push\_back(author);

}

std::vector<std::string> &**Book::getAuthors**() {

**if**(!authors.size()) {

std::cout << "The book has no author." << std::**endl**;

}

**return** authors;

}

**void** **Book::setNumberOfPages**(**const** **int** pages) {

**this**->pages = pages;

**if** (pages) {

currentPage = 1;

} **else** {

currentPage = 0;

}

}

**int** **Book::getNumberOfPages**() { **return** pages; }

**void** **Book::setPublisher**(**const** std::string & publisher) {

**this**->publisher = publisher;

}

std::string &**Book::getPublisher**() {

**if** (!publisher.size()) {

std::cout << "The book has no publisher." << std::**endl**;

}

**return** publisher;

}

**void** **Book::setGenre**(**const** std::string &genre) { **this**->genre = genre; }

std::string &**Book::getGenre**() {

**if** (!genre.size()) {

std::cout << "The book has no genre." << std::**endl**;

}

**return** genre;

}

**void** **Book::openPage**(**const** **int** page) {

**if** (page <= pages) {

currentPage = page;

} **else** {

std::cout << "Page " << page << " does not exist." << std::**endl**;

}

}

**void** **Book::nextPage**() {

**if** (currentPage == pages) {

std::cout << "Can't open next page. This page is the last." << std::**endl**;

} **else** {

currentPage++;

}

}

**int** **Book::getCurrentPage**() { **return** currentPage; }

**void** **Book::setTitle**(**const** std::string &title) {

**this**->title = title;

}

std::string &**Book::getTitle**() {

**if** (!title.size()) {

std::cout << "The book has no title." << std::**endl**;

}

**return** title;

}

**Book::~Book**() {

authors.clear();

genre.clear();

publisher.clear();

}

* Файл main.cpp

**#include** <iostream>

**#include** <vector>

**#include** <string>

**#include** <Windows.h>

**#include** "Book.h"

**using** **namespace** std;

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

Book book1;

cout << "Title: " << book1.getTitle() << **endl**;

cout << "Authors: ";

**for** (**auto** a : book1.getAuthors()) {

cout << a << " ";

}

cout << **endl**;

cout << "Number of pages: " << book1.getNumberOfPages() << **endl**;

cout << "Current page: " << book1.getCurrentPage() << **endl**;

cout << "Genre: " << book1.getGenre() << **endl**;

cout << "Publisher: " << book1.getPublisher() << **endl**;

cout << "----------------------------------------" << **endl**;

string s;

cout << "Enter title: ";

cin >> s;

book1.setTitle(s);

cout << "Enter first author: ";

cin >> s;

book1.setAuthors({s});

cout << "Enter second author: ";

cin >> s;

book1.addAuthor(s);

**int** f;

cout << "Enter number of pages: ";

cin >> f;

book1.setNumberOfPages(f);

cout << "Enter current page: ";

cin >> f;

book1.openPage(f);

book1.nextPage();

cout << "Enter publisher: ";

cin >> s;

book1.setPublisher(s);

cout << "Enter genre: ";

cin >> s;

book1.setGenre(s);

cout << "Title: " << book1.getTitle() << **endl**;

cout << "Authors: ";

**for** (**auto** a : book1.getAuthors()) {

cout << a << " ";

}

cout << **endl**;

cout << "Number of pages: " << book1.getNumberOfPages() << **endl**;

cout << "Current page: " << book1.getCurrentPage() << **endl**;

cout << "Genre: " << book1.getGenre() << **endl**;

cout << "Publisher: " << book1.getPublisher() << **endl**;

cout << "----------------------------------------" << **endl**;

s = "sss";

Book book2(250, {s}, s, s, s);

book2.openPage(250);

book2.nextPage();

cout << "Title: " << book2.getTitle() << **endl**;

cout << "Authors: ";

**for** (**auto** a : book2.getAuthors()) {

cout << a << " ";

}

cout << **endl**;

cout << "Number of pages: " << book2.getNumberOfPages() << **endl**;

cout << "Current page: " << book2.getCurrentPage() << **endl**;

cout << "Genre: " << book2.getGenre() << **endl**;

cout << "Publisher: " << book2.getPublisher() << **endl**;

cout << "----------------------------------------" << **endl**;

**system**("pause");

**return** 0;

}

1. Реализовать класс «Группа студентов»

* Файл Student.h

// Класс Student создан для создания списка студентов в классе Group

**#include** <iostream>

**#include** <string>

**#ifndef** STUDENT\_H\_

**#define** STUDENT\_H\_

**class** Student {

**int** age; // Возраст

std::string name; // Имя

std::string surname; // Фамилия

std::string phoneNumber; // Номер телефона

std::string email; // Адрес электронной почты

**public**:

**Student**(); // Конструктор, заполняющий объект нулевыми значениями

**Student**(**const** std::string &, **const** std::string &, **int**, // Конструктор, заполняющий

**const** std::string &, **const** std::string &); // все поля

**void** **setName**(**const** std::string &); // Ввод имени

**void** **setSurname**(**const** std::string &); // Ввод фамилии

**void** **setAge**(**int**); // Ввод возраста

**void** **setPhoneNumber**(**const** std::string &); // Ввод номера телефона

**void** **setEmail**(**const** std::string &); // Ввод электронной почты

std::string &**getName**(); // Получение имени

std::string &**getSurname**(); // Получение фамилии

**int** **getAge**(); // Получение возраста

std::string &**getPhoneNumber**(); // Получение номера телефона

std::string &**getEmail**(); // Получение адреса электронной почты

**void** **printInformation**(); // Вывод информаации о студенте на экран

**~Student**(); // Деструктор

};

**#endif** /\* STUDENT\_H\_ \*/

* Файл Student.cpp

**#include** "Student.h"

**Student::Student**() {

age = 0;

name = "";

surname = "";

phoneNumber = "";

email = "";

}

**Student::Student**(**const** std::string &name, **const** std::string &surname, **int** age,

**const** std::string &phoneNumber, **const** std::string &email) {

**this**->age = age;

**if** (age <= 0) {

**this**->age = 0;

std::cout << "Warning: age was entered incorrectly" << std::**endl**;

}

**this**->email = email;

**this**->name = name;

**this**->phoneNumber = phoneNumber;

**this**->surname = surname;

}

**void** **Student::printInformation**() {

std::cout << "Name: ";

**if** (name == "") {

std::cout << "student has no name" << std::**endl**;

} **else** {

std::cout << name << std::**endl**;

}

std::cout << "Surname: ";

**if** (surname == "") {

std::cout << "student has no surname" << std::**endl**;

} **else** {

std::cout << surname << std::**endl**;

}

std::cout << "Age: ";

**if** (age == 0) {

std::cout << "age is unknown" << std::**endl**;

} **else** {

std::cout << age << std::**endl**;

}

std::cout << "Phone number: ";

**if** (phoneNumber == "") {

std::cout << "student has no phone number" << std::**endl**;

} **else** {

std::cout << phoneNumber << std::**endl**;

}

std::cout << "Email: ";

**if** (email == "") {

std::cout << "student has no email" << std::**endl**;

} **else** {

std::cout << email << std::**endl**;

}

}

**void** **Student::setName**(**const** std::string &name) { **this**->name = name; }

**void** **Student::setSurname**(**const** std::string &surname) {

**this**->surname = surname;

}

**void** **Student::setAge**(**int** age) {

**this**->age = age;

**if** (age <= 0) {

**this**->age = 0;

std::cout << "Warning: age was entered incorrectly" << std::**endl**;

}

}

**void** **Student::setPhoneNumber**(**const** std::string &phoneNumber) {

**this**->phoneNumber = phoneNumber;

}

**void** **Student::setEmail**(**const** std::string &email) { **this**->email = email; }

std::string &**Student::getName**() { **return** name; }

std::string &**Student::getSurname**() { **return** surname; }

**int** **Student::getAge**() { **return** age; }

std::string &**Student::getPhoneNumber**() { **return** phoneNumber; }

std::string &**Student::getEmail**() { **return** email; }

**Student::~Student**() {}

* Файл Group.h

**#include** "Student.h"

**#include** <vector>

**#include** <string>

**#ifndef** GROUP\_H\_

**#define** GROUP\_H\_

// Ключи для изменения данных студентов

**#define** S\_NAME 1

**#define** S\_SURNAME 2

**#define** S\_AGE 3

**#define** S\_PHONE 4

**#define** S\_EMAIL 5

**class** Group {

std::vector<Student> students; // Список студентов

std::string name; // Название группы

**public**:

**Group**(); // Конструктор, заполняющий объект нулевыми значениями

// Конструктор, заполняющий все поля

**Group**(**const** std::string &, **const** std::vector<Student> &);

**void** **setName**(**const** std::string &); // Ввод названия группы

std::string &**getName**(); //Вывод названия группы

**void** **addStudent**(**const** Student &student); // Добавление студента к списку

Student &**getStudent**(**int**); // Получение студента по его номеру

**void** **printStudent**(**int**); // Печать информации о студенте по его номеру

// Изменение текстовой информации о студенте по его номеру и ключу

**void** **setStudentParam**(**int**, **int**, **const** std::string &);

// Изменение целочисленных данных о студенте по его номеру и ключу

**void** **setStudentParam**(**int**, **int**, **int**);

// Получение определённых данных о студенте по его номеру и ключу

**template**<**typename** **T**> **T** **getStudentParam**(**int**, **int**);

**void** **printIdList**(); // Вывод на экран списка студентов с их номерами

**void** **eraseStudent**(**int**); // Удаление студента из списка

**void** **clear**(); // Удаление всех студентов из списка группы

**~Group**(); // Деструктор

};

**#endif** /\* GROUP\_H\_ \*/

* Файл Group.cpp

**#include** "Group.h"

**Group::Group**() {

name = "";

students.clear();

}

**Group::Group**(**const** std::string &name, **const** std::vector<Student> &students) {

**this**->name = name;

**for** (**auto** a : students) {

**this**->students.push\_back(a);

}

}

**void** **Group::setName**(**const** std::string &name) { **this**->name = name; }

std::string &**Group::getName**() {

**if** (name == "") {

std::cout << "Warning: name of group is empty" << std::**endl**;

}

**return** name;

}

**void** **Group::addStudent**(**const** Student &student) {

students.push\_back(student);

}

Student &**Group::getStudent**(**int** id) {

**if** (id < students.size() && id >= 0) {

**return** students[id];

} **else** {

std::cout << "Warning: there is no student with such id" << std::**endl**;

Student s;

**return** s;

}

}

**void** **Group::printStudent**(**int** id) {

**if** (id < students.size() && id >= 0) {

students[id].printInformation();

} **else** {

std::cout << "Warning: there is no student with such id" << std::**endl**;

}

}

**void** **Group::setStudentParam**(**int** id, **int** name, **const** std::string &value) {

**if** (id < students.size() && id >= 0) {

**switch**(name) {

**case** S\_NAME:

students[id].setName(value);

**break**;

**case** S\_SURNAME:

students[id].setSurname(value);

**break**;

**case** S\_PHONE:

students[id].setPhoneNumber(value);

**break**;

**case** S\_EMAIL:

students[id].setEmail(value);

**break**;

**default**:

std::cout << "Warning: setStudentParam name wrong" << std::**endl**;

**break**;

}

} **else** {

std::cout << "Warning: there is no student with such id" << std::**endl**;

}

}

**void** **Group::setStudentParam**(**int** id, **int** name, **int** value) {

**if** (id < students.size() && id >= 0) {

**if** (name == S\_AGE) {

students[id].setAge(value);

} **else** {

std::cout << "Warning: wrong pair of name and value" << std::**endl**;

}

} **else** {

std::cout << "Warning: there is no student with such id" << std::**endl**;

}

}

**template**<> std::string &**Group::getStudentParam**(**int** id, **int** name) {

**if** (id < students.size() && id >= 0) {

**switch**(name) {

**case** S\_NAME:

**return** students[id].getName();

**case** S\_SURNAME:

**return** students[id].getSurname();

**case** S\_PHONE:

**return** students[id].getPhoneNumber();

**case** S\_EMAIL:

**return** students[id].getEmail();

**default**:

std::cout << "Warning: getStudentParam name wrong" << std::**endl**;

std::string s = "";

**return** s;

}

} **else** {

std::cout << "Warning: there is no student with such id" << std::**endl**;

}

}

**template**<> **int** **Group::getStudentParam**(**int** id, **int** name) {

**if** (id < students.size() && id >= 0) {

**if** (name == S\_AGE) {

**return** students[id].getAge();

} **else** {

std::cout << "Warning: wrong pair of name and return type" << std::**endl**;

}

} **else** {

std::cout << "Warning: there is no student with such id" << std::**endl**;

}

}

**void** **Group::printIdList**() {

std::cout << "List of " << name << " group:" << std::**endl**;

**if** (students.size() > 0) {

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

std::cout << i << "\t" << students[i].getSurname() << " " <<

students[i].getName() << std::**endl**;

}

} **else** {

std::cout << "List is empty" << std::**endl**;

}

}

**void** **Group::eraseStudent**(**int** id) {

**if** (id < students.size() && id >= 0) {

students.erase(students.begin() + id);

} **else** {

std::cout << "Warning: there is no student with such id" << std::**endl**;

}

}

**void** **Group::clear**() { students.clear(); }

**Group::~Group**() {}

* Файл main.cpp

**#include** <iostream>

**#include** <vector>

**#include** "Student.h"

**#include** "Group.h"

**#include** <Windows.h>

**using** **namespace** std;

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string name, surname, tel, mail;

**int** age;

cout << "Enter name: ";

cin >> name;

cout << "Enter surname: ";

cin >> surname;

cout << "Enter age: ";

cin >> age;

cout << "Enter phone number: ";

cin >> tel;

cout << "Enter email: ";

cin >> mail;

Student s1(name, surname, age, tel, mail);

Student s2("Stepan", "Kot", 40, "+79184736254", "kots@yambler.cot");

Student s3;

vector<Student> s = {s1, s2, s3};

Group inbo4("INBO-04-18", s);

inbo4.printStudent(0);

cout << "-----------------------------------" << **endl**;

inbo4.printStudent(2);

cout << "-----------------------------------" << **endl**;

inbo4.printStudent(3);

cout << "-----------------------------------" << **endl**;

inbo4.setStudentParam(2, S\_NAME, "Ivan");

inbo4.setStudentParam(2, S\_SURNAME, "Laptev");

inbo4.setStudentParam(2, S\_AGE, 18);

inbo4.setStudentParam(2, S\_PHONE, "0");

inbo4.setStudentParam(2, S\_EMAIL, "ivlaptev");

inbo4.printStudent(2);

cout << "-----------------------------------" << **endl**;

cout << inbo4.getStudentParam<std::string&>(2, S\_NAME) << **endl**;

cout << inbo4.getStudentParam<std::string&>(2, S\_SURNAME) << **endl**;

cout << inbo4.getStudentParam<**int**>(2, S\_AGE) << **endl**;

cout << inbo4.getStudentParam<std::string&>(2, S\_PHONE) << **endl**;

cout << inbo4.getStudentParam<std::string&>(2, S\_EMAIL) << **endl**;

cout << "-----------------------------------" << **endl**;

inbo4.addStudent(s2);

inbo4.printIdList();

cout << "-----------------------------------" << **endl**;

inbo4.getStudent(3).printInformation();

cout << "-----------------------------------" << **endl**;

inbo4.eraseStudent(3);

inbo4.printIdList();

cout << "-----------------------------------" << **endl**;

inbo4.clear();

inbo4.printIdList();

**system**("pause");

**return** 0;

}

1. Реализовать класс «Автомобиль»

* Файл Car.h

**#include** <iostream>

**#include** <string>

**#ifndef** CAR\_H\_

**#define** CAR\_H\_

**class** Car {

std::string model; // Модель автомобиля

**int** seats; // Количество мест в автомобиле

**double** currentSpeed; // Текущая скорость автомобиля

**double** acceleration; // Ускорение автомобиля

**double** maxSpeed; // Максимальная скорость автомобиля

**bool** isStarted; // true - автомобиль заведён, false - нет

**public**:

**Car**(); // Создаёт объект, заполненный нулевыми значениями

**Car**(**const** std::string &, **int**, **double**, **double**); // Создаёт незапущенный автомобиль

// Создаёт автомобиль, принимая значения всех полей

**Car**(**const** std::string &, **int**, **double**, **double**, **double**, **bool**);

**void** **setModel**(**const** std::string &); // Устанавливает модель автомобиля

**void** **setNumberOfSeats**(**int**); // Устанавливает количество мест в автомобиле

**void** **setCurrentSpeed**(**double**); // Устанавливает текущую скорость

**void** **setAcceleration**(**double**); // Устанавливает ускорение

**void** **setMaxSpeed**(**double**); //Устанавливает максимальную скорость

**void** **startCar**(); // "заводит" двигатель автомобиля

**void** **blockCar**(); // "глушит" двигатель автомобиля

std::string &**getModel**(); // Возвращает модель автомобиля

**int** **getNumberOfSeats**(); // Возвращает количество мест в автомобиле

**double** **getMaxSpeed**(); // Возвращает максимальную скорость

**double** **getCurrentSpeed**(); // Возвращает текущую скорость

**double** **getAcceleration**(); // Возвращает ускорение

**bool** **isWorking**(); // Возвращает состояние автомобиля (isStarted)

// Увеличивает скорость на основе времени, за которое шло ускорение

**void** **increaseSpeed**(**double**);

// Уменьшает скорость на основе времени, за которое шло ускорение

**void** **decreaseSpeed**(**double**);

**~Car**(); // Деструктор

};

**#endif** /\* CAR\_H\_ \*/

* Файл Car.cpp

**#include** "Car.h"

**Car::Car**() {

model = "";

seats = 0;

currentSpeed = 0;

acceleration = 0;

maxSpeed = 0;

isStarted = **false**;

}

**Car::Car**(**const** std::string &model, **int** seats, **double** acceleration, **double** maxSpeed) {

setModel(model);

setNumberOfSeats(seats);

setAcceleration(acceleration);

setMaxSpeed(maxSpeed);

blockCar();

}

**Car::Car**(**const** std::string &model, **int** seats, **double** maxSpeed, **double** currentSpeed,

**double** acceleration, **bool** isStarted) {

**if** (isStarted) {

startCar();

} **else** {

blockCar();

}

setModel(model);

setNumberOfSeats(seats);

setMaxSpeed(maxSpeed);

setCurrentSpeed(currentSpeed);

setAcceleration(acceleration);

}

**void** **Car::setModel**(**const** std::string &model) { **this**->model = model; }

**void** **Car::setNumberOfSeats**(**int** seats) {

**if** (seats > 0) {

**this**->seats = seats;

} **else** {

std::cout << "Warning: seats has incorrect value" << std::**endl**;

seats = 0;

}

}

**void** **Car::setCurrentSpeed**(**double** currentSpeed) {

**if** (isStarted) {

**if** (currentSpeed > 0) {

**this**->currentSpeed = currentSpeed;

} **else** {

std::cout << "Warning: currentSpeed has incorrect value" << std::**endl**;

currentSpeed = 0;

}

} **else** {

std::cout << "Warning: currentSpeed hasn't been changed" <<std::**endl** <<

"\t(car is not started)" << std::**endl**;

}

}

**void** **Car::setAcceleration**(**double** acceleration) {

**if** (acceleration > 0) {

**this**->acceleration = acceleration;

} **else** {

std::cout << "Warning: acceleration has incorrect value" << std::**endl**;

acceleration = 0;

}

}

**void** **Car::setMaxSpeed**(**double** maxSpeed) {

**if** (maxSpeed < currentSpeed) {

std::cout << "Warning: maxSpeed hasn't been changed " << std::**endl** <<

"\t(maxSpeed cannot be lower than currentSpeed)" << std::**endl**;

} **else** {

**if** (maxSpeed > 0) {

**this**->maxSpeed = maxSpeed;

} **else** {

std::cout << "Warning: maxSpeed has incorrect value" << std::**endl**;

maxSpeed = 0;

}

}

}

**void** **Car::startCar**() { isStarted = **true**; }

**void** **Car::blockCar**() {

isStarted = **false**;

currentSpeed = 0;

}

std::string &**Car::getModel**() { **return** model; }

**int** **Car::getNumberOfSeats**() { **return** seats; }

**double** **Car::getMaxSpeed**() { **return** maxSpeed; }

**double** **Car::getCurrentSpeed**() { **return** currentSpeed; }

**double** **Car::getAcceleration**() { **return** acceleration; }

**bool** **Car::isWorking**() { **return** isStarted; }

**void** **Car::increaseSpeed**(**double** deltaTime) {

currentSpeed += deltaTime \* acceleration;

**if** (currentSpeed > maxSpeed) {

currentSpeed = maxSpeed;

}

}

**void** **Car::decreaseSpeed**(**double** deltaTime) {

currentSpeed -= deltaTime \* acceleration;

**if** (currentSpeed < 0) {

currentSpeed = 0;

}

}

**Car::~Car**() {}

* Файл main.cpp

**#include** <iostream>

**#include** <Windows.h>

**#include** "Car.h"

**using** **namespace** std;

**void** **printInformation**(Car &);

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string m;

**int** s;

**double** a, v;

cout << "Enter model: ";

cin >> m;

cout << "Enter number of seats: ";

cin >> s;

cout << "Enter acceleration: ";

cin >> a;

cout << "Enter max speed: ";

cin >> v;

Car c1(m, s, a, v);

printInformation(c1);

cout << "-------------------------------" << **endl**;

c1.setCurrentSpeed(20);

c1.startCar();

c1.setCurrentSpeed(20);

printInformation(c1);

cout << "-------------------------------" << **endl**;

c1.increaseSpeed(20);

c1.setMaxSpeed(20);

c1.setAcceleration(-30);

printInformation(c1);

cout << "-------------------------------" << **endl**;

c1.decreaseSpeed(30);

c1.setMaxSpeed(10);

printInformation(c1);

cout << "-------------------------------" << **endl**;

Car c2("Kali", 1, 200, 10, 20, **true**);

printInformation(c2);

cout << "-------------------------------" << **endl**;

c2.blockCar();

printInformation(c2);

**system**("pause");

**return** 0;

}

**void** **printInformation**(Car &c) {

cout << "Model: " << c.getModel() << **endl**;

cout << "Number of seats: " << c.getNumberOfSeats() << **endl**;

cout << "Working: " << c.isWorking() << **endl**;

cout << "Max speed: " << c.getMaxSpeed() << **endl**;

cout << "Current speed: " << c.getCurrentSpeed() << **endl**;

cout << "Acceleration: " << c.getAcceleration() << **endl**;

}

1. Реализовать класс «Вектор»

* Файл Vector.h

**#include** <cmath>

**#ifndef** VECTOR\_H\_

**#define** VECTOR\_H\_

**class** Vector {

**double** x; // Координаты вектора

**double** y;

**double** z;

**public**:

**Vector**(); // Конструктор, создающий нулевой вектор

// Конструктор, создающий вектор по точкам начали и конца

**Vector**(**double**, **double**, **double**, **double**, **double**, **double**);

// Конструктор, создающий вектор по его координатам

**Vector**(**double**, **double**, **double**);

**void** **set**(**double**, **double**, **double**); // Задаёт новые координаты вектора

**void** **setX**(**double** x); // Задаёт координату x

**void** **setY**(**double** y); // Задаёт координату y

**void** **setZ**(**double** z); // Задаёт координату z

**double** **getX**(); // Возвращает координату x

**double** **getY**(); // Возвращает координату y

**double** **getZ**(); // Возвращает координату z

**double** **getLength**(); // Возвращает длину вектора

**void** **add**(Vector&); // Прибавляет вектор

**double** **algProd**(Vector&); // Возвращает алгебраическое произведение

**void** **geomProd**(Vector&); // Производит геометрическое умножение на вектор

**void** **prod**(**double**); // Производит умножение вектора на число

**~Vector**(); // Деструктор

};

**#endif** /\* VECTOR\_H\_ \*/

* Файл Vector.cpp

**#include** "Vector.h"

**Vector::Vector**() {

x = 0;

y = 0;

z = 0;

}

**Vector::Vector**(**double** xb, **double** yb, **double** zb, **double** xe, **double** ye, **double** ze) {

x = xe - xb;

y = ye - yb;

z = ze - zb;

}

**Vector::Vector**(**double** x, **double** y, **double** z) {

**this**->x = x;

**this**->y = y;

**this**->z = z;

}

**void** **Vector::set**(**double** x, **double** y, **double** z) {

**this**->x = x;

**this**->y = y;

**this**->z = z;

}

**void** **Vector::setX**(**double** x) { **this**->x = x; }

**void** **Vector::setY**(**double** y) { **this**->y = y; }

**void** **Vector::setZ**(**double** z) { **this**->z = z; }

**double** **Vector::getX**() { **return** x; }

**double** **Vector::getY**() { **return** y; }

**double** **Vector::getZ**() { **return** z; }

**double** **Vector::getLength**() { **return** **sqrt**(x \* x + y \* y + z \* z); }

**void** **Vector::add**(Vector &vector) {

x += vector.getX();

y += vector.getY();

z += vector.getZ();

}

**double** **Vector::algProd**(Vector &vector) {

**return** x \* vector.getX() + y \* vector.getY() + z \* vector.getZ();

}

**void** **Vector::geomProd**(Vector &vector) {

**double** x1 = x;

**double** y1 = y;

**double** z1 = z;

x = y1 \* vector.getZ() - vector.getY() \* z1;

y = z1 \* vector.getX() - vector.getZ() \* x1;

z = x1 \* vector.getY() - vector.getX() \* y1;

}

**void** **Vector::prod**(**double** a) {

x \*= a;

y \*= a;

z \*= a;

}

**Vector::~Vector**() {}

* Файл main.cpp

**#include** <iostream>

**#include** "Vector.h"

**using** **namespace** std;

**int** **main**() {

Vector v1;

cout << "L: " << v1.getLength() << **endl**;

**double** x, y, z;

cout << "Enter x: ";

cin >> x;

cout << "Enter y: ";

cin >> y;

cout << "Enter z: ";

cin >> z;

cout << "---------------" << **endl**;

v1.setX(x);

v1.setY(y);

v1.setZ(z);

cout << "L: " << v1.getLength() << **endl**;

cout << "---------------" << **endl**;

v1.prod(5);

cout << "L: " << v1.getLength() << **endl**;

cout << "---------------" << **endl**;

v1.set(1, 0, 0);

cout << "L: " << v1.getLength() << **endl**;

cout << "---------------" << **endl**;

Vector v2(3, 0, 0);

Vector v3(1, 1, 1, 1, 5, 1);

cout << "AlgProd: " << v2.algProd(v3) << **endl**;

v2.geomProd(v3);

cout << "GeomProd: (" << v2.getX() << "," << v2.getY() << ","

<< v2.getZ() << ")" << **endl**;

cout << "L: " << v2.getLength() << **endl**;

**system**("pause");

**return** 0;

}

1. Реализовать класс «Библиотека»

* Файл Book.h

**#include** <iostream>

**#include** <string>

**#include** <vector>

**#ifndef** BOOK\_H\_

**#define** BOOK\_H\_

**class** Book {

**int** pages; // количество страниц

std::vector<std::string> authors; // авторы

std::string title; // название

std::string genre; // жанр

std::string publisher; // издатель

**public**:

// Перегруженный оператор равенства для объектов данного класса

**friend** **bool** **operator==** (**const** Book& left, **const** Book& right);

**Book**(); // Создаёт пустой экземпляр

**Book**(**int**, **const** std::vector<std::string> &, std::string &, // Создаёт экземпляр с

**const** std::string &, **const** std::string &); // заполненными полями

**void** **setNumberOfPages**(**const** **int**); // Ввод количества страниц в книге

**int** **getNumberOfPages**(); // Получение количества страниц

**void** **setAuthors**(**const** std::vector<std::string> &); // Ввод авторов книги

**void** **addAuthor**(**const** std::string &); // Добавление одного автора к уже существующим

std::vector<std::string> &**getAuthors**(); // Получение авторов книги

**void** **setPublisher**(**const** std::string &); // Ввод издателя книги

std::string &**getPublisher**(); // Получение издателя книги

**void** **setGenre**(**const** std::string &); // Ввод жанра

std::string &**getGenre**(); // Получение жанра

**void** **setTitle**(**const** std::string &); // Ввод названия книги

std::string &**getTitle**(); // Получение названия книги

**~Book**(); // Деструктор экземпляра

};

**#endif** /\* BOOK\_H\_ \*/

* Файл Book.cpp

**#include** "Book.h"

**bool** **operator==** (**const** Book& left, **const** Book& right) {

**bool** res = **true**;

res = res && (left.pages == right.pages);

res = res && (left.authors == right.authors);

res = res && (left.genre == right.genre);

res = res && (left.publisher == right.publisher);

res = res && (left.title == right.title);

**return** res;

}

**Book::Book**() {

**this**->pages = 0;

**this**->authors.clear();

**this**->genre = "";

**this**->publisher = "";

}

**Book::Book**(**int** pages, **const** std::vector<std::string> &authors, std::string &title,

**const** std::string &genre, **const** std::string &publisher) {

**this**->pages = pages;

**this**->authors = authors;

**this**->title = title;

**this**->genre = genre;

**this**->publisher = publisher;

}

**void** **Book::setAuthors**(**const** std::vector<std::string> &authors) {

**this**->authors = authors;

}

**void** **Book::addAuthor**(**const** std::string &author) {

authors.push\_back(author);

}

std::vector<std::string> &**Book::getAuthors**() {

**if**(!authors.size()) {

std::cout << "The book has no author." << std::**endl**;

}

**return** authors;

}

**void** **Book::setNumberOfPages**(**const** **int** pages) { **this**->pages = pages; }

**int** **Book::getNumberOfPages**() { **return** pages; }

**void** **Book::setPublisher**(**const** std::string & publisher) {

**this**->publisher = publisher;

}

std::string &**Book::getPublisher**() {

**if** (!publisher.size()) {

std::cout << "The book has no publisher." << std::**endl**;

}

**return** publisher;

}

**void** **Book::setGenre**(**const** std::string &genre) { **this**->genre = genre; }

std::string &**Book::getGenre**() {

**if** (!genre.size()) {

std::cout << "The book has no genre." << std::**endl**;

}

**return** genre;

}

**void** **Book::setTitle**(**const** std::string &title) { **this**->title = title; }

std::string &**Book::getTitle**() {

**if** (!title.size()) {

std::cout << "The book has no title." << std::**endl**;

}

**return** title;

}

**Book::~Book**() {

authors.clear();

genre.clear();

publisher.clear();

}

* Файл Library.h

**#include** <string>

**#include** <vector>

**#include** <iostream>

**#include** "Book.h"

**#ifndef** LIBRARY\_H\_

**#define** LIBRARY\_H\_

**class** Library {

std::vector<Book> books; // Вектор, хранящий экземпляры книг

**int** caseSize; // Размер шкафа

**int** shelfSize; // Размер полки шкафа

**public**:

**Library**(); // Создает объект, заполненный нулями

**Library**(**int**, **int**); // Создает объект с заданными размерами

**Library**(**int**, **int**, **const** std::vector<Book> &); // Создает объект с размерами и книгами

**void** **setSize**(**int**, **int**); // Задает размер шафа и полки

**void** **addBooks**(**const** std::vector<Book> &); // Добавляет книги в библиотеку

**void** **eraseBook**(**int**); // Удаляет книги из библиотеки

**void** **eraseBook**(**int**, **int**, **int**);

**void** **findBook**(Book &); // Находит местоположение книги

**int** **getCaseSize**(); // Возвращает размер шкафа

**int** **getShelfSize**(); // Возвращает размер полки шкафа

Book &**getBook**(**int**); // Возвращает книгу, имеющую данный id

Book &**getBook**(**int**, **int**, **int**); // Выдает книгу по её местоположению

**void** **printIdList**(); // Выводит книги и соответствующие им id

**~Library**();

};

**#endif** /\* LIBRARY\_H\_ \*/

* Файл Library.cpp

**#include** "Library.h"

**Library::Library**() {

caseSize = 0;

shelfSize = 0;

books.clear();

}

**Library::Library**(**int** caseSize, **int** shelfSize) {

setSize(caseSize, shelfSize);

}

**Library::Library**(**int** caseSize, **int** shelfSize, **const** std::vector<Book> &books) {

setSize(caseSize, shelfSize);

addBooks(books);

}

**void** **Library::setSize**(**int** caseSize, **int** shelfSize) {

**if** (caseSize > 0 && shelfSize > 0) {

**this**->caseSize = caseSize;

**this**->shelfSize = shelfSize;

} **else** {

std::cout << "Warning: cannot set size" << std::**endl** <<

"\t(It must be more than 0)" << std::**endl**;

}

}

**void** **Library::addBooks**(**const** std::vector<Book> &books) {

**for**(**auto** a : books) {

**if** ((a.getAuthors().size() != 0) && (a.getTitle() != "") && (a.getNumberOfPages() != 0)) {

**this**->books.push\_back(a);

} **else** {

std::cout << "Warning: one book cannot be added" << std::**endl**;

}

}

}

**void** **Library::eraseBook**(**int** id) {

books.erase(books.begin() + id);

}

**void** **Library::eraseBook**(**int** bookcase, **int** shelf, **int** position) {

**int** id = (bookcase - 1) \* caseSize + (shelf - 1) \* shelfSize + position - 1;

eraseBook(id);

}

**void** **Library::findBook**(Book &book) {

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

**if** (book == books[i]) {

**int** c, s, p;

c = i / caseSize + 1;

s = (i % caseSize) / shelfSize + 1;

p = (i % caseSize) % shelfSize + 1;

std::cout << "Book was found in the " << c << " bookcase " <<

s << " bookshelf " << p << " position" << std::**endl**;

}

}

}

**int** **Library::getCaseSize**() { **return** caseSize; }

**int** **Library::getShelfSize**() { **return** shelfSize; }

Book &**Library::getBook**(**int** id) { **return** books[id]; }

Book &**Library::getBook**(**int** bookcase, **int** shelf, **int** position) {

**int** id = (bookcase - 1) \* caseSize + (shelf - 1) \* shelfSize + position - 1;

**return** getBook(id);

}

**void** **Library::printIdList**() {

std::cout << "ID List:" << std::**endl**;

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

std::cout << i << "\t" << books[i].getTitle() << ' '

<< books[i].getAuthors()[0] << std::**endl**;

}

}

**Library::~Library**() {}

* Файл main.cpp

**#include** <iostream>

**#include** "Library.h"

**#include** "Book.h"

**#include** <Windows.h>

**using** **namespace** std;

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

vector<Book> b;

**for** (**int** i = 0; i < 3; i++) {

string s;

Book book;

cout << "Book " << i + 1 << ":" << **endl**;

cout << "Title: ";

cin >> s;

book.setTitle(s);

cout << "Author: ";

cin >> s;

book.addAuthor(s);

**int** a;

cout << "Number of pages: ";

cin >> a;

book.setNumberOfPages(a);

b.push\_back(book);

}

Library l(10, 5, b);

l.printIdList();

l.findBook(b[2]);

l.eraseBook(1, 1, 3);

l.printIdList();

**system**("pause");

**return** 0;

}

**Вывод**

Все поставленные задачи были выполнены. Для этого были использованы библиотека STL, перегрузка оператора равенства, шаблоны функций. Все задачи были выполнены в виде много файлового проекта с разбиением классов на файлы объявления и реализации класса. Результаты работы программ приведены в рисунках (рис. 7-11). Код программ также доступен на GitHub по ссылкам:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr3.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr3.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Pr3.3>
4. <https://github.com/IvLaptev/OOP/tree/master/Pr3.4>
5. <https://github.com/IvLaptev/OOP/tree/master/Pr3.5>

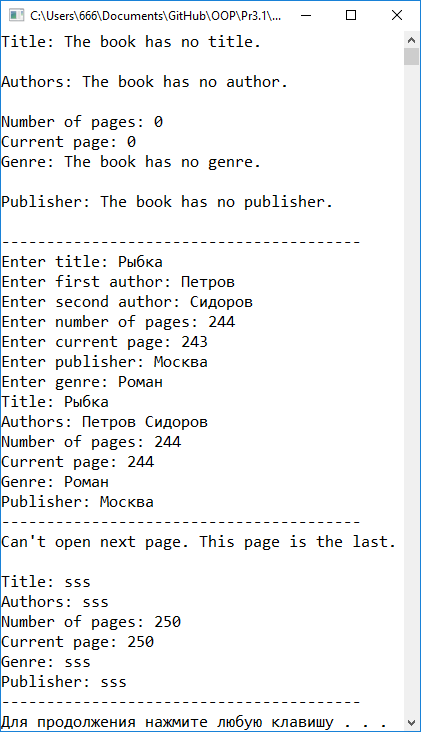
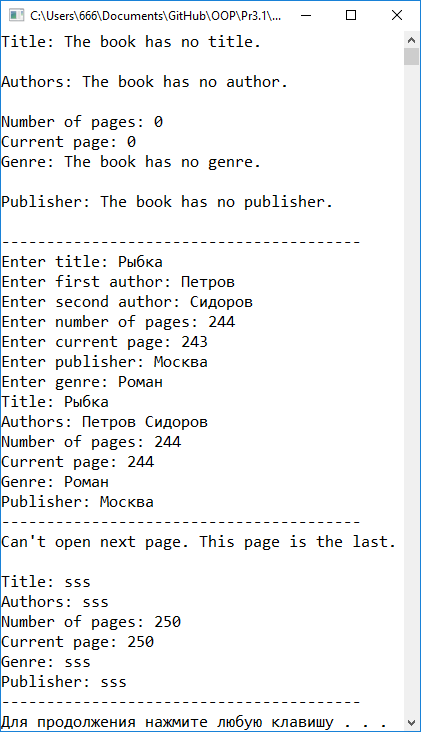


Рис. 7 Результат работы программы №1

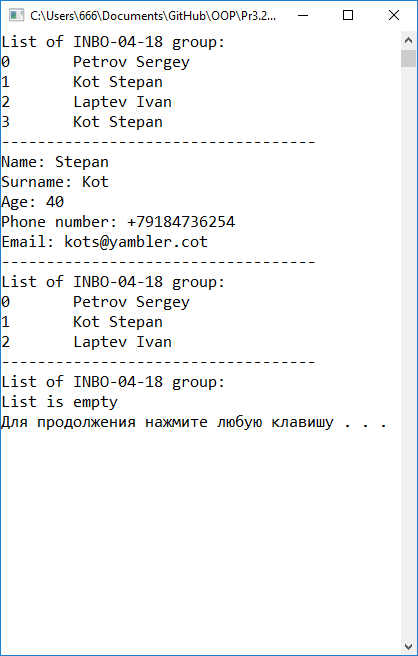
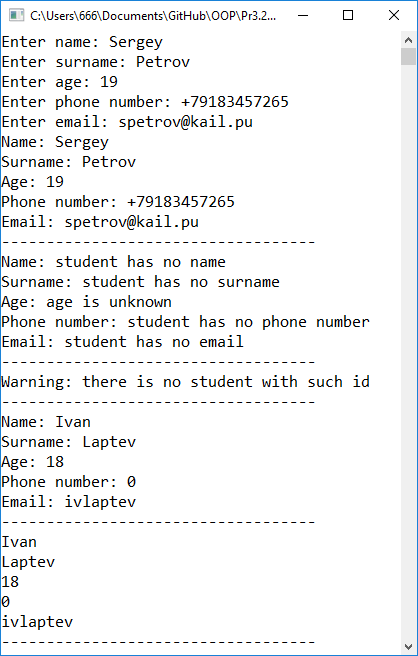


Рис. 8 Результат работы программы №2

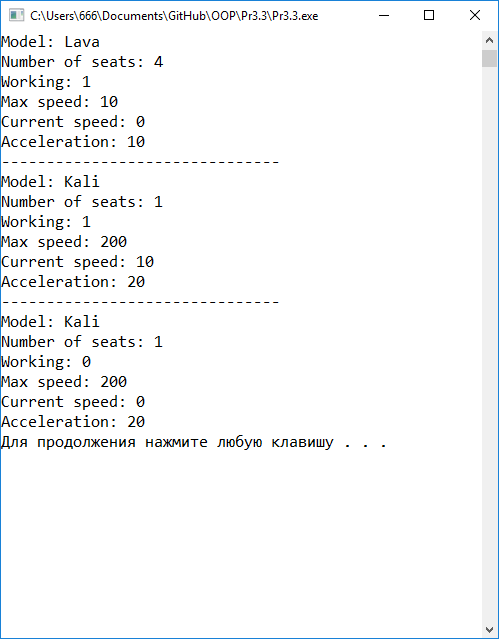
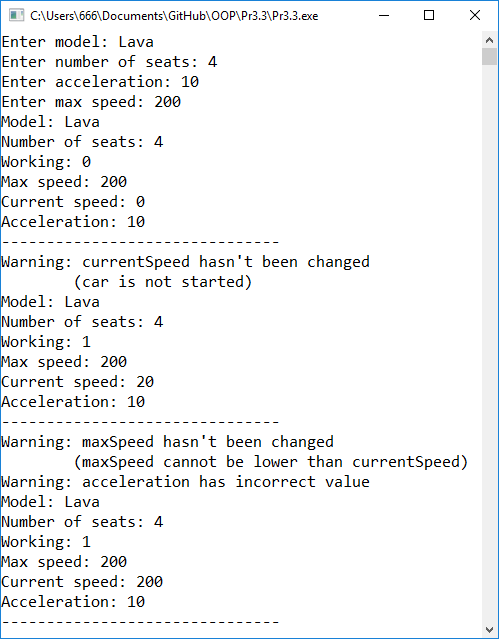


Рис. 9 Результат работы программы №4

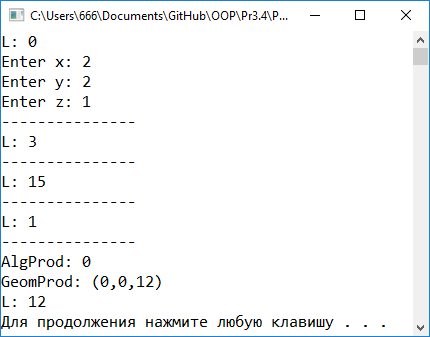


Рис 10 Результат работы программы №5

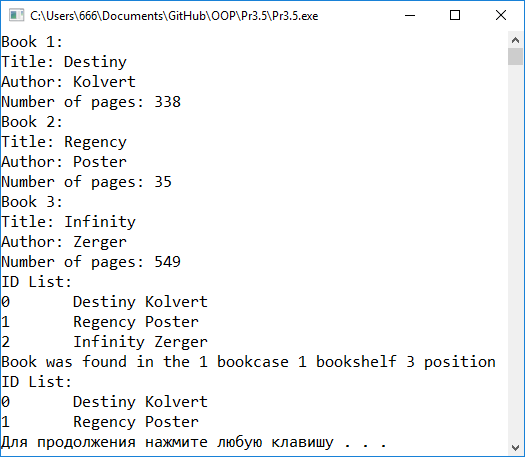


Рис. 11 Результат работы программы №5

## Практическая работа №4

**Динамическое выделение памяти для объектов в языке С++**

**Цель практической работы**

Целью данной практической работы является изучение динамического выделения и удаления памяти, а так же работа с указателями на языке С++.

**Задачи**

1. Объявите указатель на массив типа double и предложите пользователю выбрать его размер. Далее напишите четыре функции: первая должна выделить память для массива, вторая – заполнить ячейки данными, третья – показать данные на экран, четвёртая – освободить занимаемую память. Программа должна предлагать продолжать работу (создавать новые массивы) или выйти из программы.
2. Объявите указатель на массив типа int и выделите память для 12-ти элементов. Необходимо написать функцию, которая поменяет значения четных и нечетных ячеек массива.
3. Объявить и заполнить двумерный динамический массив случайными числами от 10 до 50. Показать его на экран. Для заполнения и показа на экран написать отдельные функции. Количество строк и столбцов выбирает пользователь

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <ctime>

**using** **namespace** std;

**double** \***createArray**(**int**); // Создаёт массив

**void** **fillArray**(**double** \*, **int**); // Заполняет массив случайными числами

**void** **printArray**(**double** \*, **int**); // Выводит массив на экран

**void** **deleteArray**(**double** \*, **int**); // Удаляет массив

**int** **main**() {

**srand**(**time**(NULL));

**int** size;

cout << "Enter size of massive(lower than 1 if you want to exit the program): ";

cin >> size;

**while** (size > 0) {

**double** \*arr = createArray(size);

fillArray(arr, size);

printArray(arr, size);

deleteArray(arr, size);

cout << "Enter size of massive(lower than 1 if you want to exit the program): ";

cin >> size;

}

**return** 0;

}

**double** \***createArray**(**int** size) { **return** **new** **double**[size]; }

**void** **fillArray**(**double** \*arr, **int** size) {

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

arr[i] = **rand**() % 100;

}

}

**void** **printArray**(**double** \*arr, **int** size) {

cout << "Array: ";

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **deleteArray**(**double** \*arr, **int** size) { **delete** [] arr; }

1. Программа №2

**#include** <iostream>

**#include** <ctime>

**using** **namespace** std;

**void** **fillArray**(**int** \*, **int**); // Заполняет массив случайными числами

**void** **printArray**(**int** \*, **int**); // Выводит массив на экран

**void** **changeElements**(**int** \*, **int**); // Меняет местами пары элементов массива

**int** **main**() {

**srand**(**time**(NULL));

**int** size = 12;

**int** \*arr = **new** **int**[size];

fillArray(arr, size);

printArray(arr, size);

changeElements(arr, 12);

printArray(arr, size);

**delete** [] arr;

**system**("pause");

**return** 0;

}

**void** **fillArray**(**int** \*arr, **int** size) {

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

arr[i] = **rand**() % 100;

}

}

**void** **printArray**(**int** \*arr, **int** size) {

cout << "Array: ";

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **changeElements**(**int** \*arr, **int** size) {

**for** (**int** i = 0; i < size; i = i + 2) {

**int** c = arr[i];

arr[i] = arr[i+1];

arr[i+1] = c;

}

}

1. Программа №3

**#include** <iostream>

**#include** <ctime>

**using** **namespace** std;

**void** **fillArray**(**int** \*\*, **int**, **int**); // Заполняет массив случайными числами

**void** **printArray**(**int** \*\*, **int**, **int**); // Выводит массив на экран

**int** **main**() {

**srand**(**time**(NULL));

**int** m, n;

cout << "Enter number of rows and columns: ";

cin >> n >> m;

**int** \*\*arr = **new** **int**\*[n];

**for** (**int** i = 0; i < n; i++) {

arr[i] = **new** **int**[m];

}

fillArray(arr, n, m);

printArray(arr, n, m);

**for** (**int** i = 0; i < n; i++) {

**delete** [] arr[i];

}

**delete** [] arr;

**system**("pause");

**return** 0;

}

**void** **fillArray**(**int** \*\*arr, **int** n, **int** m) {

**for** (**int** i = 0; i < n; i++) {

**for** (**int** j = 0; j < m; j++) {

arr[i][j] = **rand**() % 41 + 10;

}

}

}

**void** **printArray**(**int** \*\*arr, **int** n, **int** m) {

cout << "Array:" << **endl**;

**for** (**int** i = 0; i < n; i++) {

**for** (**int** j = 0; j < m; j++) {

cout << arr[i][j] << ' ';

}

cout << **endl**;

}

}

**Вывод**

Все поставленные задачи были выполнены. При выполнении использовались массивы с динамическим выделением памяти, которые создавались с помощью генератора случайных чисел. Для генератора случайных чисел была подключена библиотека ctime. Результат работы программ приведён на рисунках (рис.12-14). Так же исходный код программ доступен на GitHub по ссылкам:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr4.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr4.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Pr4.3>

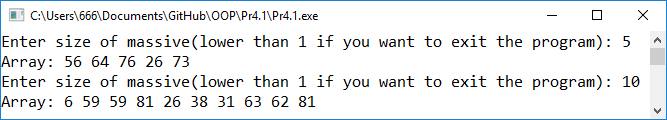


Рис. 12 Результат работы программы №1

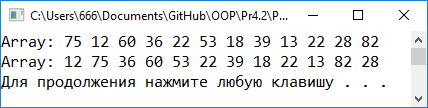


Рис. 13 Результат работы программы №2

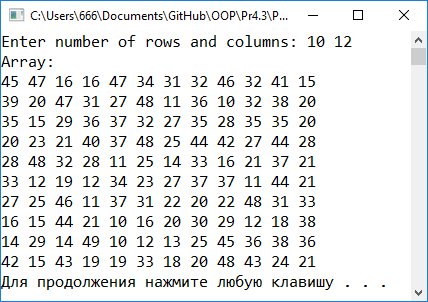


Рис. 14 Результат работы программы №3

## Практическая работа №5

**Перегрузка функций**

**Цель практической работы**

Целью данной практической работы является приобретение практических навыков по программированию перегрузки функций на языке C++.

**Задачи**

1. Реализовать сортировку пузырьком для целых чисел, а затем перегрузить её для дробных.
2. Реализовать сортировку выбором для целых чисел, а затем перегрузить её для дробных.
3. Реализовать сортировку вставками для целых чисел, а затем перегрузить её для дробных.
4. Реализовать программу-калькулятор, работающую с разными типами данных.

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <vector>

**#include** <ctime>

**using** **namespace** std;

**void** **printArray**(**const** vector<**int**> &); // Выводит массив на экран

**void** **printArray**(**const** vector<**double**> &);

**void** **fillArray**(vector<**int**> &, **int**); // Генерирует элементы массива

**void** **fillArray**(vector<**double**> &, **int**);

**void** **bubbleSort**(vector<**int**> &); // Сортирует пузырьком

**void** **bubbleSort**(vector<**double**> &);

**int** **main**() {

**srand**(**time**(NULL));

**int** size;

cout << "Enter size of arrays: ";

cin >> size;

vector<**int**> arrI;

vector<**double**> arrD;

fillArray(arrI, size);

fillArray(arrD, size);

cout << "Input arrays: " << **endl**;

cout << "Int:\t";

printArray(arrI);

cout << "Double:\t";

printArray(arrD);

bubbleSort(arrI);

bubbleSort(arrD);

cout << "Output arrays: " << **endl**;

cout << "Int:\t";

printArray(arrI);

cout << "Double:\t";

printArray(arrD);

**system**("pause");

**return** 0;

}

**void** **printArray**(**const** vector<**int**> &arr) {

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **printArray**(**const** vector<**double**> &arr) {

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **bubbleSort**(vector<**int**> &arr) {

**for** (**int** i = arr.size() - 1; i > 0; i--) {

**for** (**int** j = 0; j < i; j++) {

**if** (arr[j] > arr[j + 1]) {

**int** c = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = c;

}

}

}

}

**void** **bubbleSort**(vector<**double**> &arr) {

**for** (**int** i = arr.size() - 1; i > 0; i--) {

**for** (**int** j = 0; j < i; j++) {

**if** (arr[j] > arr[j + 1]) {

**double** c = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = c;

}

}

}

}

**void** **fillArray**(vector<**int**> &arr, **int** size) {

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

arr.push\_back(**rand**() % 100);

}

}

**void** **fillArray**(vector<**double**> &arr, **int** size) {

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

arr.push\_back(((**double**)(**rand**() % 1000)) / 10);

}

}

1. Программа №2

**#include** <iostream>

**#include** <vector>

**#include** <ctime>

**using** **namespace** std;

**void** **printArray**(**const** vector<**int**> &); // Выводит массив на экран

**void** **printArray**(**const** vector<**double**> &);

**void** **fillArray**(vector<**int**> &, **int**); // Генерирует элементы массива

**void** **fillArray**(vector<**double**> &, **int**);

**void** **selectionSort**(vector<**int**> &); // Сортирует пузырьком

**void** **selectionSort**(vector<**double**> &);

**int** **main**() {

**srand**(**time**(NULL));

**int** size;

cout << "Enter size of arrays: ";

cin >> size;

vector<**int**> arrI;

vector<**double**> arrD;

fillArray(arrI, size);

fillArray(arrD, size);

cout << "Input arrays: " << **endl**;

cout << "Int:\t";

printArray(arrI);

cout << "Double:\t";

printArray(arrD);

selectionSort(arrI);

selectionSort(arrD);

cout << "Output arrays: " << **endl**;

cout << "Int:\t";

printArray(arrI);

cout << "Double:\t";

printArray(arrD);

**system**("pause");

**return** 0;

}

**void** **printArray**(**const** vector<**int**> &arr) {

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **printArray**(**const** vector<**double**> &arr) {

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **selectionSort**(vector<**int**> &arr) {

**for** (**int** i = arr.size() - 1; i > 0; i--) {

**int** maxn = 0;

**for** (**int** j = 1; j <= i; j++) {

**if** (arr[maxn] < arr[j]) {

maxn = j;

}

}

**int** c = arr[maxn];

arr[maxn] = arr[i];

arr[i] = c;

}

}

**void** **selectionSort**(vector<**double**> &arr) {

**for** (**int** i = arr.size() - 1; i > 0; i--) {

**int** maxn = 0;

**for** (**int** j = 1; j <= i; j++) {

**if** (arr[maxn] < arr[j]) {

maxn = j;

}

}

**double** c = arr[maxn];

arr[maxn] = arr[i];

arr[i] = c;

}

}

**void** **fillArray**(vector<**int**> &arr, **int** size) {

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

arr.push\_back(**rand**() % 100);

}

}

**void** **fillArray**(vector<**double**> &arr, **int** size) {

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

arr.push\_back(((**double**)(**rand**() % 1000)) / 10);

}

}

1. Программа №3

**#include** <iostream>

**#include** <vector>

**#include** <ctime>

**using** **namespace** std;

**void** **printArray**(**const** vector<**int**> &); // Выводит массив на экран

**void** **printArray**(**const** vector<**double**> &);

**void** **fillArray**(vector<**int**> &, **int**); // Генерирует элементы массива

**void** **fillArray**(vector<**double**> &, **int**);

**void** **insSort**(vector<**int**> &); // Сортирует пузырьком

**void** **insSort**(vector<**double**> &);

**int** **main**() {

**srand**(**time**(NULL));

**int** size;

cout << "Enter size of arrays: ";

cin >> size;

vector<**int**> arrI;

vector<**double**> arrD;

fillArray(arrI, size);

fillArray(arrD, size);

cout << "Input arrays: " << **endl**;

cout << "Int:\t";

printArray(arrI);

cout << "Double:\t";

printArray(arrD);

insSort(arrI);

insSort(arrD);

cout << "Output arrays: " << **endl**;

cout << "Int:\t";

printArray(arrI);

cout << "Double:\t";

printArray(arrD);

**system**("pause");

**return** 0;

}

**void** **printArray**(**const** vector<**int**> &arr) {

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **printArray**(**const** vector<**double**> &arr) {

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

cout << arr[i] << ' ';

}

cout << **endl**;

}

**void** **insSort**(vector<**int**> &arr) {

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

**for** (**int** j = i; j > 0; j--) {

**if** (arr[j] < arr[j - 1]) {

**int** c = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = c;

} **else** {

**break**;

}

}

}

}

**void** **insSort**(vector<**double**> &arr) {

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

**for** (**int** j = i; j > 0; j--) {

**if** (arr[j] < arr[j - 1]) {

**double** c = arr[j];

arr[j] = arr[j - 1];

arr[j - 1] = c;

} **else** {

**break**;

}

}

}

}

**void** **fillArray**(vector<**int**> &arr, **int** size) {

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

arr.push\_back(**rand**() % 100);

}

}

**void** **fillArray**(vector<**double**> &arr, **int** size) {

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

arr.push\_back(((**double**)(**rand**() % 1000)) / 10);

}

}

1. Программа №4

**#include** <iostream>

**#include** <ctime>

**using** **namespace** std;

**void** **keyboardInput**(**int**&, **int**&, **double**&, **double**&); // Ввод чисел с клавиатуры

**void** **randInput**(**int**&, **int**&, **double**&, **double**&); // Генерация случяйных чисел

**void** **sum**(**int**, **int**); // Сумма и разность чисел

**void** **sum**(**double**, **double**);

**void** **prod**(**int**, **int**); // Произведение чисел

**void** **prod**(**double**, **double**);

**void** **quot**(**int**, **int**); // Частное чисел

**void** **quot**(**double**, **double**);

**int** **main**() {

cout << "Do you want to enter numbers with keyboard?" << **endl**;

cout << "1) Yes" << **endl** << "0) No" << **endl**;

**int** c;

cin >> c;

**int** ai, bi;

**double** ad, bd;

**if**(c) {

keyboardInput(ai, bi, ad, bd);

} **else** {

randInput(ai, bi, ad, bd);

}

cout << "Int:" << **endl**;

cout << "a = " << ai << "\tb = " << bi << **endl**;

sum(ai, bi);

prod(ai, bi);

quot(ai, bi);

cout << "---------------------" << **endl**;

cout << "Double:" << **endl**;

cout << "a = " << ad << "\tb = " << bd << **endl**;

sum(ad, bd);

prod(ad, bd);

quot(ad, bd);

**system**("pause");

**return** 0;

}

**void** **keyboardInput**(**int** &ai, **int** &bi, **double** &ad, **double** &bd) {

cout << "Enter int values:" << **endl**;

cout << "a = ";

cin >> ai;

cout << "b = ";

cin >> bi;

cout << "Enter double values:" << **endl**;

cout << "a = ";

cin >> ad;

cout << "b = ";

cin >> bd;

}

**void** **randInput**(**int** &ai, **int** &bi, **double** &ad, **double** &bd) {

**srand**(**time**(NULL));

ai = **rand**() % 100;

bi = **rand**() % 100;

ad = (**rand**() % 10000) / 100.0;

bd = (**rand**() % 10000) / 100.0;

}

**void** **sum**(**int** a, **int** b) {

cout << "Sum: " << (a + b) << **endl**;

cout << "Difference: " << (a - b) << **endl**;

}

**void** **sum**(**double** a, **double** b) {

cout << "Sum: " << (a + b) << **endl**;

cout << "Difference: " << (a - b) << **endl**;

}

**void** **prod**(**int** a, **int** b) { cout << "Product: " << a \* b << **endl**; }

**void** **prod**(**double** a, **double** b) { cout << "Product: " << a \* b << **endl**; }

**void** **quot**(**int** a, **int** b) { cout << "Quotient: " << (**float**)a/b << **endl**; }

**void** **quot**(**double** a, **double** b) { cout << "Quotient: " << a/b << **endl**; }

**Вывод**

Все задачи были выполнены. В ходе выполнения была изучена техника перегрузки функций. Так же для заполнения массивов использовался генератор случайных чисел. Результат работы программы приведён на рисунках (рис. 15-18). Исходный код программ так же доступен на GitHub по ссылкам:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr5.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr5.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Pr5.3>
4. <https://github.com/IvLaptev/OOP/tree/master/Pr5.4>

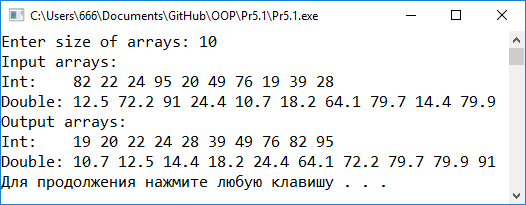


Рис. 15 Результат работы программы №1

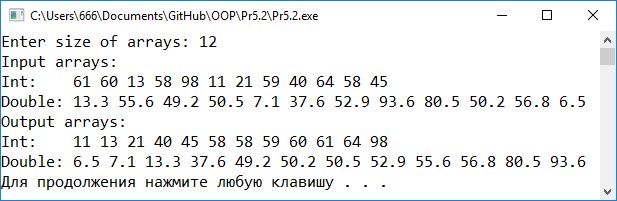


Рис. 16 Результат работы программы №2

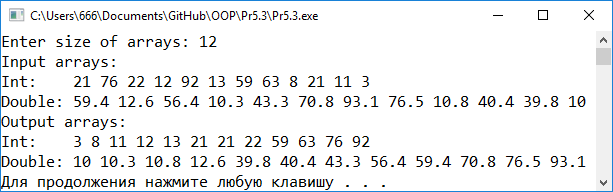


Рис. 17 Результат работы программы №3

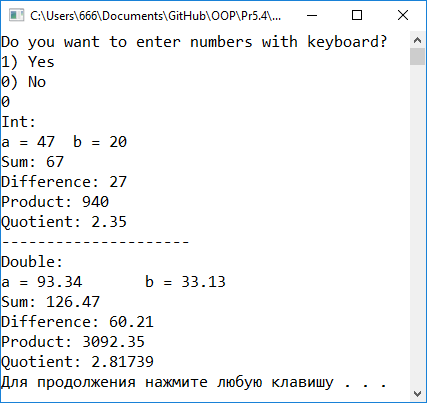


Рис. 18 Результат работы программы №4

## Практическая работа №6

**Решение задач на наследование на языке С++**

**Цель практической работы**

Целью данной практической работы является приобретение практических навыков использования принципа ООП – наследования для разработки программ на языке программирования C++.

**Задачи**

1. Написать иерархию классов, описывающих имущество налогоплательщиков. Она должна состоять из абстрактного базового класса Property и производных от него классов Appartment, Car и CountryHouse. Базовый класс должен иметь поле worth (стоимость), конструктор с одним параметром, заполняющий это поле, и чисто виртуальный метод рассчета налога, переопределенный в каждом из производных классов. Налог на квартиру вычисляется как 1/1000 ее стоимости, на машину – 1/200, на дачу – 1/500. Также, каждый производный класс должен иметь конструктор с одним параметром, передающий свой параметр конструктору базового класса. В функции main завести массив из 7 указателей на Property и заполнить его указателями на динамические объекты производных классов (первые 3 – Appartment, следующие 2 – Car и последние 2 – CountryHouse). Вывести на экран величину налога для всех 7 объектов. Не забудь также уничтожить динамические объекты перед завершением программы.
2. Написать набор классов, представляющий выражения. В этом наборе должен быть один абстрактный базовый тип, а также набор производных от него типов по видам выражений (константа, переменная, сумма, разность, произведение, частное, sin, cos, exp, ln). У каждого из классов должны быть следующие виртуальные функции: напечатать выражение (без параметров), вычислить выражение (параметр – значение переменной, результат – значение выражения), вернуть производную выражения (без параметров), создать копию выражения (тоже без параметров).

**Ход работы**

1. Программа №1

**#include** <iostream>

**using** **namespace** std;

**class** Property { // Базовый класс "Собственность"

**protected**:

**double** worth;

**public**:

**Property**(**double** worth) {

**this**->worth = worth;

}

**virtual** **double** **calculateTax**() = 0;

**virtual** **~Property**() {

}

};

**class** Appartment : **public** Property { // Производный класс "Квартира"

**public**:

**Appartment**(**double** worth) : Property(worth) {};

**double** **calculateTax**() { **return** worth/1000; }

};

**class** Car : **public** Property { // Производный класс "Автомобиль"

**public**:

**Car**(**double** worth) : Property(worth) {};

**double** **calculateTax**() { **return** worth/200; }

};

**class** CountryHouse : **public** Property { // Производный класс "Загородный дом"

**public**:

**CountryHouse**(**double** worth) : Property(worth) {};

**double** **calculateTax**() { **return** worth/500; }

};

**int** **main**() {

Property \*p[7];

**for** (**int** i = 0; i < 7; i++) {

**double** worth;

cout << "Enter worth of the " << i + 1 << " : ";

cin >> worth;

**if** (i < 3) {

p[i] = **new** Appartment(worth);

} **else** {

**if** (i < 5) {

p[i] = **new** Car(worth);

} **else** {

p[i] = **new** CountryHouse(worth);

}

}

}

cout << "Taxes:" << **endl**;

**for** (**int** i = 0; i < 7; i++) {

cout << i + 1 << ". " << p[i]->calculateTax() << **endl**;

}

**system**("pause");

**delete** [] p;

**return** 0;

1. Программа №2

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**class** Function { // Базовый класс функция

**public**:

**virtual** **void** **printFunction**() = 0; // Выводит функцию на экран

**virtual** **double** **count**(**double** x) = 0; // Считает значение функции от аргумента

**virtual** **void** **takeDerivative**() = 0; // Выводит производную

};

**class** Const : **public** Function{ // Класс, описывающий константу

**double** c;

**public**:

**Const**(**double** c) { **this**->c = c; }

**void** **printFunction**() { cout << "Function: f(x) = " << c << **endl**; }

**double** **count**(**double** x) { **return** c; }

**void** **takeDerivative**() { cout << "Derivative: 0" << **endl**; }

};

**class** Variable : **public** Function { // Класс, описывающий переменную

**public**:

**void** **printFunction**() { cout << "Function: f(x) = x" << **endl**; }

**double** **count**(**double** x) { **return** x; }

**void** **takeDerivative**() { cout << "Derivative: 1" << **endl**; }

};

**class** Sum : **public** Function { // Класс, описывающий сумму двух чисел

**double** y;

**public**:

**Sum**(**double** y) { **this**->y = y; }

**void** **printFunction**() { cout << "Function: f(x) = x + y" << **endl**; }

**double** **count**(**double** x) { **return** x + y; }

**void** **takeDerivative**() { cout << "Derivative: 2" << **endl**; }

};

**class** Difference : **public** Function { // Класс, описывающий разность двух чисел

**double** y;

**public**:

**Difference**(**double** y) { **this**->y = y; }

**void** **printFunction**() { cout << "Function: f(x) = x - y" << **endl**; }

**double** **count**(**double** x) { **return** x - y; }

**void** **takeDerivative**() { cout << "Derivative: 0" << **endl**; }

};

**class** Product : **public** Function { // Класс, описывающий произведение двух чисел

**double** y;

**public**:

**Product**(**double** y) { **this**->y = y; }

**void** **printFunction**() { cout << "Function: f(x) = x \* y" << **endl**; }

**double** **count**(**double** x) { **return** x \* y; }

**void** **takeDerivative**() { cout << "Derivative: y \* dx" << **endl**; }

};

**class** Quotient : **public** Function { // Класс, описывающий частное двух чисел

**double** y;

**public**:

**Quotient**(**double** y) { **this**->y = y; }

**void** **printFunction**() { cout << "Function: f(x) = x / y" << **endl**; }

**double** **count**(**double** x) { **return** x / y; }

**void** **takeDerivative**() { cout << "Derivative: dx / y" << **endl**; }

};

**class** Sin : **public** Function { // Класс, описывающий синус угла в радианах

**public**:

**void** **printFunction**() { cout << "Function: f(x) = sin(x)" << **endl**; }

**double** **count**(**double** x) { **return** **sin**(x); }

**void** **takeDerivative**() { cout << "Derivative: cos(x)" << **endl**; }

};

**class** Cos : **public** Function { // Класс, описывающий косинус угла в радианах

**public**:

**void** **printFunction**() { cout << "Function: f(x) = cos(x)" << **endl**; }

**double** **count**(**double** x) { **return** **cos**(x); }

**void** **takeDerivative**() { cout << "Derivative: -sin(x)" << **endl**; }

};

**class** Exp : **public** Function { // Класс, описывающий возведение экспоненты в степень

**public**:

**void** **printFunction**() { cout << "Function: f(x) = e^x" << **endl**; }

**double** **count**(**double** x) { **return** **exp**(x); }

**void** **takeDerivative**() { cout << "Derivative: e^x" << **endl**; }

};

**class** Ln : **public** Function { // Класс, описывающий взятие натурального логарифма

**public**:

**void** **printFunction**() { cout << "Function: f(x) = ln(x)" << **endl**; }

**double** **count**(**double** x) { **return** **log**(x); }

**void** **takeDerivative**() { cout << "Derivative: 1/x" << **endl**; }

};

**int** **main**() {

**double** y, x;

cout << "Constant" << **endl** << "Enter value: ";

cin >> y;

Const c(y);

cout << "Enter argument: ";

cin >> x;

c.printFunction();

cout << "f(" << x << ") = " << c.count(x) << **endl**;

c.takeDerivative();

cout << **endl** << "Variable" << **endl** << "Enter argument: ";

cin >> x;

Variable v;

v.printFunction();

cout << "f(" << x << ") = " << v.count(x) << **endl**;

v.takeDerivative();

cout << **endl** << "Sum" << **endl** << "Enter x and y: ";

cin >> x >> y;

Sum s(y);

s.printFunction();

cout << "f(" << x << ") = " << s.count(x) << **endl**;

s.takeDerivative();

cout << **endl** << "Difference" << **endl** << "Enter x and y: ";

cin >> x >> y;

Difference d(y);

d.printFunction();

cout << "f(" << x << ") = " << d.count(x) << **endl**;

d.takeDerivative();

cout << **endl** << "Product" << **endl** << "Enter x and y: ";

cin >> x >> y;

Product p(y);

p.printFunction();

cout << "f(" << x << ") = " << p.count(x) << **endl**;

p.takeDerivative();

cout << **endl** << "Quotient" << **endl** << "Enter x and y: ";

cin >> x >> y;

Quotient q(y);

q.printFunction();

cout << "f(" << x << ") = " << q.count(x) << **endl**;

q.takeDerivative();

cout << **endl** << "Sin" << **endl** << "Enter argument: ";

cin >> x;

Sin si;

si.printFunction();

cout << "f(" << x << ") = " << si.count(x) << **endl**;

si.takeDerivative();

cout << **endl** << "Cos" << **endl** << "Enter argument: ";

cin >> x;

Cos co;

co.printFunction();

cout << "f(" << x << ") = " << co.count(x) << **endl**;

co.takeDerivative();

cout << **endl** << "Exp" << **endl** << "Enter argument: ";

cin >> x;

Exp e;

e.printFunction();

cout << "f(" << x << ") = " << e.count(x) << **endl**;

e.takeDerivative();

cout << **endl** << "Ln" << **endl** << "Enter argument: ";

cin >> x;

Ln l;

l.printFunction();

cout << "f(" << x << ") = " << l.count(x) << **endl**;

l.takeDerivative();

**system**("pause");

**return** 0;

}

**Вывод**

Все поставленные задачи были выполнены. В ходе выполнения работы был изучен механизм наследования классов в C++. Также для выполнения работы были использованы абстрактные классы и чисто виртуальные методы базовых абстрактных классов. Результаты работы программ представлены на рисунках (рис 19, 20). Исходный код программ также доступен на GitHub по ссылкам:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr6.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr6.2>

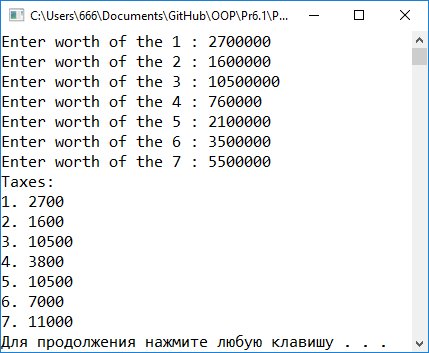


Рис. 19 Результат работы программы №1

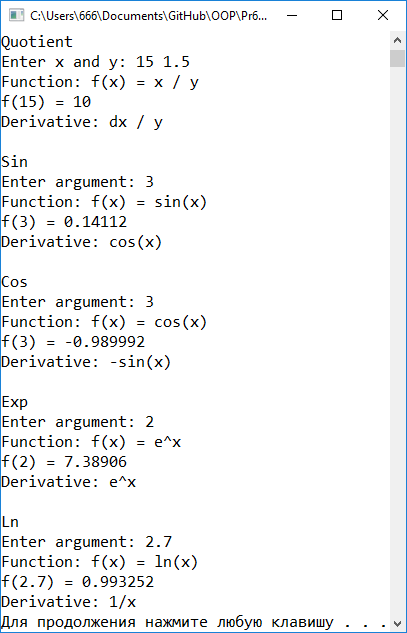
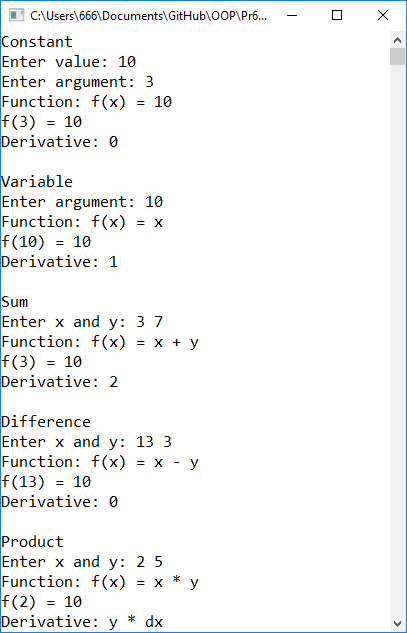


Рис. 20 Результат работы программы №2

## Практическая работа №7

**Абстрактные классы. Множественное наследование**

**Цель практической работы**

Целью данной практической работы является знакомство с абстрактными классами и механизмом виртуальных функций, а так же множественным наследованием на языке программирования C++.

**Задачи**

1. Реализовать абстрактный класс «Животное» и путём наследования от него получить классы «Кошка», «Собака», «Попугай».
2. Реализовать абстрактный класс «Фигура» и путём наследования от него получить абстрактный класс «Четырёхугольник», и затем путём наследования получить классы «Ромб», «Прямоугольник».
3. Реализовать абстрактный класс «Транспортное средство» и путём наследования от него получить классы «Автомобиль», «Автобус», «Велосипед».
4. Реализовать абстрактные классы «Экран» и «Клавиатура», путём наследования от них получить классы «Ноутбук», «Телефон», «Стационарный компьютер».

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <string>

**using** **namespace** std;

**class** Animal { // базовый класс "Животное"

**protected**:

**int** legs;

string name;

**int** age;

**public**:

**Animal**(string&, **int**, **int**);

**void** **setName**(string &name) { **this**->name = name; }

string &**getName**() { **return** name; }

**void** **setAge**(**int** age) { **this**->age = age; }

**int** **getAge**() { **return** age; }

**void** **setLegs**(**int** legs) { **this**->legs = legs; }

**int** **getLegs**() { **return** legs; }

**virtual** **void** **say**() = 0; // Чисто виртуальная функция, выводящая на экран речь животного

};

**class** Cat : **public** Animal { // Производный класс "Кошка"

**public**:

**Cat**(string &name, **int** legs, **int** age) : Animal(name, legs, age) {}

**void** **say**() { cout << "Meow" << **endl**; }

};

**class** Dog : **public** Animal { // Производный класс "Собака"

**public**:

**Dog**(string &name, **int** legs, **int** age) : Animal(name, legs, age) {}

**void** **say**() { cout << "Woof" << **endl**; }

};

**class** Parrot : **public** Animal { // Производный класс "Попугай"

**public**:

**Parrot**(string &name, **int** legs, **int** age) : Animal(name, legs, age) {}

**void** **say**() { cout << "I am man!" << **endl**; }

};

**int** **main**() {

string s1 = "Mur", s2 = "Doggy", s3 = "Vasili";

Cat c(s1, 4, 5);

Dog d(s2, 4, 7);

Parrot p(s3, 2, 6);

cout << "Cat says - ";

c.say();

cout << "Dog says - ";

d.say();

cout << "And only parrot says - ";

p.say();

cout << "But everyone of them are animals" << **endl**;

**system**("pause");

**return** 0;

}

**Animal::Animal**(string &name, **int** legs, **int** age) {

**this**->name = name;

**this**->age = age;

**this**->legs = legs;

}

1. Программа №2

**#include** <iostream>

**#include** <vector>

**#include** <cmath>

**using** **namespace** std;

**class** Figure { // Базовый абстрактный класс "Фигура"

**protected**:

**int** vertices; // Количество вершин

**public**:

**Figure**(**int** vertices) { **this**->vertices = vertices; }

**virtual** **double** **getSquare**() = 0; // Виртуальный метод, считающий площадь

};

**class** Quadrilateral : **public** Figure{ // Абстрактный класс "Четырёхугольник"

**protected**:

vector<**double**> x; // х составляющая координат точек

vector<**double**> y; // у составляющая координат точек

**public**:

**Quadrilateral**(vector<**double**>&, vector<**double**>&);

**virtual** **double** **getSquare**() = 0;

};

**class** Rhombus : **public** Quadrilateral { // Класс "Ромб"

**public**:

**Rhombus**(vector<**double**> &x, vector<**double**> &y) : Quadrilateral(x,y) {}

**double** **getSquare**(); // Определённый метод для подсчёта площади фигуры

**~Rhombus**() { }

};

**class** Rectangle : **public** Quadrilateral { // Класс "Прямоугольник"

**public**:

**Rectangle**(vector<**double**> &x, vector<**double**> &y) : Quadrilateral(x, y) {}

**double** **getSquare**(); // Определённый метод для подсчёта площади фигуры

**~Rectangle**() {}

};

**int** **main**() {

vector<**double**> x1(4);

vector<**double**> y1(4);

vector<**double**> x2(4);

vector<**double**> y2(4);

**for** (**int** i = 0; i < 4; i++) {

cout << "Enter rhombus vertex " << i + 1 << ": ";

cin >> x1[i] >> y1[i];

}

**for** (**int** i = 0; i < 4; i++) {

cout << "Enter rectangle vertex " << i + 1 << ": ";

cin >> x2[i] >> y2[i];

}

Rhombus r(x1, y1);

Rectangle rect(x2, y2);

cout << "Rhombus square: " << r.getSquare() << **endl**;

cout << "Rectangle square: " << rect.getSquare() << **endl**;

**system**("pause");

**return** 0;

}

**Quadrilateral::Quadrilateral**(vector<**double**> &x, vector<**double**> &y) : Figure(4) {

**this**->x = x;

**this**->y = y;

}

**double** **Rhombus::getSquare**() {

**return** **sqrt**((x[0] - x[2]) \* (x[0] - x[2]) + (y[0] - y[2]) \* (y[0] - y[2])) \*

**sqrt**((x[1] - x[3]) \* (x[1] - x[3]) + (y[1] - y[3]) \* (y[1] - y[3])) / 2;

}

**double** **Rectangle::getSquare**() {

**return** **sqrt**((x[0] - x[1]) \* (x[0] - x[1]) + (y[0] - y[1]) \* (y[0] - y[1])) \*

**sqrt**((x[2] - x[3]) \* (x[2] - x[3]) + (y[2] - y[3]) \* (y[2] - y[3]));

}

1. Программа №3

**#include** <iostream>

**using** **namespace** std;

**class** Transport { // Базовый абстрактный класс "Транспорт"

**protected**:

**double** maxVelocity; // Максимальная скорость

**public**:

**Transport**(**double** maxVelocity) { **this**->maxVelocity = maxVelocity; }

**double** **getMaxVelocity** () { **return** maxVelocity; }

**virtual** **void** **showAdvantages**() = 0; // виртуальный метод, выводящий достоинства транспорта

};

**class** Car : **public** Transport{ // Класс "Автомобиль"

**public**:

**Car**(**int** maxVelocity) : Transport(maxVelocity) {};

**void** **showAdvantages**(); // Определённый метод, выводящий достоинства автомобиля

};

**class** Bus : **public** Transport { // Класс "Автобус"

**public**:

**Bus**(**int** maxVelocity) : Transport(maxVelocity) {};

**void** **showAdvantages**(); // Определённый метод, выводящий достоинства автобуса

};

**class** Bicycle : **public** Transport { // Класс "Велосипед"

**public**:

**Bicycle**(**int** maxVelocity) : Transport(maxVelocity) {};

**void** **showAdvantages**(); // Определённый метод, выводящий достоинства велосипеда

};

**int** **main**() {

**double** a, b, c;

cout << "Enter max speed of a car: ";

cin >> a;

cout << "Enter max speed of a bus: ";

cin >> b;

cout << "Enter max speed of a bicycle: ";

cin >> c;

Car car(a);

Bus bus(b);

Bicycle bicycle(c);

car.showAdvantages();

bus.showAdvantages();

bicycle.showAdvantages();

**system**("pause");

**return** 0;

}

**void** **Car::showAdvantages**() {

cout << "Car:" << **endl**;

cout << "Can park in the yard" << **endl**;

cout << "Max speed: " << maxVelocity << **endl**;

}

**void** **Bus::showAdvantages**() {

cout << "Bus:" << **endl**;

cout << "Has a big size" << **endl**;

cout << "Max speed: " << maxVelocity << **endl**;

}

**void** **Bicycle::showAdvantages**() {

cout << "Bicycle:" << **endl**;

cout << "Can ride everywhere" << **endl**;

cout << "Max speed: " << maxVelocity << **endl**;

}

1. Программа №4

**#include** <iostream>

**using** **namespace** std;

**class** Screen { // Базовый абстрактный класс "Экран"

**protected**:

**double** screenSize;

**public**:

**Screen**(**double** size) { **this**->screenSize = size; }

**virtual** **void** **changeScreen**(**double** size) = 0; // Виртуальный метод заены экрана

};

**class** Keyboard { // Базовый абстрактный класс "Клавиатура"

**protected**:

**int** buttons;

**public**:

**Keyboard**(**int** buttons) { **this**->buttons = buttons; }

**virtual** **void** **changeKeyboard**(**int** buttons) = 0; // Виртуальный метод замены клавиатуры

};

**class** Computer : **public** Screen, Keyboard { // Класс "Стационарный компьютер"

**public**:

**Computer**(**double** screenSize, **int** buttons) : Screen(screenSize), Keyboard(buttons) {};

**void** **changeScreen**(**double** size);

**void** **changeKeyboard**(**int** buttons);

**~Computer**() {}

};

**class** Notebook : **public** Screen, Keyboard { // Класс "Ноутбук"

**public**:

**Notebook**(**double** screenSize, **int** buttons) : Screen(screenSize), Keyboard(buttons) {};

**void** **changeScreen**(**double** size);

**void** **changeKeyboard**(**int** buttons);

**~Notebook**() {}

};

**class** Phone : **public** Screen, Keyboard { // Класс "Телефон"

**public**:

**Phone**(**double** screenSize, **int** buttons) : Screen(screenSize), Keyboard(buttons) {};

**void** **changeScreen**(**double** size);

**void** **changeKeyboard**(**int** buttons);

**~Phone**() {}

};

**int** **main**() {

Computer c(19, 80);

Notebook n(15, 80);

Phone p(5, 40);

c.changeKeyboard(120);

c.changeScreen(21);

n.changeKeyboard(120);

n.changeScreen(15);

p.changeKeyboard(40);

p.changeScreen(6);

**system**("pause");

**return** 0;

}

**void** **Computer::changeScreen**(**double** size) {

cout << "Screen with " << screenSize <<

" size has been changed to " << size << " size" << **endl**;

screenSize = size;

}

**void** **Computer::changeKeyboard**(**int** buttons) {

cout << "Keyboard with " << **this**->buttons << " buttons " <<

"has been changed to keyboard with " << buttons << " buttons" << **endl**;

**this**->buttons = buttons;

}

**void** **Notebook::changeScreen**(**double** size) {

**if** (screenSize == size) {

cout << "Screen has been changed" << **endl**;

} **else** {

cout << "Notebook screen can be replaced only by the screen of the same size" << **endl**;

}

}

**void** **Notebook::changeKeyboard**(**int** buttons) {

**if** (**this**->buttons == buttons) {

cout << "Keyboard has been changed" << **endl**;

} **else** {

cout << "Notebook keyboard can be replaced only by the same keyboard" << **endl**;

}

}

**void** **Phone::changeScreen**(**double** size) {

**if** (screenSize == size) {

cout << "Screen has been changed" << **endl**;

} **else** {

cout << "Phone screen can be replaced only by the screen of the same size" << **endl**;

}

}

**void** **Phone::changeKeyboard**(**int** buttons) {

cout << "Phone keyboard can not be changed" << **endl**;

}

**Вывод**

Все задачи были выполнены. В ходе выполнения работы были более плотно изучены множественное наследование и абстрактные классы. Для достижения поставленной цели были использованы различные STL контейнеры и библиотека cmath. Результаты работы программ представлены на изображениях (рис. 21-24). Исходный кодвыполненных програм также доступе но ссылка на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr7.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr7.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Pr7.3>
4. <https://github.com/IvLaptev/OOP/tree/master/Pr7.4>

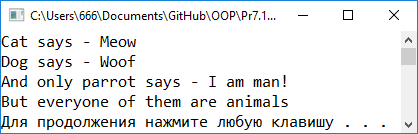


Рис. 21 Результат работы программы №1

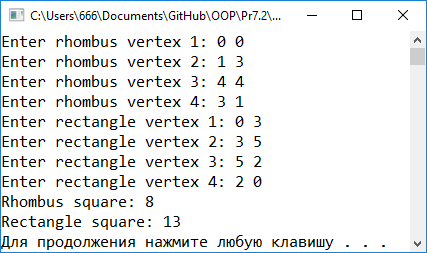


Рис. 22 Результат работы программы №2

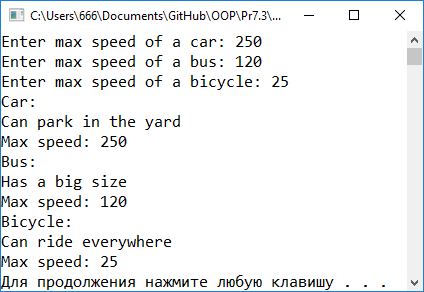


Рис. 23 Результат работы программы №3

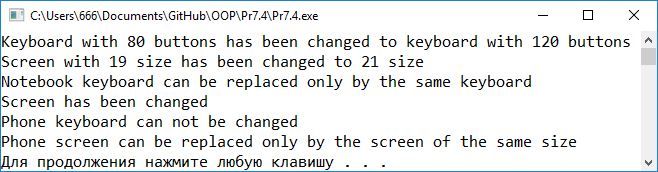


Рис. 24 Результат работы программы №4

## Практическая работа №8

**Работа с файлами в языке С++**

**Цель практической работы**

Целью данной практической работы является приобретение практических навыков по работе с файлами на языке программирования языке C++.

**Задачи**

1. Реализуйте программу, считывающую текст из файла и выводящую каждое слово с новой строки.
2. Реализуйте программу, считывающую текст с клавиатуры и записывающую его в файл.

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <fstream>

**#include** <string>

**#include** <Windows.h>

**using** **namespace** std;

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

ifstream fin("a.txt"); // Открытие файла на считывание данных из него

**while** (!fin.eof()) {

string s;

fin >> s; // Считывание данных из файла

cout << s << **endl**;

}

fin.close(); // Закрытие файла

**system**("pause");

**return** 0;

}

1. Программа №2

**#include** <iostream>

**#include** <fstream>

**#include** <Windows.h>

**using** **namespace** std;

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

ofstream fout("a.txt"); // Открытие файла для записи

**char** c = 0;

**while** (c != '\n') {

cin.get(c);

fout << c; // Посимвольный ввод текста в файл

}

fout.close(); // Закрытие файла

**system**("pause");

**return** 0;

}

**Вывод**

Все задачи были выполнены успешно. В ходе работы была исследована работа с файлами. Для достижения результата использовалась библиотек fstream, позволяющая производить работу с файлами. Результаты работ программ приведены на рисунках (рис. 25, 26). Также исходный код програм доступенн по ссылкам на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Pr8.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Pr8.2>

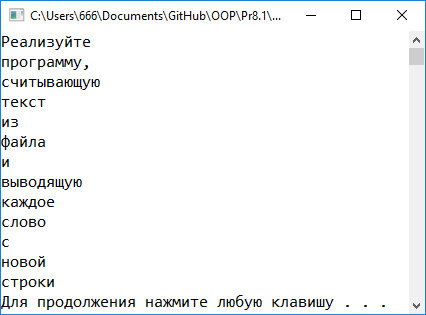


Рис. 25 Результат работы программы №1

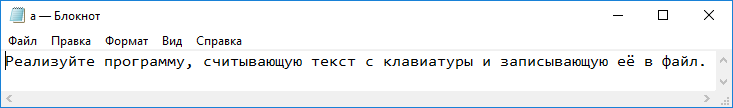
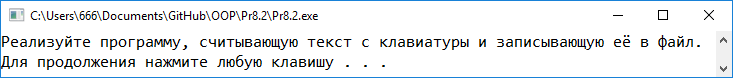


Рис. 26 Результат работы программы №2

## Лабораторная работа №1

**Создание многофайловых проектов**

**Цель практической работы**

Целью данной лабораторной работы освоить на практике создание многофайловых проектов а языке Си/Си++, познакомиться с директивами условной компиляции.

**Задачи**

1. Написать программу – калькулятор комплексных чисел.
2. На основе задания 1 написать программу, считывающую комплексные числа из файла complex.txt и находящуюю в них число с наибольшим модулем.

**Ход работы**

Файл заголовков Complex.h:

**#include** <iostream>

**#include** <cmath>

**#ifndef** COMPLEX\_H\_

**#define** COMPLEX\_H\_

**class** Complex {

**double** re; // Действительная часть числа

**double** im; // Мнимая часть числа

**public**:

**Complex**(); // Создаёт число равное 0

**Complex**(**double**, **double**); // Создает число с известными мнимой и действительной частями

**void** **setNumber**(**double**, **double**); // Задаёт действительную и мнимую часть

**double** **getRe**(); // Возвращает действительную часть числа

**double** **getIm**(); // Возвращает мнимую часть числа

**double** **countModule**(); // Возвращает модуль комплексного числа

Complex& **operator=** (**const** Complex&); // Перегруженный оператор присвоения

**friend** **const** Complex **operator+** (**const** Complex&, **const** Complex&); // Сложение

**const** Complex **operator-** (**const** Complex&, **const** Complex&); // Вычитание

**friend** **const** Complex **operator\*** (**const** Complex&, **const** Complex&); // Умножение

**friend** **const** Complex **operator/** (**const** Complex&, **const** Complex&); // Деление

**friend** std::ostream& **operator<<** (std::ostream&, **const** Complex&); // Вывод

**~Complex**(); // Деструктор

};

**#endif** /\* COMPLEX\_H\_ \*/

Файл реализации класса Complex.cpp:

**#include** "Complex.h"

**Complex::Complex**() { setNumber(0, 0); }

**Complex::Complex**(**double** re, **double** im) { setNumber(re, im); }

**void** **Complex::setNumber**(**double** re, **double** im) {

**this**->im = im;

**this**->re = re;

}

**double** **Complex::getRe**() { **return** re; }

**double** **Complex::getIm**() { **return** im; }

**double** **Complex::countModule**() { **return** **sqrt**(re \* re + im \* im); }

Complex& **Complex::operator=** (**const** Complex& c) {

re = c.re;

im = c.im;

**return** \***this**;

}

**const** Complex **operator+** (**const** Complex& c1, **const** Complex& c2) {

**return** Complex(c1.re + c2.re, c1.im + c2.im);

}

**const** Complex **operator-** (**const** Complex& c1, **const** Complex& c2) {

**return** Complex(c1.re - c2.re, c1.im - c2.im);

}

**const** Complex **operator\*** (**const** Complex& c1, **const** Complex& c2) {

**return** Complex(c1.re \* c2.re - c1.im \* c2.im, c1.im \* c2.re + c1.re \* c2.im);

}

**const** Complex **operator/** (**const** Complex& c1, **const** Complex& c2) {

**return** Complex((c1.re \* c2.re + c1.im \* c2.im) / (c2.re \* c2.re + c2.im \* c2.im),

(c1.im \* c2.re - c1.re \* c2.im) / (c2.re \* c2.re + c2.im \* c2.im));

}

std::ostream& **operator<<** (std::ostream& out, **const** Complex& c) {

out << c.re;

**if** (c.im < 0) {

out << c.im << "i";

} **else** {

out << "+" << c.im << "i";

}

**return** out;

}

**Complex::~Complex**() {}

1. Программа №1

**#include** <iostream>

**#include** "Complex.h"

**using** **namespace** std;

**int** **main**() {

**double** re, im;

cout << "Enter Re(a): ";

cin >> re;

cout << "Enter Im(a): ";

cin >> im;

Complex c1(re, im);

cout << "Enter Re(b): ";

cin >> re;

cout << "Enter Im(b): ";

cin >> im;

Complex c2(re, im);

cout << "a + b = " << c1 + c2 << **endl**;

cout << "a - b = " << c1 - c2 << **endl**;

cout << "a \* b = " << c1 \* c2 << **endl**;

cout << "a / b = " << c1 / c2 << **endl**;

**system**("pause");

**return** 0;

}

1. Программа №2

**#include** <iostream>

**#include** <fstream>

**#include** "Complex.h"

**using** **namespace** std;

**const** Complex **readNumber**(ifstream&); // Считывает одно комплексное число из файла

**int** **main**() {

ifstream f("complex.txt");

**int** maxModule = -1;

Complex max;

**int** n;

f >> n;

**for** (**int** i = 0; i < n; i++) {

Complex c;

c = readNumber(f);

**if** (c.countModule() > maxModule) {

max = c;

maxModule = c.countModule();

}

}

cout << max << " has max module - " << max.countModule() << **endl**;

f.close();

**system**("pause");

**return** 0;

}

**const** Complex **readNumber**(ifstream& f) {

**double** re, im;

f >> re >> im;

f.get();

**return** Complex(re, im);

}

**Вывод**

Все поставленные задачи были выполнены. Для достижения требуемого результата был создан класс, разбитый на файлы с реализацией и объявлением. Также были использованы некоторые дерективы препрцессора для защиты от многократного подключения файлов и перегружены некоторые операторы. Результаты работы программ приведены на рисунках (рис. 27, 28). Исходный код также доступен по ссылкам на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Lab1.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Lab1.2>

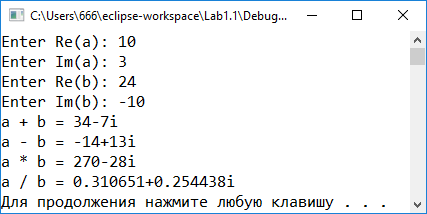


Рис. 27 Результат работы программы №1

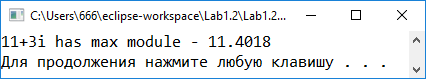


Рис. 28 Результат работы программы №2

## Лабораторная работа №2

**Указатели на функции**

**Цель практической работы**

Целью данной лабораторной работы освоить на практике вызов функции с использованием указателей.

**Задачи**

1. Напишите программу, которая вызывает различные виды функции в зависимости от заданного условия.
2. Напишите программу с использованием некой универсальной функции, которая в качестве возвращаемого значения возвращает указатель на функцию, зависящий от некоторого условия.v

**Ход работы**

1. Программа №1

**#include** <iostream>

**using** **namespace** std;

**double** **taxOverMil**(**double** value); // Вычисляет налог на сумму больше миллиона

**double** **taxLessMil**(**double** value); // Вычисляет налог на сумму до миллиона

**double** (\*getTax)(**double** value); // Выбирает, как вычислять налог

**int** **main**() {

**double** income;

cout << "Enter your income: ";

cin >> income;

**if** (income > 1000000) {

getTax = taxOverMil;

} **else** {

getTax = taxLessMil;

}

cout << "Your tax: " << getTax(income) << **endl**;

**system**("pause");

**return** 0;

}

**double** **taxOverMil**(**double** value) {

**return** value \* 0.2;

}

**double** **taxLessMil**(**double** value) {

**return** value \* 0.1;

}

1. Программа №2

**#include** <iostream>

**#include** <ctime>

**using** **namespace** std;

**void** **invert**(**int** \*arr, **int** size); // Инвертирует массив

**void** **sortMinToMax**(**int** \*arr, **int** size); // Сортирует массив в порядке неубывания

**void** **sortMaxToMin**(**int** \*arr, **int** size); // Сортирует массив в порядке невозрастания

//Возвращает функцию, которую надо выполнить

**void** (\***f**(**int** \*arr, **int** size))(**int** \*arr, **int** size);

**int** **main**() {

**srand**(**time**(0));

**int** n = 10;

**int** \*arr = **new** **int**[n];

cout << "Input array: ";

**for** (**int** i = 0; i < n; i++) {

arr[i] = **rand**() % 100;

cout << arr[i] << " ";

}

f(arr, n)(arr, n);

cout << **endl** << "Output array: ";

**for** (**int** i = 0; i < n; i++) {

cout << arr[i] << " ";

}

cout << **endl**;

**system**("pause");

**return** 0;

}

**void** **invert**(**int** \*arr, **int** size) {

**for** (**int** i = 0; i < size/2; i++) {

**int** c = arr[i];

arr[i] = arr[size - 1 - i];

arr[size - 1 - i] = c;

}

}

**void** **sortMinToMax**(**int** \*arr, **int** size) {

**for** (**int** i = 0; i < size - 1; i++) {

**for** (**int** j = i + 1; j < size; j++) {

**if** (arr[j] < arr[i]) {

**int** c = arr[i];

arr[i] = arr[j];

arr[j] = c;

}

}

}

}

**void** **sortMaxToMin**(**int** \*arr, **int** size) {

**for** (**int** i = 0; i < size - 1; i++) {

**for** (**int** j = i + 1; j < size; j++) {

**if** (arr[j] > arr[i]) {

**int** c = arr[i];

arr[i] = arr[j];

arr[j] = c;

}

}

}

}

**void** (\***f**(**int** \*arr, **int** size))(**int** \*arr, **int** size) {

**int** sum = 0;

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

sum += arr[i];

}

**if** (sum == arr[0]) {

**return** invert;

} **else** {

**if** (sum < arr[0]) {

**return** sortMaxToMin;

} **else** {

**return** sortMinToMax;

}

}

}

**Вывод**

Все поставленные задачи были выполнены. Для достижения поставленных целей были использованы указатели на функции с различными возвращаемыми значениями. Результаты работы программ можно увидеть на рисунках (рис. 29, 30). Исходный код программ также доступен по ссылкам на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Lab2.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Lab2.2>

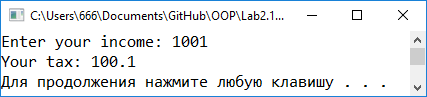


Рис. 29 Результат работы программы №1

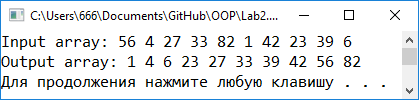


Рис. 30 Результат работы программы №2

## Лабораторная работа №3

**Классы**

**Цель практической работы**

Целью данной лабораторной работы является знакомство с классами в языке С++.

**Задачи**

1. Определить класс Child, который содержит такие поля (члены класса): закрытые — имя ребенка, фамилию и возраст, публичные — методы ввода данных и отображения их на экран. Объявить два объекта класса, внести данные и показать их.
2. Создать класс Tiles (кафель), который будет содержать поля с открытым доступом: brand, size\_h, size\_w, price и метод класса getData(). В главной функции объявить пару объектов класса и внести данные в поля. Затем отобразить их, вызвав метод getData().
3. Создать класс Complex, в котором реализовано комплексное число. В данном классе должны присутствовать методы, позволяющие рассчтать и вывеси модуль и аргументы данного числа.
4. Реализовать класс Vector, позволяющий хранить в себе математический вектор. В классе должно присутствовать метод позволяющей получить модуль вектора и методы, позволяющие складывать и вычитать разные векторы.

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <string>

**#include** <Windows.h>

**using** **namespace** std;

**class** Child {

string name;

string surname;

**int** age;

**public**:

**Child**(string&, string&, **int**);

**void** **setName**(string& name) { **this**->name = name; } // Методы ввода данных

**void** **setSurname**(string& surname) { **this**->surname = surname; }

**void** **setAge**(**int** age) { **this**->age = age; }

**void** **display**(); // Вывод данных на экран

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string name, surname;

**int** age;

cout << "Enter name of the first child: ";

cin >> name;

cout << "Enter surname of the first child: ";

cin >> surname;

cout << "Enter age of the first child: ";

cin >> age;

Child c1(name, surname, age);

cout << "Enter name of the second child: ";

cin >> name;

cout << "Enter surname of the second child: ";

cin >> surname;

cout << "Enter age of the second child: ";

cin >> age;

Child c2(name, surname, age);

c1.display();

c2.display();

**system**("pause");

**return** 0;

}

**Child::Child**(string& name, string& surname, **int** age) {

setName(name);

setSurname(surname);

setAge(age);

}

**void** **Child::display**() {

cout << "Name: " << name << **endl**;

cout << "Surname: " << surname << **endl**;

cout << "Age: " << age << **endl**;

}

1. Программы №2

**#include** <iostream>

**#include** <Windows.h>

**#include** <string>

**using** **namespace** std;

**class** Tiles {

**public**:

string brand;

**double** size\_w;

**double** size\_h;

**double** price;

**void** **getData**(); // Вывод данных на экран

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

Tiles t1;

cout << "Enter brand of the first tiles: ";

cin >> t1.brand;

cout << "Enter width of the first tiles: ";

cin >> t1.size\_w;

cout << "Enter height of the first tiles: ";

cin >> t1.size\_h;

cout << "Enter price of the first tiles: ";

cin >> t1.price;

Tiles t2;

cout << "Enter brand of the first tiles: ";

cin >> t2.brand;

cout << "Enter width of the first tiles: ";

cin >> t2.size\_w;

cout << "Enter height of the first tiles: ";

cin >> t2.size\_h;

cout << "Enter price of the first tiles: ";

cin >> t2.price;

cout << "The first tiles" << **endl**;

t1.getData();

cout <<**endl** << "The second tiles" << **endl**;

t2.getData();

**system**("pause");

**return** 0;

}

**void** **Tiles::getData**() {

cout << "Brand: " << brand << **endl**;

cout << "Width: " << size\_w << **endl**;

cout << "Height: " << size\_h << **endl**;

cout << "Price: " << price << **endl**;

}

1. Программа №3

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**class** Complex {

**double** re;

**double** im;

**public**:

**Complex**(**double** re, **double** im) { setNumber(re, im); }

**void** **setNumber**(**double**, **double**); // Получает мнимую и действительную части числа

**double** **getModule**() { **return** **sqrt**(re \* re + im \* im); } // Возвращает модуль

**void** **displayData**(); // Выводит информацию о числе на экран

};

**int** **main**() {

**double** re, im;

cout << "Enter Re(a): ";

cin >> re;

cout << "Enter Im(a): ";

cin >> im;

Complex c1(re, im);

cout << "Enter Re(b): ";

cin >> re;

cout << "Enter Im(b): ";

cin >> im;

Complex c2(re, im);

c1.displayData();

c2.displayData();

**system**("pause");

**return** 0;

}

**void** **Complex::setNumber**(**double** re, **double** im) {

**this**->re = re;

**this**->im = im;

}

**void** **Complex::displayData**() {

cout << "Number: " << re;

**if** (im < 0) {

cout << im << "i" << **endl**;

} **else** {

cout << "+" << im << "i" << **endl**;

}

cout << "Module: " << getModule() << **endl**;

}

1. Программа №4

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**class** Vector {

**double** x;

**double** y;

**double** z;

**public**:

**Vector**(**double** x, **double** y, **double** z) { setVector(x, y, z); }

**void** **setVector**(**double** x, **double** y, **double** z); // Получает координаты вектора

**double** **getModule**() { **return** **sqrt**(x \* x + y \* y + z \* z); } // Возвращает модуль

**void** **printData**(); // Выводит данные на экран

Vector **operator=** (**const** Vector& v) { **return** Vector(v.x, v.y, v.z); }

**friend** Vector **operator+** (**const** Vector& v1, **const** Vector& v2) {

**return** Vector(v1.x + v2.x, v1.y + v2.y, v1.z + v2.z);

}

**friend** Vector **operator-** (**const** Vector& v1, **const** Vector& v2) {

**return** Vector(v1.x - v2.x, v1.y - v2.y, v1.z - v2.z);

}

};

**int** **main**() {

**double** x, y, z;

cout << "Set coordinates of the vector:" << **endl** << "x = ";

cin >> x;

cout << "y = ";

cin >> y;

cout << "z = ";

cin >> z;

Vector v1(x, y, z);

Vector v2 = v1 + v1 - v1;

v2.printData();

**system**("pause");

**return** 0;

}

**void** **Vector::setVector**(**double** x, **double** y, **double** z) {

**this**->x = x;

**this**->y = y;

**this**->z = z;

}

**void** **Vector::printData**() {

cout << "Coordinates: (" << x << ", " << y << ", " << z << ")" << **endl**;

cout << "Module: " << getModule() << **endl**;

}

**Вывод**

Все поставленные задачи были достигнуты. Для получения требуемого результата былииспользованы конструкции класса и перегрузка некоторых операторов. Результаты работы программ изображены на рисунках (рис. 31-34). Исходные коды программ также доступны по ссылкам на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Lab3.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Lab3.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Lab3.3>
4. <https://github.com/IvLaptev/OOP/tree/master/Lab3.4>

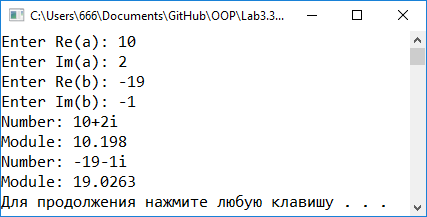


Рис. 31 Результат работы программы №1

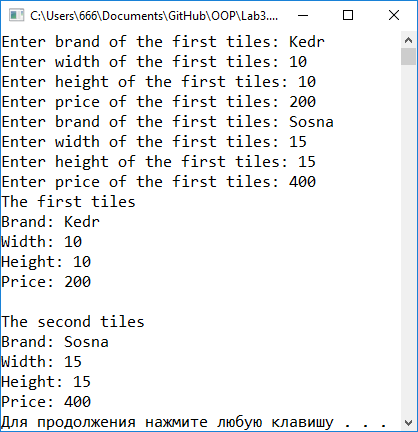


Рис. 32 Результат работы программы №2

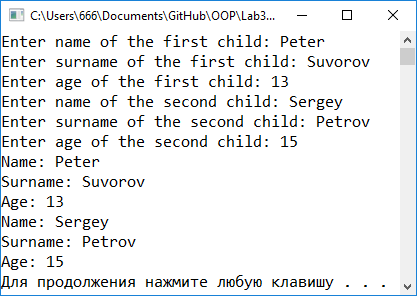


Рис. 33 Результат работы программы №3

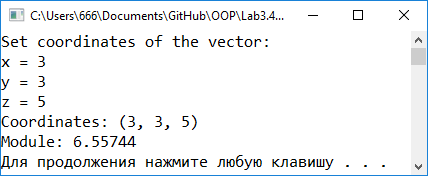


Рис. 34 Результат работы программы №4

## Лабораторная работа №4

**Конструкторы и деструкторы**

**Цель практической работы**

Целью данной лабораторной работы является знакомство с конструкторами и деструкторами в языке С++.

**Задачи**

Из прошлой лабораторной работы дополнить всем видами конструкторов и деструкторами классы:

1. Complex
2. Vector
3. Tiles
4. Child

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <string>

**#include** <Windows.h>

**using** **namespace** std;

**class** Child {

string name;

string surname;

**int** age;

**public**:

**Child**(string&, string&, **int**); // Конструкторы класса

**Child**(**const** Child&);

Child **operator=** (**const** Child&);

**void** **setName**(string& name) { **this**->name = name; } // Методы ввода данных

**void** **setSurname**(string& surname) { **this**->surname = surname; }

**void** **setAge**(**int** age) { **this**->age = age; }

**void** **display**(); // Выводит данные на экран

**~Child**() {}

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string name, surname;

**int** age;

cout << "Enter name of the first child: ";

cin >> name;

cout << "Enter surname of the first child: ";

cin >> surname;

cout << "Enter age of the first child: ";

cin >> age;

Child c1(name, surname, age);

Child c2(c1);

Child c3 = c2;

c1.display();

c2.display();

c3.display();

**system**("pause");

**return** 0;

}

Child **Child::operator=** (**const** Child& c) {

**this**->name = c.name;

**this**->age = c.age;

**this**->surname = c.surname;

**return** \***this**;

}

**Child::Child**(**const** Child& c) {

name = c.name;

surname = c.surname;

age = c.age;

}

**Child::Child**(string& name, string& surname, **int** age) {

setName(name);

setSurname(surname);

setAge(age);

}

**void** **Child::display**() {

cout << "Name: " << name << **endl**;

cout << "Surname: " << surname << **endl**;

cout << "Age: " << age << **endl**;

}

1. Программа №2

**#include** <iostream>

**#include** <Windows.h>

**#include** <string>

**using** **namespace** std;

**class** Tiles {

**public**:

string brand;

**double** size\_w;

**double** size\_h;

**double** price;

**Tiles**(string& brand, **double** size\_w, **double** size\_h, **double** price) :

brand(brand), size\_w(size\_w), size\_h(size\_h), price(price) {}

**Tiles**(**const** Tiles&);

Tiles **operator=** (**const** Tiles&);

**void** **getData**();

**~Tiles**() {}

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string brand;

**double** size\_w, size\_h, price;

cout << "Enter brand of the first tiles: ";

cin >> brand;

cout << "Enter width of the first tiles: ";

cin >> size\_w;

cout << "Enter height of the first tiles: ";

cin >> size\_h;

cout << "Enter price of the first tiles: ";

cin >> price;

Tiles t1(brand, size\_w, size\_h, price);

Tiles t2(t1);

Tiles t3 = t1;

cout << "The first tiles" << **endl**;

t1.getData();

cout <<**endl** << "The second tiles" << **endl**;

t2.getData();

cout <<**endl** << "The third tiles" << **endl**;

t3.getData();

**system**("pause");

**return** 0;

}

**Tiles::Tiles**(**const** Tiles& t) {

**this**->brand = t.brand;

**this**->price = t.price;

**this**->size\_h = t.size\_h;

**this**->size\_w = t.size\_w;

}

Tiles **Tiles::operator=** (**const** Tiles& t) {

**this**->brand = t.brand;

**this**->price = t.price;

**this**->size\_h = t.size\_h;

**this**->size\_w = t.size\_w;

**return** \***this**;

}

**void** **Tiles::getData**() {

cout << "Brand: " << brand << **endl**;

cout << "Width: " << size\_w << **endl**;

cout << "Height: " << size\_h << **endl**;

cout << "Price: " << price << **endl**;

}

1. Программа №3

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**class** Complex {

**double** re;

**double** im;

**public**:

**Complex**(**double** re, **double** im) : re(re), im(im) {}

**Complex**(**const** Complex& c);

Complex **operator=** (**const** Complex& c);

**void** **setNumber**(**double** re, **double** im);

**double** **getModule**();

**void** **displayData**();

**~Complex**() {}

};

**int** **main**() {

**double** re, im;

cout << "Enter Re(a): ";

cin >> re;

cout << "Enter Im(a): ";

cin >> im;

Complex c1(re, im);

Complex c2(c1);

Complex c3 = c1;

c1.displayData();

c2.displayData();

c3.displayData();

**system**("pause");

**return** 0;

}

**Complex::Complex**(**const** Complex& c) {

re = c.re;

im = c.re;

}

Complex **Complex::operator=** (**const** Complex& c) { **return** Complex(c.re, c.im); }

**void** **Complex::setNumber**(**double** re, **double** im) {

**this**->re = re;

**this**->im = im;

}

**double** **Complex::getModule**() { **return** **sqrt**(re \* re + im \* im); }

**void** **Complex::displayData**() {

cout << "Number: " << re;

**if** (im < 0) {

cout << im << "i" << **endl**;

} **else** {

cout << "+" << im << "i" << **endl**;

}

cout << "Module: " << getModule() << **endl**;

}

1. Программа №4

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**class** Vector {

**double** x;

**double** y;

**double** z;

**public**:

**Vector**(**double** x, **double** y, **double** z);

**Vector**(**const** Vector& v);

**void** **setVector**(**double** x, **double** y, **double** z);

**double** **getModule**();

**void** **printData**();

Vector **operator=** (**const** Vector& v);

**friend** Vector **operator+** (**const** Vector& v1, **const** Vector& v2);

**friend** Vector **operator-** (**const** Vector& v1, **const** Vector& v2);

};

**int** **main**() {

**double** x, y, z;

cout << "Set coordinates of the vector:" << **endl** << "x = ";

cin >> x;

cout << "y = ";

cin >> y;

cout << "z = ";

cin >> z;

Vector v1(x, y, z);

Vector v2 = v1;

Vector v3(v2);

v1.printData();

v2.printData();

v3.printData();

**system**("pause");

**return** 0;

}

**Vector::Vector**(**double** x, **double** y, **double** z) { setVector(x, y, z); }

**Vector::Vector**(**const** Vector& v) {

x = v.x;

y = v.y;

z = v.z;

}

**void** **Vector::setVector**(**double** x, **double** y, **double** z) {

**this**->x = x;

**this**->y = y;

**this**->z = z;

}

**double** **Vector::getModule**() { **return** **sqrt**(x \* x + y \* y + z \* z); }

**void** **Vector::printData**() {

cout << "Coordinates: (" << x << ", " << y << ", " << z << ")" << **endl**;

cout << "Module: " << getModule() << **endl**;

}

Vector **Vector::operator=** (**const** Vector& v) { **return** Vector(v.x, v.y, v.z); }

Vector **operator+** (**const** Vector& v1, **const** Vector& v2) {

**return** Vector(v1.x + v2.x, v1.y + v2.y, v1.z + v2.z);

}

Vector **operator-** (**const** Vector& v1, **const** Vector& v2) {

**return** Vector(v1.x - v2.x, v1.y - v2.y, v1.z - v2.z);

}

**Вывод**

Были выполнены все задачи. В процессе работы были более подробно изучены разные виды конструкторов и деструкторы. Для выполнения работы были использованы простые конструкторы, конструкторы копирования, а также перегруженный оператор присваивания. Результаты работы программ можно увидеть на рисунках (рис. 35-38). Исходный код программ также доступен по ссылкам на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Lab4.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Lab4.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Lab4.3>
4. <https://github.com/IvLaptev/OOP/tree/master/Lab4.4>

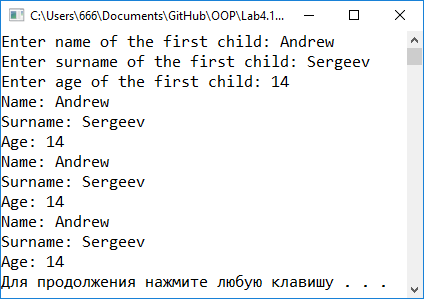


Рис. 35 Результат работы программы №1

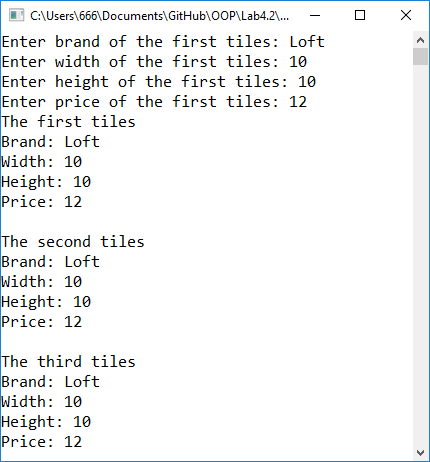


Рис. 36 Результат работы программы №2

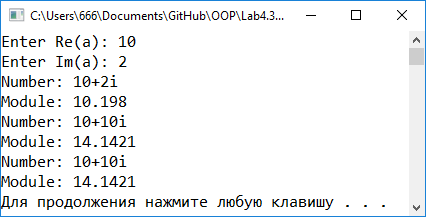


Рис. 37 Результат работы программы №3

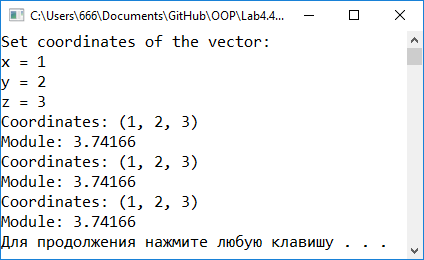


Рис. 38 Результат работы программы №4

## Лабораторная работа №5

**Перегрузка операторов в языке программирования C++**

**Цель практической работы**

Целью данной лабораторной работы является изучениеперегрузки операторов в языке С++ и использование перегруженныхоператоров на практике.

**Задачи**

1. Для класса Complex перегрузить операторы присваивания, инкремента, декремента, сравнения, ввода и вывода.
2. Для класса Vector перегрузить операторы присваивания, сравнения, ввода и вывода.

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**class** Complex {

**double** re;

**double** im;

**public**:

**Complex**() {}

**Complex**(**double** re, **double** im) : re(re), im(im) {}

Complex **operator=** (**const** Complex& c);

**friend** **const** Complex **operator++** (Complex& c, **int**);

**friend** **const** Complex **operator--** (Complex& c, **int**);

**friend** **const** **bool** **operator==** (**const** Complex& c1, **const** Complex& c2);

**friend** **const** ostream& **operator<<** (ostream& out, **const** Complex& c);

**friend** **const** istream& **operator>>** (istream& in, Complex& c);

**~Complex**() {}

};

**int** **main**() {

Complex c1;

cout << "Enter complex number in format x+yi: ";

cin >> c1;

Complex c2 = c1;

**if** (c1 == c2) {

c2++;

}

c1--;

cout << c1;

cout << **endl** << c2;

cout << **endl**;

**system**("pause");

**return** 0;

}

Complex **Complex::operator=** (**const** Complex& c) { **return** Complex(c.re, c.im); }

**const** Complex **operator++** (Complex& c, **int**) {

Complex oldC(c);

c.re++;

**return** oldC;

}

**const** Complex **operator--** (Complex& c, **int**) {

Complex oldC(c);

c.re--;

**return** oldC;

}

**const** **bool** **operator==** (**const** Complex& c1, **const** Complex& c2) {

**return** (c1.re == c2.re) && (c1.im == c2.im);

}

**const** ostream& **operator<<** (ostream& out, **const** Complex& c) {

out << c.re;

**if** (c.im < 0) {

out << c.im << "i";

} **else** {

out << "+" << c.im << "i";

}

**return** out;

}

**const** istream& **operator>>** (istream& in, Complex& c) {

in >> c.re >> c.im;

in.get();

**return** in;

}

1. Программа №2

**#include** <iostream>

**#include** <cmath>

**using** **namespace** std;

**class** Vector {

**double** x;

**double** y;

**double** z;

**public**:

**Vector**() : x(0), y(0), z(0) {}

**Vector**(**double** x, **double** y, **double** z) : x(x), y(y), z(z) {};

**Vector**(**const** Vector& v);

Vector **operator=** (**const** Vector& v);

**friend** Vector **operator+** (**const** Vector& v1, **const** Vector& v2);

**friend** Vector **operator-** (**const** Vector& v1, **const** Vector& v2);

**friend** **const** **bool** **operator==** (**const** Vector& v1, **const** Vector& v2);

**friend** **const** istream& **operator>>** (istream& in, Vector& v);

**friend** **const** ostream& **operator<<** (ostream& out, **const** Vector& v);

};

**int** **main**() {

cout << "Set coordinates of the vector:";

Vector v1;

cin >> v1;

Vector v2 = v1;

**if** (v1 == v2) {

cout << v1;

}

cout << **endl**;

**system**("pause");

**return** 0;

}

**Vector::Vector**(**const** Vector& v) {

x = v.x;

y = v.y;

z = v.z;

}

Vector **Vector::operator=** (**const** Vector& v){**return** Vector(v.x, v.y, v.z);}

Vector **operator+** (**const** Vector& v1, **const** Vector& v2) {

**return** Vector(v1.x + v2.x, v1.y + v2.y, v1.z + v2.z);

}

Vector **operator-** (**const** Vector& v1, **const** Vector& v2) {

**return** Vector(v1.x - v2.x, v1.y - v2.y, v1.z - v2.z);

}

**const** **bool** **operator==** (**const** Vector& v1, **const** Vector& v2) {

**return** (v1.x == v2.x) && (v1.y == v2.y) && (v1.z == v2.z);

}

**const** istream& **operator>>** (istream& in, Vector& v) {

in >> v.x >> v.y >> v.z;

**return** in;

}

**const** ostream& **operator<<** (ostream& out, **const** Vector& v) {

out << v.x << " " << v.y << " " << v.z;

**return** out;

}

**Вывод**

Все поставленные задачи были выполнены. В процессе выполнения работы был более плотно изучен механизм перегрузки. Для выполнения поставленных задач была использована перегрузка некоторых бинарных и унарных операторов. Результаты работы программ можно увидеть на рисунках (рис. 39, 40). Исходный код программ также доступен по ссылкам на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Lab5.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Lab5.2>

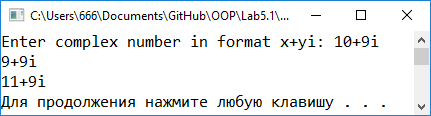


Рис. 39 Результат работы программы №1

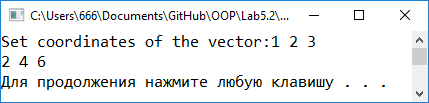


Рис. 40 Результат работы программы №2

## Лабораторная работа №6

**Наследование**

**Цель практической работы**

Целью данной лабораторной работы является изучение наследованная классов в языке С++.

**Задачи**

1. Создать класс «Староста», производный от класса «Студент». Новый класс должен содержать несколько дополнительных методов и полей.
2. Создать класс Alive и расширить его до Bird, Fish, Animal.
3. Создать класс Animal, и расширить его до Dog, Cat.

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <string>

**#include** <Windows.h>

**using** **namespace** std;

**class** Student { // базовый класс "Студент"

string name;

string surname;

**int** age;

**public**:

**Student**(string &name, string &surname, **int** age) : name(name), surname(surname), age(age) {};

string &**getName**() { **return** name; }

string &**getSurname**() { **return** surname; }

**int** **getAge**() { **return** age; }

};

**class** Headman : **public** Student { // Класс "Староста", производный от "Студент"

string phone;

string email;

**public**:

**Headman**(string &name, string &surname, **int** age, string &phone, string &email) :

Student(name, surname, age), phone(phone), email(email) {};

string &**getEmail**() { **return** email; }

string &**getPhone**() { **return** phone; }

**friend** **const** ostream& **operator<<** (ostream& out, **const** Headman& h);

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string name, surname, phone, email;

**int** age;

cout << "Enter name: ";

cin >> name;

cout << "Enter surname: ";

cin >> surname;

cout << "Enter age: ";

cin >> age;

cout << "Enter phone: ";

cin >> phone;

cout << "Enter email: ";

cin >> email;

Headman h(name, surname, age, phone, email);

cout << **endl** << "Information about headman:" << **endl**;

cout << "Name: " << h.getName() << **endl**;

cout << "Surname: " << h.getSurname() << **endl**;

cout << "Age: " << h.getAge() << **endl**;

cout << "Phone: " << h.getPhone() << **endl**;

cout << "Email: " << h.getEmail() << **endl**;

cout << "Crown phrase: " << h;

cout << **endl**;

**system**("pause");

**return** 0;

}

**const** ostream& **operator<<** (ostream& out, **const** Headman& h) {

out << "I am your headman!";

**return** out;

}

1. Программа №2

**#include** <iostream>

**#include** <string>

**#include** <Windows.h>

**using** **namespace** std;

**class** Alive { // Базовый класс

**protected**:

string name;

**int** age;

**public**:

**Alive**(string &name, **int** age) : name(name), age(age) {};

string &**getName**() { **return** name; }

**int** **getAge**() { **return** age; }

**virtual** **void** **move**() = 0;

};

**class** Animal : **public** Alive { // Производные классы

**public**:

**Animal**(string &name, **int** age) : Alive(name, age) {};

**void** **move**() { cout << name << " is running" << **endl**; }

};

**class** Fish : **public** Alive {

**public**:

**Fish**(string &name, **int** age) : Alive(name, age) {};

**void** **move**() { cout << name << " is swimming" << **endl**; }

};

**class** Bird : **public** Alive {

**public**:

**Bird**(string &name, **int** age) : Alive(name, age) {};

**void** **move**() { cout << name << " is flying"; }

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string name;

**int** age;

cout << "Fish" << **endl**;

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

Fish f(name, age);

cout << **endl** << "Animal" << **endl**;

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

Animal a(name, age);

cout << **endl** << "Bird" << **endl**;

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

Bird b(name, age);

cout << **endl** << "Your bird" << **endl** << "Name: " << b.getName() << **endl** << "Age: " <<

b.getAge() << **endl**;

b.move();

cout << **endl** << "Your fish" << **endl** << "Name: " << f.getName() << **endl** << "Age: " <<

f.getAge() << **endl**;

f.move();

cout << **endl** << "Your animal" << **endl** << "Name: " << a.getName() << **endl** << "Age: " <<

a.getAge() << **endl**;

a.move();

**system**("pause");

**return** 0;

}

1. Программа №3

**#include** <iostream>

**#include** <string>

**#include** <Windows.h>

**using** **namespace** std;

**class** Animal { // Базовый класс

**protected**:

string name;

**int** age;

**public**:

**Animal**(string &name, **int** age) : name(name), age(age) {};

string &**getName**() { **return** name; }

**int** **getAge**() { **return** age; }

**virtual** **void** **talk**() = 0;

};

**class** Dog : **public** Animal { // Производные классы

**public**:

**Dog**(string &name, **int** age) : Animal(name, age) {};

**void** **talk**() { cout << "woof" << **endl**; }

};

**class** Cat : **public** Animal {

**public**:

**Cat**(string &name, **int** age) : Animal(name, age) {};

**void** **talk**() { cout << "meow" << **endl**; }

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string name;

**int** age;

cout << "Cat" << **endl**;

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

Cat c(name, age);

cout << **endl** << "Dog" << **endl**;

cout << "Enter name: ";

cin >> name;

cout << "Enter age: ";

cin >> age;

Dog d(name, age);

cout << **endl** << "Your dog" << **endl** << "Name: " << d.getName() << **endl** << "Age: " <<

d.getAge() << **endl**;

d.talk();

cout << **endl** << "Your cat" << **endl** << "Name: " << c.getName() << **endl** << "Age: " <<

c.getAge() << **endl**;

c.talk();

**system**("pause");

**return** 0;

}

**Вывод**

Все поставленные задачи были выполнены. В процессе выполнения работы было изучено и использовано наследование. Результат работы программ изображены на рисунках (рис. 41-43). Исходный код программ также можно просмотреть на GitHub:

1. <https://github.com/IvLaptev/OOP/tree/master/Lab6.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Lab6.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Lab6.3>

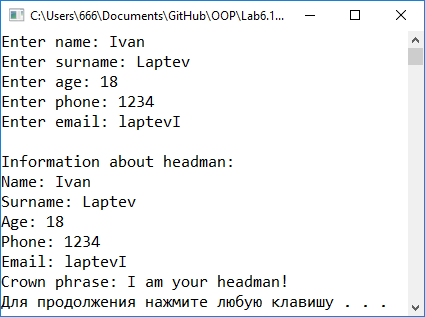


Рис. 41 Результат работы программы №1

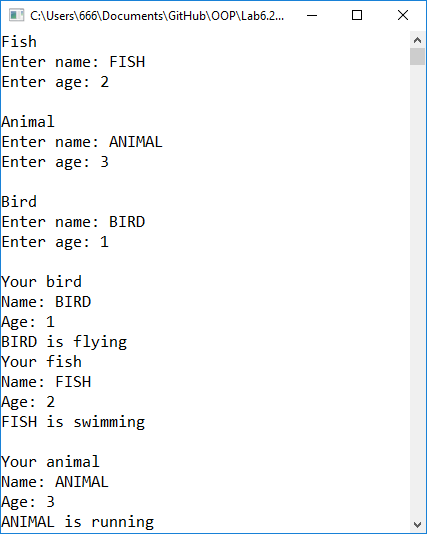


Рис. 42 Результат работы программы №2

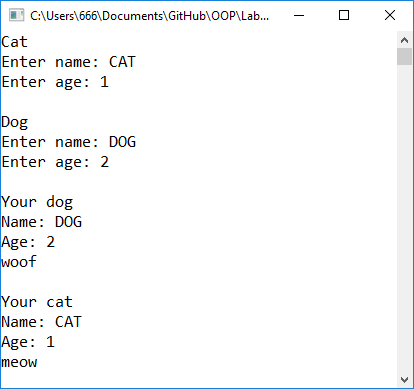


Рис. 43 Результат работы программы №3

## Лабораторная работа №7

**Создание абстрактных классов**

**Цель практической работы**

Целью данной лабораторной работы является изучение и создание абстрактных классов в языке С++.

**Задачи**

Реализовать систему из классов, изображённую на UML диаграмме.

**Ход работы**

**#include** <iostream>

**#include** <string>

**#include** <Windows.h>

**using** **namespace** std;

**class** Human {

**protected**:

string name;

string surname;

string midname;

**int** age;

**public**:

**Human**() : name(""), surname(""), midname(""), age(0) {};

**Human**(string surname, string name, string midname, **int** age) :

name(name), surname(surname), midname(midname), age(age) {};

**virtual** **void** **print**() = 0;

**~Human**() {}

};

**class** Student : **public** Human {

**bool** on\_lesson;

**public**:

**Student**() : Human(), on\_lesson(**false**) {};

**Student**(string surname, string name, string midname, **int** age, **bool** on\_lesson) :

Human(surname, name, midname, age), on\_lesson(on\_lesson) {};

**void** **print**();

**~Student**() {};

};

**class** Boss : **public** Human {

**int** number\_of\_workers;

**public**:

**Boss**() : Human(), number\_of\_workers(0) {};

**Boss**(string surname, string name, string midname, **int** age, **int** number\_of\_workers) :

Human(surname, name, midname, age), number\_of\_workers(number\_of\_workers) {};

**void** **print**();

**~Boss**() {};

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

string surname, name, midname;

**int** age, number\_of\_workers;

**bool** on\_lesson;

cout.setf(ios::*boolalpha*);

cout << "Student" << **endl**;

cout << "Enter surname: ";

cin >> surname;

cout << "Enter name: ";

cin >> name;

cout << "Enter midname: ";

cin >> midname;

cout << "Enter age: ";

cin >> age;

cout << "Is on lesson? (1 - yes, 0 - no): ";

cin >> on\_lesson;

Student s(surname, name, midname, age, on\_lesson);

cout << **endl** << "Boss" << **endl**;

cout << "Enter surname: ";

cin >> surname;

cout << "Enter name: ";

cin >> name;

cout << "Enter midname: ";

cin >> midname;

cout << "Enter age: ";

cin >> age;

cout << "Enter number of workers: ";

cin >> number\_of\_workers;

Boss b(surname, name, midname, age, number\_of\_workers);

s.print();

b.print();

**system**("pause");

**return** 0;

}

**void** **Student::print**() {

cout << "Name: " << surname << " " << name << " " << midname << **endl**;

cout << "Age: " << age << **endl**;

cout << "Is on lesson: " << on\_lesson << **endl**;

}

**void** **Boss::print**() {

cout << "Name: " << surname << " " << name << " " << midname << **endl**;

cout << "Age: " << age << **endl**;

cout << "Number of workers: " << number\_of\_workers << **endl**;

}

**Вывод**

Все поставленные задачи были выполнены. При выплнении работы были изучены основные элементы UML диаграмм. Для достижения поставленной цели были использованы механизмы наследования, переопределения функций и абстрактные классы. Результат работы программы изображён на рисунке (рис. 44). Код программы также доступен на GitHub по ссылке:

<https://github.com/IvLaptev/OOP/tree/master/Lab7>

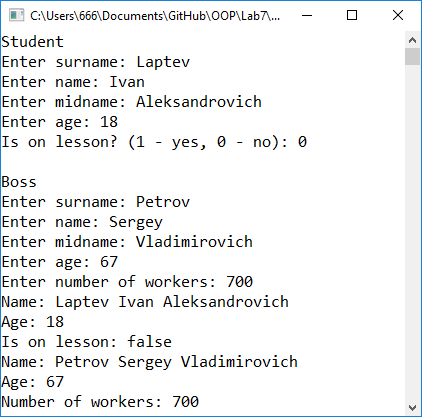


Рис. 44 Результат работы программы

## Лабораторная работа №8

**Бибилиотека STL**

**Цель практической работы**

Целью данной лабораторной работы является знакомство с библиотекой STL – стандартной библиотекой шаблонов - в языке С++, а также показать ее использование на примерах.

**Задачи**

1. Используйте шаблон vector для массива данных о студентах.
2. Используйте шаблон list для двусвязного списка данных класса Complex.
3. Используйте шаблон queue для очереди авто на мойке.

**Ход работы**

1. Программа №1

**#include** <iostream>

**#include** <string>

**#include** <vector>

**#include** <Windows.h>

**using** **namespace** std;

**class** Student {

string name;

string surname;

**int** age;

**public**:

**Student**(string &name, string &surname, **int** age) : name(name), surname(surname), age(age) {};

string &**getName**() { **return** name; }

string &**getSurname**() { **return** surname; }

**int** **getAge**() { **return** age; }

};

**int** **main**() {

**SetConsoleCP**(1251);

**SetConsoleOutputCP**(1251);

vector<Student> students;

**for** (**int** i = 0; i < 3; i++) {

string name, surname;

**int** age;

cout << "Student " << i+1 << **endl**;

cout << "Enter name: ";

cin >> name;

cout << "Enter surname: ";

cin >> surname;

cout << "Enter age: ";

cin >> age;

students.push\_back(Student(name, surname, age));

}

**for** (**auto** a : students) {

cout << **endl** << "Name: " << a.getName() << **endl**;

cout << "Surname: " << a.getSurname() << **endl**;

cout << "Age: " << a.getAge() << **endl**;

}

**system**("pause");

**return** 0;

}

1. Программа №2

Класс Complex был взят из лабораторной работы №1.

**#include** <iostream>

**#include** <cmath>

**#include** <list>

**using** **namespace** std;

**int** **main**() {

list<Complex> l;

**for** (**int** i = 0; i < 3; i++) {

**double** re, im;

cout << "Enter Re(x" << i+1 << "): ";

cin >> re;

cout << "Enter Im(x" << i+1 << "): ";

cin >> im;

l.push\_back(Complex(re, im));

}

**for** (**auto** a : l) {

cout << **endl**;

a.displayData();

}

**system**("pause");

**return** 0;

}

1. Программа №3

**#include** <iostream>

**#include** <string>

**#include** <queue>

**#include** <Windows.h>

**using** **namespace** std;

**class** Client {

string name;

string car;

**public**:

**Client**(string& name, string& car) : name(name), car(car) {};

**void** **getData**(); // Печатает информацию о клиенте

};

**int** **main**() {

queue<Client> q; // Создаётся очередь из объектов класса Client

**for** (**int** i = 0; i < 3; i++) {

string name, car;

cout << "Client " << i+1 << **endl**;

cout << "Enter name: ";

cin >> name;

cout << "Enter car model: ";

cin >> car;

q.push(Client(name, car)); // В конец очереди ставится новый объект

}

cout << "Queue" << **endl**;

**for** (**int** i = 1; !q.empty(); i++) {

cout << i;

q.front().getData(); // Берётся первый объект из очереди

q.pop(); // Удаляется первый объект очереди

}

**return** 0;

}

**void** **Client::getData**() {

cout << **endl** << "Name: " << name << **endl**;

cout << "Car: " << car << **endl**;

}

**Вывод**

Все поставленные задачи были выполнены. Была изучена библиотека STL. Для выполнения работы были использованы коллекции queue, string, list и vector. Результат работы программ изображён на рисунках (рис. 45-47). Исходный код программ так же доступен на GitHub по ссылкам:

1. <https://github.com/IvLaptev/OOP/tree/master/Lab8.1>
2. <https://github.com/IvLaptev/OOP/tree/master/Lab8.2>
3. <https://github.com/IvLaptev/OOP/tree/master/Lab8.3>

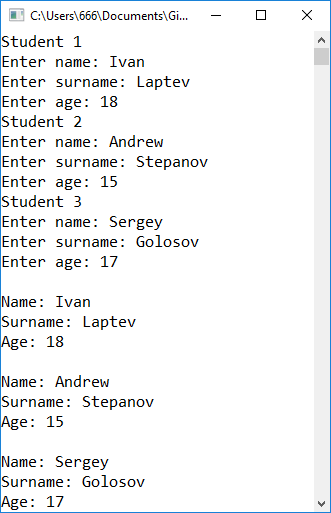


Рис. 45 Результат работы программы №1

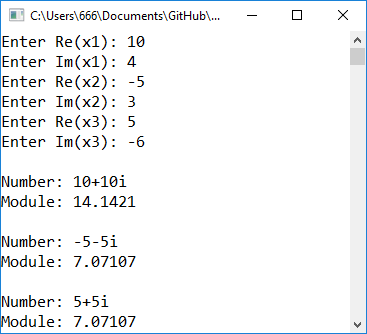


Рис. 46 Результат работы программы №2

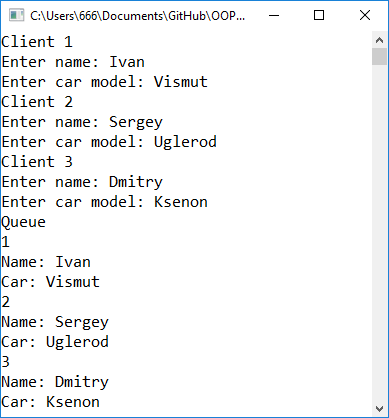


Рис. 47 Результат работы программы №3