|  |  |
| --- | --- |
|  | |
| 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>

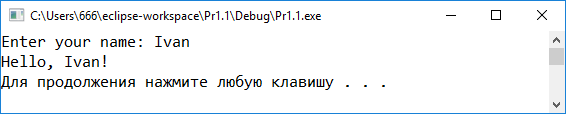


Рис. 1 Результат работы первой программы

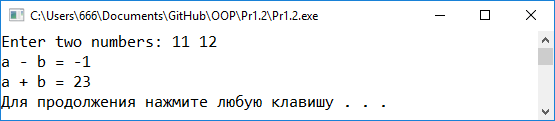


Рис. 2 Результат работы второй программы

## 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>

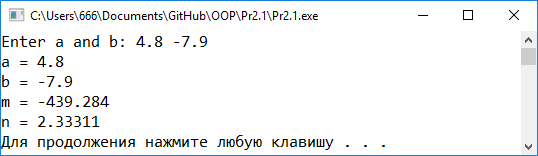


Рис. 3 Результат практической работы №2 вариант 1

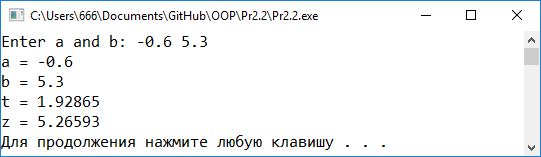


Рис. 4 Результат практической работы №2 вариант 2

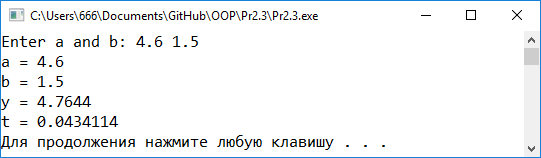


Рис. 5 Результат практической работы №2 вариант 3

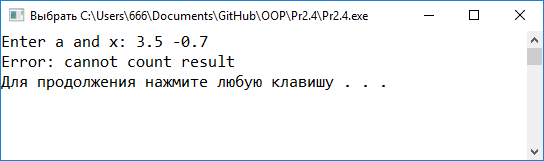


Рис. 6 Результат практической работы №2 вариант 4

## 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++.

**Задание**

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