Министерство науки и высшего образования Российской Федерации

Федерального государственного бюджетного образовательного учреждения высшего образования

"Сочинский государственный университет"

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮПроректор по учебной работе\_\_\_\_\_\_\_\_\_\_\_\_ А.В.Ревнивых«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. |

Регистрационный номер \_\_\_

Факультет «Инженерно-экологический»

Кафедра ИТ «Информационные технологии»

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

**по дисциплине**

**«Алгоритмизация и программирование»**

**Лабораторный практикум**

для направлений (уровень бакалавриата):
11.03.03 «Программирование в экономике», 27.03.04 «Управление в технических системах»

**Автор:**

**Драч В.E., к.т.н., доцент**

vladimir@drach.pro

г. Сочи, 2021 г.

Автор:



Драч Е.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Рецензент:

Руководитель направления по работе с ключевыми заказчиками

Siemens Industry Software, OOO

Product Lifecycle Management

Н.Г. Тутуев

Утверждена на заседании кафедры «Информационные технологии»

Протокол №3 от «\_\_27\_\_» \_\_октября\_\_ 2021 г.

Заведующий кафедрой

А.С.Копырин \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Декан факультета

Волков A.H. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Оглавление**

Оглавление

[Лабораторная работа 1. Знакомство со средой разработки 5](#_Toc89072910)

[**Предварительные замечания** 5](#_Toc89072911)

[**Файловая система** 5](#_Toc89072912)

[**Среда разработки** 5](#_Toc89072913)

[**Создание проекта** 5](#_Toc89072914)

[**Код первой программы** 5](#_Toc89072915)

[**Описание синтаксиса** 5](#_Toc89072916)

[**Компиляция и запуск** 6](#_Toc89072917)

[**Содержание отчёта** 6](#_Toc89072918)

[Лабораторная работа 2. Арифметика 7](#_Toc89072919)

[Типы данных 7](#_Toc89072920)

[Основные типы данных в C++ 7](#_Toc89072921)

[Объявление переменной 7](#_Toc89072922)

[Пример объявления переменных 7](#_Toc89072923)

[Простой калькулятор на C++ 7](#_Toc89072924)

[Разбор кода 8](#_Toc89072925)

[Задание 8](#_Toc89072926)

[**Содержание отчёта** 8](#_Toc89072927)

[Лабораторная работа 3. Условия 9](#_Toc89072928)

[Оператор if 9](#_Toc89072929)

[Пример конструкции ветвления 9](#_Toc89072930)

[Задание 11](#_Toc89072931)

[**Содержание отчёта** 11](#_Toc89072932)

[Лабораторная работа 4. Циклы 12](#_Toc89072933)

[Цикл for 12](#_Toc89072934)

[Описание синтаксиса 12](#_Toc89072935)

[Пример кода 12](#_Toc89072936)

[Цикл while 13](#_Toc89072937)

[Цикл do while 14](#_Toc89072938)

[**Задание** 14](#_Toc89072939)

[**Содержание отчёта** 14](#_Toc89072940)

[Лабораторная работа 5. Функции 16](#_Toc89072941)

[Пример построения функции 16](#_Toc89072942)

[**Задание** 18](#_Toc89072943)

[**Содержание отчёта** 18](#_Toc89072944)

[Лабораторная работа 6. Классы 20](#_Toc89072945)

[Основные понятия 20](#_Toc89072946)

[Модификаторы доступа public и private 21](#_Toc89072947)

[Программа учета успеваемости студентов 21](#_Toc89072948)

[Отделение данных от логики 24](#_Toc89072949)

[Создание объекта через указатель 26](#_Toc89072950)

[Конструктор и деструктор класса 27](#_Toc89072951)

[Лабораторная работа 7. Ещё классы 28](#_Toc89072952)

[Конструктор Students 28](#_Toc89072953)

[Сохранение оценок в файл 31](#_Toc89072954)

[Деструктор Students 32](#_Toc89072955)

[Приложения 36](#_Toc89072956)

[Решение проблем 36](#_Toc89072957)

# Лабораторная работа 1. Знакомство со средой разработки

Цель: сформировать практические навыки разработки консольных приложений на языке С++ в интегрированной среде разработки Visual Studio 2013.

**Предварительные замечания**

**Файловая система**

В дисплейных залах СГУ пользователи имеют право работать только в каталоге c:\windows\TEMP.

Рекомендуется перед началом ЛР создать в этой папке свою личную папку.

Например, **c:\windows\TEMP\avramenko\_an**

**Среда разработки**

В качестве среды разработки под Windows рекомендуется использовать [Visual Studio 2008](https://code-live.ru/post/visual-studio-2008-express-edition-russian/%22%20%5Co%20%22%D0%A1%D0%BA%D0%B0%D1%87%D0%B0%D1%82%D1%8C%20Visual%20Studio%202008) (800 Мб), если у вас Windows XP, [Visual Studio 2013](http://download.microsoft.com/download/5/9/0/590B3BE3-9FC4-4FD2-9056-4B52020C7F90/vs2013.4_dskexp_RUS.iso) (6.5 Гб) для Windows 7 и выше, или [Dev C++ 5](https://code-live.ru/post/dev-cpp-free-cpp-ide-for-windows/%22%20%5Co%20%22%D0%A1%D0%BA%D0%B0%D1%87%D0%B0%D1%82%D1%8C%20Dev%20C%2B%2B%205) (41 Мб) для тех, у кого слишком узкий канал.

**Создание проекта**

Откройте меню «Файл → Создать → Проект». Перейдите на вкладку «Общие» и выберите «Пустой проект». Придумайте проекту любое название, например «LR\_1» и нажмите «OK».

В окне обозревателя решений (обычно он находится в левом верхнем углу) щелкните правой кнопкой на папке «файлы исходного кода». В диалоговом окне выберите пункт меню «Добавить → Создать элемент». Введите название для нового файла — main.cpp и нажмите кнопку «Добавить».

**Код первой программы**

Наберите следующий код:

**#include <iostream>**

**#include <cstdlib> // для system**

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

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

{

 cout << "Hello, world!" << endl;

 system("pause"); *// Только для тех, у кого MS Visual Studio*

 **return** 0;

}

**Описание синтаксиса**

Директива #include используется для подключения других файлов в код. Строка #include <iostream>, будет заменена содержимым файла «iostream.h», который находится в стандартной библиотеке языка и отвечает за ввод и вывод данных на экран.

#include <cstdlib> подключает стандартную библиотеку языка С. Это подключение необходимо для работы функции system.

Содержимое третьей строки — using namespace std; указывает на то, что мы используем по умолчанию [пространство имен](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BC%D1%91%D0%BD_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) с названием «std». Все то, что находится внутри фигурных скобок функции int main() {} будет автоматически выполняться после запуска программы.

Строка cout << "Hello, world!" << endl; говорит программе выводить сообщение с текстом **«Hello, world»** на экран.

Оператор cout предназначен для вывода текста на экран командной строки. После него ставятся две угловые кавычки (<<). Далее идет текст, который должен выводиться. Он помещается в двойные кавычки. Оператор endl переводит строку на уровень ниже.

Если в процессе выполнения произойдет какой-либо сбой, то будет сгенерирован код ошибки, отличный от нуля. Если же работа программы завершилась без сбоев, то код ошибки будет равен нулю. Команда return 0 необходима для того, чтобы передать операционной системе сообщение об удачном завершении программы.

В конце каждой команды ставится **точка с запятой.**

**Компиляция и запуск**

Теперь скомпилируйте и запустите программу. Тем, кто пользуется MS Visual Studio, нужно нажать сочетание клавиш «Ctrl+F5».

Если программа собралась с первого раза, то хорошо. Если компилятор говорит о наличии ошибок, значит вы что-то сделали неправильно.

Прочитайте текст ошибки и попробуйте ее исправить своими силами. Типичные проблемы и методы их решения приведены в Приложениях, в конце документа. Если не получится, обратитесь к ~~соседу~~ преподавателю.

В качестве дополнительного задания, переделайте эту программу так, чтобы вместо, сообщения «Hello, World» выводилось сообщение «Hello, \_ваше имя\_», поэкспериментируйте с дополнительными элементами << endl.

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы и его описание
4. Описание действий и результатов
5. Выводы

# Лабораторная работа 2. Арифметика

## Типы данных

В языке С++ все переменные имеют определенный тип данных. Например, переменная, имеющая целочисленный тип, не может содержать ничего кроме целых чисел, а переменная с плавающей точкой — только дробные числа.

**Тип данных** присваивается переменной при ее объявлении или инициализации. Ниже приведены основные типы данных языка C++, которые нам понадобятся.

### Основные типы данных в C++

* **int** — целочисленный тип данных.
* **float** — тип данных с плавающей запятой.
* **double** — тип данных с плавающей запятой двойной точности.
* **char** — символьный тип данных.
* **bool** — логический тип данных.

## Объявление переменной

Объявление переменной в C++ происходит таким образом: сначала указывается тип данных для этой переменной, а затем название этой переменной.

### Пример объявления переменных

**int** a; *// объявление переменной a целого типа.*

**float** b; *// объявление переменной b типа данных с плавающей запятой.*

**double** c = 14.2; *// инициализация переменной типа double.*

**char** d = 's'; *// инициализация переменной типа char.*

**bool** k = true; *// инициализация логической переменной k.*

* Заметьте, что в C++ **оператор присваивания** **(=)** — не является знаком равенства и не может использоваться для сравнения значений.
* Оператор сравнения записывается как «двойное равно» — ==. Для запоминания используем подсказку: «**двойное** равно» читается как **два слова** «равно ли»?
* Присваивание используется для сохранения определенного значение в переменной. Например, запись вида a = 10 задает переменной a значение 10.

## Простой калькулятор на C++

Разработаем программу-калькулятор, которая будет принимать от пользователя два целых числа, а затем определять их сумму:

**#include <iostream>**

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

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

{

 setlocale(0, "");

 */\*7\*/* **int** a, b; *// объявление двух переменных a и b целого типа данных.*

 cout << "Введите первое число: ";

 cin >> a; *// пользователь присваивает переменной a какое-либо значение.*

 cout << "Введите второе число: ";

 cin >> b;

 */\*12\*/* **int** c = a + b; *// новой переменной c присваиваем значение суммы введенных пользователем данных.*

 cout << "Сумма чисел = " << c << endl; *// вывод ответа.*

 **return** 0;

}

### Разбор кода

В 7-й строке кода программы мы объявляем переменные «a» и «b» целого типа int. В следующей строке кода выводится сообщение пользователю, чтобы он ввел с клавиатуры первое число.

В 9-й строке стоит конструкция — cin >>. С помощью нее у пользователя запрашивается ввод значения переменной «a» с клавиатуры. Аналогичным образом задается значение переменной «b».

В 12-й строке мы производим инициализацию переменной «c» суммой переменных «a» и «b». Далее находится оператор cout, который выводит на экран строку и значение переменной «c».

При выводе переменных, они не заключаются в кавычки, в отличие от строк.

## Задание

Модифицируйте калькулятор согласно варианту.

1. Сложение трёх чисел
2. Вычитание трёх чисел
3. Умножение трёх чисел
4. Сумма корней двух чисел
5. Корень\* суммы двух чисел
6. a – b + x
7. a + b - x
8. a \* b / x
9. a / b \* x
10. a \* b + x

\*) Потребуется подключение дополнительной библиотеки для sqrt, см. подсказки в интернете.

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы и его описание
4. Описание действий и результатов
5. Выводы

# Лабораторная работа 3. Условия

Иногда некоторые части программы должны выполняться не всегда, а при соблюдении некоторых условий.

Например, рассмотрим задачу: если введённое с клавиатуры число больше десяти, то программа должна выполнить одно действие, иначе — другое. Выведем надпись по условию.

**#include <iostream>**

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

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

{

 setlocale(0, "");

 **double** num;

 cout << "Введите произвольное число: ";

 cin >> num;

 **if** (num < 10) { *// Если введенное число меньше 10.*

 cout << "Это число меньше 10." << endl;

 } **else** { *// иначе*

 cout << "Это число больше либо равно 10." << endl;

 }

 **return** 0;

}

Выполните программу; при вводе числа, меньшего десяти, будет выводиться соответствующее сообщение. Если введенное число окажется большим, либо равным десяти — отобразится другое сообщение.

## Оператор if

**Оператор if** служит для того, чтобы выполнить какую-либо операцию в том случае, когда условие является верным. Условная конструкция в С++ всегда записывается в круглых скобках после оператора if.

Внутри фигурных скобок указывается тело условия. Если условие выполнится, то начнется выполнение всех команд, которые находятся между фигурными скобками.

## Пример конструкции ветвления

**if** (num < 10) { *// Если введенное число меньше 10.*

 cout << "Это число меньше 10." << endl;

} **else** { *// иначе*

 cout << "Это число больше либо равно 10." << endl;

}

Здесь говорится: «**Если** переменная num меньше 10 — вывести соответствующее сообщение. **Иначе**, вывести другое сообщение».

Усовершенствуем программу так, чтобы она выводила сообщение, о том, что переменная num равна десяти:

**if** (num < 10) { *// Если введенное число меньше 10.*

 cout << "Это число меньше 10." << endl;

} **else** **if** (num == 10) {

 cout << "Это число равно 10." << endl;

} **else** { *// иначе*

 cout << "Это число больше 10." << endl;

}

Здесь мы проверяем три условия:

* Первое — когда введенное число меньше 10-ти
* Второе — когда число равно 10-ти
* И третье — когда число больше десяти

Заметьте, что во втором условии, при проверке равенства, мы используем оператор равенства — ==, а не [оператор присваивания](http://code-live.ru/post/cpp-variables-and-datatypes/#assignment-statement), потому что мы не изменяем значение переменной при проверке, а сравниваем ее текущее значение с числом 10.

Если поставить оператор присваивания в условии, то при проверке условия, **значение переменной изменится**, после чего условие будет выполнено.

Каждому **оператору if** соответствует только один оператор *else*. Совокупность этих операторов — **else if** означает, что если не выполнилось предыдущее условие, то проверить данное. Если ни одно из условий не верно, то выполняется тело оператора *else*.

Если после оператора **if**, **else** или их связки **else if** должна выполняться только одна команда, то фигурные скобки можно не ставить. Предыдущую программу можно записать следующим образом:

**#include <iostream>**

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

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

{

 setlocale(0, "");

 **double** num;

 cout << "Введите произвольное число: ";

 cin >> num;

 **if** (num < 10) *// Если введенное число меньше 10.*

 cout << "Это число меньше 10." << endl;

 **else** **if** (num == 10)

 cout << "Это число равно 10." << endl;

 **else** *// иначе*

 cout << "Это число больше 10." << endl;

 **return** 0;

}

Такой метод записи выглядит более компактно. Если при выполнении условия нам требуется выполнить более одной команды, то фигурные скобки необходимы. Например:

**#include <iostream>**

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

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

{

 setlocale(0, "");

 **double** num;

 **int** k;

 cout << "Введите произвольное число: ";

 cin >> num;

 **if** (num < 10) { *// Если введенное число меньше 10.*

 cout << "Это число меньше 10." << endl;

 k = 1;

 } **else** **if** (num == 10) {

 cout << "Это число равно 10." << endl;

 k = 2;

 } **else** { *// иначе*

 cout << "Это число больше 10." << endl;

 k = 3;

 }

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

 **return** 0;

}

Данная программа проверяет значение переменной num. Если она меньше 10, то присваивает переменной k значение единицы. Если переменная num равна десяти, то присваивает переменной k значение двойки. В противном случае — значение тройки. После выполнения ветвления, значение переменной k выводится на экран.

## Задание

Разработать специализированный калькулятор, который реализует вычисление значения функции по варианту.

1. Введите число t – температуру воды. Проанализируйте состояние (лёд, вода, пар) и выведите его на экран.
2. Введите число х. Если х больше 100, то прибавить 1000, если меньше – умножить на 2.
3. Введите два числа (x, y). Первое число умножить на 2, ко второму прибавить 20. На экран выдать получившиеся два числа в порядке возрастания.
4. Введите два числа (x, y). Первое число умножить на 10, ко второму прибавить 100. На экран выдать получившиеся два числа в порядке возрастания.
5. Введите три числа (a, b, c). Найти сумму чисел, сравнить со значением 100, выдать соответствующее сообщение и сумму. Пример: Сумма **больше** 100: 777.
6. Введите число t – температуру воды. Проанализируйте состояние (лёд, вода, пар) и выведите его на экран.
7. Введите число х. Если х больше 10, то умножить на 100, если меньше – разделить на 2.
8. Введите два числа (x, y). Найти их корни. На экран выдать получившиеся два числа в порядке возрастания.
9. Введите два числа (x, y). Из большего вычесть меньшее. Результат вывести на экран.
10. Введите три числа (a, b, c). Найти сумму чисел, сравнить со значением 100, выдать соответствующее сообщение и сумму. Пример: Сумма **больше** 100: 777.

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы и его описание
4. Описание действий и результатов
5. Выводы

# Лабораторная работа 4. Циклы

## Цикл for

Если мы знаем точное количество действий (итераций) цикла, то можем использовать **цикл for**. Синтаксис его выглядит примерно так:

**for** (действие до начала цикла;

 условие продолжения цикла;

 действия в конце каждой итерации цикла) {

 инструкция цикла;

 инструкция цикла 2;

 инструкция цикла N;

}

Итерацией цикла называется один проход этого цикла

Существует частный случай этой записи, который мы сегодня и разберем:

**for** (счетчик = значение; счетчик < значение; шаг цикла) {

 тело цикла;

}

**Счетчик цикла** — это переменная, в которой хранится количество проходов данного цикла.

### Описание синтаксиса

1. Сначала присваивается первоначальное значение счетчику, после чего ставится точка с запятой.
2. Затем задается конечное значение счетчика цикла. После того, как значение счетчика достигнет указанного предела, цикл завершится. Снова ставим точку с запятой.
3. Задаем шаг цикла. **Шаг цикла** — это значение, на которое будет увеличиваться или уменьшаться счетчик цикла при каждом проходе.

### Пример кода

Напишем программу, которая будет считать сумму всех чисел от 1 до 1000.

**#include <iostream>**

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

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

{

 **int** i; *// счетчик цикла*

 **int** sum = 0; *// сумма чисел от 1 до 1000.*

 setlocale(0, "");

 **for** (i = 1; i <= 1000; i++) *// задаем начальное значение 1, конечное 1000 и задаем шаг цикла - 1.*

 {

 sum = sum + i;

 }

 cout << "Сумма чисел от 1 до 1000 = " << sum << endl;

 **return** 0;

}

Если мы скомпилируем этот код и запустим программу, то она покажет нам ответ: 500500. Это и есть сумма всех целых чисел от 1 до 1000. Если считать это вручную, понадобится очень много времени и сил. Цикл выполнил всю рутинную работу за нас.

Заметьте, что конечное значение счетчика задано нестрогим неравенством ( <= — меньше либо равно), поскольку, если бы использовался знак «меньше», то цикл произвел бы 999 итераций, т.е. на одну меньше, чем требуется. Значение шага цикла равно единице. **i++** — это тоже самое, что и **i = i + 1** (инкремент).

В теле цикла, при каждом проходе программа увеличивает значение переменной sum на i. В начале программы переменной sum присваивается нулевое значение. Если этого не сделать, программа вылетит в [сегфолт](http://ru.wikipedia.org/wiki/%D0%9E%D1%88%D0%B8%D0%B1%D0%BA%D0%B0_%D1%81%D0%B5%D0%B3%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D0%B8).

## Цикл while

Когда мы не знаем, сколько итераций должен произвести цикл, нам понадобится цикл **while** или **do...while**. Синтаксис цикла **while** в C++ выглядит следующим образом.

while (Условие) {

 Тело цикла;

}

Данный цикл будет выполняться, пока условие, указанное в круглых скобках является истиной. Решим ту же задачу с помощью цикла **while**. Хотя здесь мы точно знаем, сколько итераций должен выполнить цикл, очень часто бывают ситуации, когда это значение неизвестно.

Ниже приведен исходный код программы, считающей сумму всех целых чисел от 1 до 1000.

**#include <iostream>**

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

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

{

 setlocale(0, "");

 **int** i = 0; *// инициализируем счетчик цикла.*

 **int** sum = 0; *// инициализируем счетчик суммы.*

 **while** (i < 1000)

 {

 i++;

 sum += i;

 }

 cout << "Сумма чисел от 1 до 1000 = " << sum << endl;

 **return** 0;

}

После компиляции программа выдаст результат, аналогичный результату работы предыдущей программы. Тут строгое неравенство в условии цикла и счетчик i инициализирован нулем, так как в цикле **while** происходит на одну итерацию больше, потому он будет выполняться, до тех пор, пока значение счетчика перестает удовлетворять условию, но данная итерация все равно выполнится. Если бы мы поставили нестрогое неравенство, то цикл бы закончился, когда переменная i стала бы равна 1001 и выполнилось бы на одну итерацию больше.

Теперь давайте рассмотрим по порядку исходный код нашей программы. Сначала мы инициализируем счетчик цикла и переменную, хранящую сумму чисел.

В данном случае мы обязательно должны присвоить счетчику цикла какое-либо значение, т.к. в предыдущей программе мы это значение присваивали внутри цикла **for**, здесь же, если мы не инициализируем счетчик цикла, то в него попадет «мусор» и компилятор в лучшем случае выдаст нам ошибку, а в худшем, если программа соберется — сегфолт практически неизбежен.

Затем мы описываем условие цикла — **«пока переменная i меньше 1000 — выполняй цикл»**. При каждой итерации цикла значение переменной-счетчика i увеличивается на единицу внутри цикла.

Когда выполнится 1000 итераций цикла, счетчик станет равным 999 и следующая итерация уже не выполнится, поскольку 1000 не меньше 1000. Выражение sum += i является укороченной записью sum = sum + i.

После окончания выполнения цикла, выводим сообщение с ответом.

## Цикл do while

Цикл **do while** очень похож на цикл **while**. Единственное их различие в том, что при выполнении цикла **do while** один проход цикла будет выполнен независимо от условия. Решение задачи на поиск суммы чисел от 1 до 1000, с применением цикла *do while*.

**#include <iostream>**

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

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

{

 setlocale(0, "");

 **int** i = 0; *// инициализируем счетчик цикла.*

 **int** sum = 0; *// инициализируем счетчик суммы.*

 **do** {*// выполняем цикл.*

 i++;

 sum += i;

 } **while** (i < 1000); *// пока выполняется условие.*

 cout << "Сумма чисел от 1 до 1000 = " << sum << endl;

 **return** 0;

}

Принципиального отличия нет, но если присвоить переменной i значение, большее, чем 1000, то цикл все равно выполнит хотя бы один проход.

**Задание**

Написать программу, находящую все четные числа от х до n, где х – номер варианта, n вводится с клавиатуры.

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы и его описание
4. Описание действий и результатов
5. Выводы

# Лабораторная работа 5. Функции

Функции — один из самых важных компонентов языка C++.

* Любая функция имеет тип, также, как и любая переменная.
* Функция может возвращать значение, тип которого в большинстве случаев аналогично типу самой функции.
* Если функция не возвращает никакого значения, то она должна иметь тип **void** (такие функции иногда называют процедурами)
* При объявлении функции, после ее типа должно находиться имя функции и две круглые скобки — открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов функции, которых также может не быть вообще.
* После списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции.
* В конце тела функции обязательно ставится закрывающая фигурная скобка.

## Пример построения функции

**#include <iostream>**

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

**void** **function\_name** ()

{

 cout << "Hello, world" << endl;

}

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

{

 function\_name(); *// Вызов функции*

 **return** 0;

}

Перед вами тривиальная программа, **Hello, world**, только реализованная с использованием функций.

Если мы хотим вывести «Hello, world» где-то еще, нам просто нужно вызвать соответствующую функцию. В данном случае это делается так: function\_name();. Вызов функции имеет вид имени функции с последующими круглыми скобками. Эти скобки могут быть пустыми, если функция не имеет аргументов. Если же аргументы в самой функции есть, их необходимо указать в круглых скобках.

Также существует такое понятие, как параметры функции по умолчанию. Такие параметры можно не указывать при вызове функции, т.к. они примут значение по умолчанию, указанно после знака присваивания после данного параметра и списке всех параметров функции.

В предыдущих примерах мы использовали функции типа void, которые не возвращают никакого значения. Как многие уже догадались, оператор return используется для возвращения вычисляемого функцией значения.

Рассмотрим пример функции, возвращающей значение на примере проверки пароля.

**#include <iostream>**

**#include <string>**

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

string **check\_pass** (string password)

{

 string valid\_pass = "qwerty123";

 string error\_message;

 **if** (password == valid\_pass) {

 error\_message = "Доступ разрешен.";

 } **else** {

 error\_message = "Неверный пароль!";

 }

 **return** error\_message;

}

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

{

 string user\_pass;

 cout << "Введите пароль: ";

 getline (cin, user\_pass);

 string error\_msg = check\_pass (user\_pass);

 cout << error\_msg << endl;

 **return** 0;

}

В данном случае функция **check\_pass** имеет тип **string**, следовательно, она будет возвращать только значение типа string (строку). Давайте рассмотрим алгоритм работы этой программы.

Самой первой выполняется функция **main()**, которая должна присутствовать в каждой программе. Теперь объявляем переменную **user\_pass** типа string, затем выводим пользователю сообщение «Введите пароль», который после ввода попадает в строку user\_pass. А вот дальше начинает работать наша собственная функция check\_pass().

В качестве аргумента этой функции передается строка, введенная пользователем.

Аргумент функции — это, если сказать простым языком переменные или константы вызывающей функции, которые будет использовать вызываемая функция.

При объявлении функций создается **формальный параметр**, имя которого может отличаться от параметра, передаваемого при вызове этой функции. Но типы формальных параметров и передаваемых функции аргументов в большинстве случаев должны быть аналогичны.

После того, как произошел вызов функции check\_pass(), начинает работать данная функция. Если функцию нигде не вызвать, то этот код будет проигнорирован программой. Итак, мы передали в качестве аргумента строку, которую ввел пользователь.

Теперь эта строка в полном распоряжении функции (хочу обратить Ваше внимание на то, что переменные и константы, объявленные в разных функциях независимы друг от друга, они даже могут иметь одинаковые имена. В следующих уроках я расскажу о том, что такое область видимости, локальные и глобальные переменные).

Теперь мы проверяем, правильный ли пароль ввел пользователь или нет. если пользователь ввел правильный пароль, присваиваем переменной error\_message соответствующее значение. если нет, то сообщение об ошибке.

После этой проверки мы **возвращаем** переменную error\_message. На этом работа нашей функции закончена. А теперь, в функции main(), то значение, которое возвратила наша функция мы присваиваем переменной error\_msg и выводим это значение (строку) на экран терминала.

Также, можно организовать повторный ввод пароля с помощью [**рекурсии**](https://code-live.ru/post/cpp-recursive-factorial-calculating/). Если объяснять вкратце, рекурсия — вызов функцией самой себя. Ещё один пример:

**#include <iostream>**

**#include <string>**

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

**bool** **password\_is\_valid** (string password)

{

 string valid\_pass = "secret"; // Правильный пароль - secret

 **if** (valid\_pass == password)

 **return** true;

 **else**

 **return** false;

}

**void** **get\_pass** ()

{

 string user\_pass;

 cout << "Введите пароль: ";

 getline(cin, user\_pass);

 **if** (!password\_is\_valid(user\_pass)) {

 cout << "Неверный пароль!" << endl;

 get\_pass (); *// Здесь делаем рекурсию*

 } **else** {

 cout << "Доступ разрешен." << endl;

 }

}

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

{

 get\_pass ();

 **return** 0;

}

Функции очень сильно облегчают работу программисту и намного повышают читаемость и понятность кода, в том числе и для самого разработчика.

**Задание**

Вариант **Чётный**.

Напишите функцию min(a, b), вычисляющую минимум двух чисел. Разработайте программу, вычисляющую минимум 4 чисел с помощью функции min. Считайте четыре целых числа с клавиатуры и выведите их минимум.

Вариант **Нечётный**.

Напишите функцию max(a, b), вычисляющую минимум двух чисел. Разработайте программу, вычисляющую минимум 4 чисел с помощью функции min. Считайте четыре целых числа с клавиатуры и выведите их минимум.

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы и его описание
4. Описание действий и результатов
5. Выводы

# Лабораторная работа 6. Классы

Весь реальный мир состоит из объектов. Города состоят из районов, в каждом районе есть свои названия улиц, на каждой улице находятся жилые дома, которые также состоят из объектов.

Практически любой материальный предмет можно представить в виде совокупности объектов, из которых он состоит. Допустим, что нам нужно написать программу для учета успеваемости студентов. Можно представить группу студентов, как класс языка C++. Назовем его Students.

**class** Students {

 *// Имя студента*

 std::string name;

 *// Фамилия*

 std::string last\_name;

 *// Пять промежуточных оценок студента*

 **int** scores[5];

 *// Итоговая оценка за семестр*

 **float** average\_ball;

};

## Основные понятия

Классы в программировании состоят из свойств и методов. Свойства — это любые данные, которыми можно характеризовать объект класса. В нашем случае, объектом класса является студент, а его свойствами — имя, фамилия, оценки и средний балл.

У каждого студента есть имя — name и фамилия last\_name . Также, у него есть промежуточные оценки за весь семестр. Эти оценки мы будем записывать в целочисленный [массив](http://code-live.ru/post/cpp-arrays/) из пяти элементов. После того, как все пять оценок будут проставлены, определим средний балл успеваемости студента за весь семестр — свойство average\_ball.

Методы — это [функции](http://code-live.ru/post/cpp-functions/), которые могут выполнять какие-либо действия над данными (свойствами) класса. Добавим в наш класс функцию calculate\_average\_ball(), которая будет определять средний балл успеваемости ученика.

* **Методы** класса — это его функции.
* **Свойства** класса — его [переменные](http://code-live.ru/post/cpp-variables-and-datatypes/).

**class** Students {

 **public**:

 *// Функция, считающая средний балл*

 **void** **calculate\_average\_ball**()

 {

 **int** sum = 0; *// Сумма всех оценок*

 **for** (**int** i = 0; i < 5; ++i) {

 sum += scores[i];

 }

 *// считаем среднее арифметическое*

 average\_ball = sum / 5.0;

 }

 *// Имя студента*

 std::string name;

 *// Фамилия*

 std::string last\_name;

 *// Пять промежуточных оценок студента*

 **int** scores[5];

 **private**:

 *// Итоговая оценка за семестр*

 **float** average\_ball;

};

Функция calculate\_average\_ball() просто делит сумму всех промежуточных оценок на их количество.

## Модификаторы доступа public и private

Все свойства и методы классов имеют права доступа. По умолчанию, все содержимое класса является доступным для чтения и записи только для него самого. Для того, чтобы разрешить доступ к данным класса извне, используют модификатор доступа public. Все функции и переменные, которые находятся после модификатора public, становятся доступными из всех частей программы.

Закрытые данные класса размещаются после модификатора доступа private. Если отсутствует модификатор public, то все функции и переменные, по умолчанию являются закрытыми (как в первом примере).

Обычно, приватными делают все свойства класса, а публичными — его методы. Все действия с закрытыми свойствами класса реализуются через его методы. Рассмотрим следующий код.

**class** Students {

 **public**:

 *// Установка среднего балла*

 **void** **set\_average\_ball**(**float** ball)

 {

 average\_ball = ball;

 }

 *// Получение среднего балла*

 **float** **get\_average\_ball**()

 {

 **return** average\_ball;

 }

 std::string name;

 std::string last\_name;

 **int** scores[5];

 **private**:

 **float** average\_ball;

};

Мы не можем напрямую обращаться к закрытым данными класса. Работать с этими данными можно только посредством методов этого класса. В примере выше, мы используем функцию get\_average\_ball() для получения средней оценки студента, и set\_average\_ball() для выставления этой оценки.

Функция set\_average\_ball() принимает средний балл в качестве параметра и присваивает его значение закрытой переменной average\_ball. Функция get\_average\_ball() просто возвращает значение этой переменной.

## Программа учета успеваемости студентов

Создадим программу, которая будет заниматься учетом успеваемости студентов в группе. Создайте заголовочный файл **students.h**, в котором будет находиться класс Students.

*/\* students.h \*/*

**#include <string>**

**class** Students {

 **public**:

 *// Установка имени студента*

 **void** **set\_name**(std::string student\_name)

 {

 name = student\_name;

 }

 *// Получение имени студента*

 std::string **get\_name**()

 {

 **return** name;

 }

 *// Установка фамилии студента*

 **void** **set\_last\_name**(std::string student\_last\_name)

 {

 last\_name = student\_last\_name;

 }

 *// Получение фамилии студента*

 std::string **get\_last\_name**()

 {

 **return** last\_name;

 }

 *// Установка промежуточных оценок*

 **void** **set\_scores**(**int** student\_scores[])

 {

 **for** (**int** i = 0; i < 5; ++i) {

 scores[i] = student\_scores[i];

 }

 }

 *// Установка среднего балла*

 **void** **set\_average\_ball**(**float** ball)

 {

 average\_ball = ball;

 }

 *// Получение среднего балла*

 **float** **get\_average\_ball**()

 {

 **return** average\_ball;

 }

 **private**:

 *// Промежуточные оценки*

 **int** scores[5];

 *// Средний балл*

 **float** average\_ball;

 *// Имя*

 std::string name;

 *// Фамилия*

 std::string last\_name;

};

Мы добавили в наш класс новые методы, а также сделали приватными все его свойства. Функция set\_name() сохраняет имя студента в переменной name, а get\_name() возвращает значение этой переменной. Принцип работы функций set\_last\_name() и get\_last\_name() аналогичен.

Функция set\_scores() принимает массив с промежуточными оценками и сохраняет их в приватную переменную int scores[5].

Теперь создайте файл **main.cpp** со следующим содержимым.

*/\* main.cpp \*/*

**#include <iostream>**

**#include "students.h"**

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

{

 *// Создание объекта класса Student*

 Students student;

 std::string name;

 std::string last\_name;

 *// Ввод имени с клавиатуры*

 std::cout << "Name: ";

 getline(std::cin, name);

 *// Ввод фамилии*

 std::cout << "Last name: ";

 getline(std::cin, last\_name);

 *// Сохранение имени и фамилии в объект класса Students*

 student.set\_name(name);

 student.set\_last\_name(last\_name);

 *// Оценки*

 **int** scores[5];

 *// Сумма всех оценок*

 **int** sum = 0;

 *// Ввод промежуточных оценок*

 **for** (**int** i = 0; i < 5; ++i) {

 std::cout << "Score " << i+1 << ": ";

 std::cin >> scores[i];

 *// суммирование*

 sum += scores[i];

 }

 *// Сохраняем промежуточные оценки в объект класса Student*

 student.set\_scores(scores);

 *// Считаем средний балл*

 **float** average\_ball = sum / 5.0;

 *// Сохраняем средний балл в объект класса Students*

 student.set\_average\_ball(average\_ball);

 *// Выводим данные по студенту*

 std::cout << "Average ball for " << student.get\_name() << " "

 << student.get\_last\_name() << " is "

 << student.get\_average\_ball() << std::endl;

 **return** 0;

}

В самом начале программы создается объект класса Students. Сам класс является только описанием его объекта. Класс Students является описанием любого из студентов, у которого есть имя, фамилия и возможность получения оценок.

Объект класса Students характеризует конкретного студента. Если мы захотим выставить оценки всем ученикам в группе, то будем создавать новый объект для каждого из них. Использование классов хорошо подходит для описания объектов реального мира.

После создания объекта student, мы вводим с клавиатуры фамилию, имя и промежуточные оценки для конкретного ученика. Пусть у студента есть 5 оценок за семестр — две тройки, две четверки и одна пятерка.

Введенные данные мы передаем **set**-функциям, которые присваивают их закрытым переменным класса. После того, как были введены промежуточные оценки, мы высчитываем средний балл на основе этих оценок, а затем сохраняем это значение в закрытом свойстве average\_ball, с помощью функции set\_average\_ball().

Скомпилируйте и запустите программу.



## Отделение данных от логики

Вынесем реализацию всех методов класса в отдельный файл **students.cpp**.

*/\* students.cpp \*/*

**#include <string>**

**#include "students.h"**

*// Установка имени студента*

**void** Students::set\_name(std::string student\_name)

{

 Students::name = student\_name;

}

*// Получение имени студента*

std::string Students::get\_name()

{

 **return** Students::name;

}

*// Установка фамилии студента*

**void** Students::set\_last\_name(std::string student\_last\_name)

{

 Students::last\_name = student\_last\_name;

}

*// Получение фамилии студента*

std::string Students::get\_last\_name()

{

 **return** Students::last\_name;

}

*// Установка промежуточных оценок*

**void** Students::set\_scores(**int** scores[])

{

 **for** (**int** i = 0; i < 5; ++i) {

 Students::scores[i] = scores[i];

 }

}

*// Установка среднего балла*

**void** Students::set\_average\_ball(**float** ball)

{

 Students::average\_ball = ball;

}

*// Получение среднего балла*

**float** Students::get\_average\_ball()

{

 **return** Students::average\_ball;

}

А в заголовочном файле **students.h** оставим только прототипы этих методов.

*/\* students.h \*/*

**#pragma once */\* Защита от двойного подключения заголовочного файла \*/***

**#include <string>**

**class** Students {

 **public**:

 *// Установка имени студента*

 **void** **set\_name**(std::string);

 *// Получение имени студента*

 std::string **get\_name**();

 *// Установка фамилии студента*

 **void** **set\_last\_name**(std::string);

 *// Получение фамилии студента*

 std::string **get\_last\_name**();

 *// Установка промежуточных оценок*

 **void** **set\_scores**(**int** []);

 *// Установка среднего балла*

 **void** **set\_average\_ball**(**float**);

 *// Получение среднего балла*

 **float** **get\_average\_ball**();

 **private**:

 *// Промежуточные оценки*

 **int** scores[5];

 *// Средний балл*

 **float** average\_ball;

 *// Имя*

 std::string name;

 *// Фамилия*

 std::string last\_name;

};

Такой подход называется [абстракцией данных](http://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) — одного из [фундаментальных принципов](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#.D0.9E.D1.81.D0.BD.D0.BE.D0.B2.D0.BD.D1.8B.D0.B5_.D0.BF.D0.BE.D0.BD.D1.8F.D1.82.D0.B8.D1.8F) объектно-ориентированного программирования. К примеру, если кто-то другой захочет использовать наш класс в своем коде, ему не обязательно знать, как именно высчитывается средний балл. Он просто будет использовать функцию calculate\_average\_ball() из второго примера, не вникая в алгоритм ее работы.

Над крупными проектами обычно работает несколько программистов. Каждый из них занимается написанием определенной части продукта. В современных масштабах кода одному человеку нереально запомнить, как работает каждая из внутренних функций проекта. В нашей программе, мы используем оператор потокового вывода cout, не задумываясь о том, как он реализован на низком уровне. Кроме того, [отделение данных от логики](http://ru.wikipedia.org/wiki/Model-View-Controller) является хорошим тоном программирования.

Каждый класс в C++ использует свое пространство имен. Это сделано для того, чтобы избежать конфликтов при именовании переменных и функций. В файле students.cpp мы используем оператор принадлежности :: перед именем каждой функции. Это делается для того, чтобы указать компилятору, что эти функции принадлежат классу Students.

## Создание объекта через указатель

При создании объекта, лучше не копировать память для него, а выделять ее в **куче** с помощью [указателя](https://code-live.ru/post/cpp-pointers/). И освобождать ее после того, как мы закончили работу с объектом. Реализуем это в нашей программе, немного изменив содержимое файла **main.cpp**.

*/\* main.cpp \*/*

**#include <iostream>**

**#include "students.h"**

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

{

 *// Выделение памяти для объекта Students*

 Students \*student = **new** Students;

 std::string name;

 std::string last\_name;

 *// Ввод имени с клавиатуры*

 std::cout << "Name: ";

 getline(std::cin, name);

 *// Ввод фамилии*

 std::cout << "Last name: ";

 getline(std::cin, last\_name);

 *// Сохранение имени и фамилии в объект класса Students*

 student->set\_name(name);

 student->set\_last\_name(last\_name);

 *// Оценки*

 **int** scores[5];

 *// Сумма всех оценок*

 **int** sum = 0;

 *// Ввод промежуточных оценок*

 **for** (**int** i = 0; i < 5; ++i) {

 std::cout << "Score " << i+1 << ": ";

 std::cin >> scores[i];

 *// суммирование*

 sum += scores[i];

 }

 *// Сохраняем промежуточные оценки в объект класса Student*

 student->set\_scores(scores);

 *// Считаем средний балл*

 **float** average\_ball = sum / 5.0;

 *// Сохраняем средний балл в объект класса Students*

 student->set\_average\_ball(average\_ball);

 *// Выводим данные по студенту*

 std::cout << "Average ball for " << student->get\_name() << " "

 << student->get\_last\_name() << " is "

 << student->get\_average\_ball() << std::endl;

 *// Удаление объекта student из памяти*

 **delete** student;

 **return** 0;

}

При создании статического объекта, для доступа к его методам и свойствам, используют операция прямого обращения — «**.**» (символ точки). Если же память для объекта выделяется посредством указателя, то для доступа к его методам и свойствам используется оператор косвенного обращения — «**->**».

## Конструктор и деструктор класса

Конструктор класса — это специальная функция, которая автоматически вызывается сразу после создания объекта этого класса. Он не имеет типа возвращаемого значения и должен называться также, как класс, в котором он находится. По умолчанию, заполним двойками массив с промежуточными оценками студента.

**class** Students {

 **public**:

 *// Конструктор класса Students*

 Students(**int** default\_score)

 {

 **for** (**int** i = 0; i < 5; ++i) {

 scores[i] = default\_score;

 }

 }

 **private**:

 **int** scores[5];

};

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

{

 *// Передаем двойку в конструктор*

 Students \*student = **new** Students(2);

 **return** 0;

}

Мы можем исправить двойки, если ученик будет хорошо себя вести, и вовремя сдавать домашние задания.

Деструктор класса вызывается при уничтожении объекта. Имя деструктора аналогично имени конструктора, только в начале ставится знак тильды **~**. Деструктор не имеет входных параметров.

**#include <iostream>**

**class** Students {

 **public**:

 *// Деструктор*

 ~Students()

 {

 std::cout << "Memory has been cleaned. Good bye." << std::endl;

 }

};

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

{

 Students \*student = **new** Students;

 *// Уничтожение объекта*

 **delete** student;

 **return** 0;

}



**Задание**

Вариант **Один для всех**.

Откомпилировать программу учёта успеваемости студентов, пользуясь понятием класс.

Продемонстрируйте её работу.

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы с использованием понятия **класс** и его описание (алгоритм должен получиться не более 11 элементов).
4. Описание действий и результатов.
5. Выводы

# Лабораторная работа 7. Использование библиотеки fstream

Более детально познакомимся с **конструкторами** и **деструкторами** класса, а также научимся работать с файлами в потоковом режиме, с помощью библиотеки **fstream**. Продолжим написание программы учета оценок.

## Конструктор Students

Добавим в класс Students конструктор, который будет принимать имя и фамилию ученика, и сохранять эти значения в соответствующих переменных класса.

*// Конструктор Students*

Students::Students(std::string name, std::string last\_name)

{

 Students::set\_name(name);

 Students::set\_last\_name(last\_name);

}

При создании нового объекта, мы должны передать конструктору имя и фамилию студента. Иначе компиляция программы завершится с ошибкой.

std::string name = "Василий";

std::string last\_name = "Пупкин";

Students \*student = **new** Students(name, last\_name);

Теперь добавим прототип конструктора в файл **students.h**.

*/\* students.h \*/*

**#pragma once */\* Защита от двойного подключения заголовочного файла \*/***

**#include <string>**

**class** Students {

 **public**:

 *// Конструктор класса Students*

 Students(std::string, std::string);

 *// Установка имени студента*

 **void** **set\_name**(std::string);

 *// Получение имени студента*

 std::string **get\_name**();

 *// Установка фамилии студента*

 **void** **set\_last\_name**(std::string);

 *// Получение фамилии студента*

 std::string **get\_last\_name**();

 *// Установка промежуточных оценок*

 **void** **set\_scores**(**int** []);

 *// Установка среднего балла*

 **void** **set\_average\_ball**(**float**);

 *// Получение среднего балла*

 **float** **get\_average\_ball**();

 **private**:

 *// Промежуточные оценки*

 **int** scores[5];

 *// Средний балл*

 **float** average\_ball;

 *// Имя*

 std::string name;

 *// Фамилия*

 std::string last\_name;

};

В файле **students.cpp** определим сам конструктор.

*/\* students.cpp \*/*

**#include <string>**

**#include <fstream>**

**#include "students.h"**

*// Конструктор Students*

Students::Students(std::string name, std::string last\_name)

{

 Students::set\_name(name);

 Students::set\_last\_name(last\_name);

}

*// Установка имени студента*

**void** Students::set\_name(std::string student\_name)

{

 Students::name = student\_name;

}

*// Получение имени студента*

std::string Students::get\_name()

{

 **return** Students::name;

}

*// Установка фамилии студента*

**void** Students::set\_last\_name(std::string student\_last\_name)

{

 Students::last\_name = student\_last\_name;

}

*// Получение фамилии студента*

std::string Students::get\_last\_name()

{

 **return** Students::last\_name;

}

*// Установка промежуточных оценок*

**void** Students::set\_scores(**int** scores[])

{

 **int** sum = 0;

 **for** (**int** i = 0; i < 5; ++i) {

 Students::scores[i] = scores[i];

 sum += scores[i];

 }

}

*// Установка среднего балла*

**void** Students::set\_average\_ball(**float** ball)

{

 Students::average\_ball = ball;

}

*// Получение среднего балла*

**float** Students::get\_average\_ball()

{

 **return** Students::average\_ball;

}

В **main()** мы принимаем от пользователя имя и фамилию ученика, и сохраняем их во временных локальных переменных. После этого создаем новый объект класса Students, передавая его конструктору эти переменные.

*/\* main.cpp \*/*

**#include <iostream>**

**#include "students.h"**

**int** **main**(**int** argc, **char** \*argv[])

{

 *// Локальная переменная, хранящая имя ученика*

 std::string name;

 *// И его фамилию*

 std::string last\_name;

 *// Ввод имени*

 std::cout << "Name: ";

 getline(std::cin, name);

 *// И фамилии*

 std::cout << "Last name: ";

 getline(std::cin, last\_name);

 *// Передача параметров конструктору*

 Students \*student = **new** Students(name, last\_name);

 *// Оценки*

 **int** scores[5];

 *// Сумма всех оценок*

 **int** sum = 0;

 *// Ввод промежуточных оценок*

 **for** (**int** i = 0; i < 5; ++i) {

 std::cout << "Score " << i+1 << ": ";

 std::cin >> scores[i];

 *// суммирование*

 sum += scores[i];

 }

 *// Сохраняем промежуточные оценки в объект класса Student*

 student->set\_scores(scores);

 *// Считаем средний балл*

 **float** average\_ball = sum / 5.0;

 *// Сохраняем средний балл*

 student->set\_average\_ball(average\_ball);

 *// Выводим данные по студенту*

 std::cout << "Average ball for " << student->get\_name() << " "

 << student->get\_last\_name() << " is "

 << student->get\_average\_ball() << std::endl;

 *// Удаление объекта student из памяти*

 **delete** student;

 **return** 0;

}

## Сохранение оценок в файл

Чтобы после завершения работы с программой, все данные сохранялись, мы будем записывать их в текстовый файл.

Оценки каждого студента будут находится в отдельной строке. Имя и фамилии будут разделяться пробелами. После имени и фамилии ученика ставится еще один пробел, а затем перечисляются все его оценки.

Пример файла с оценками:

Василий Пупкин 5 4 5 3 3

Иван Сидоров 5 5 3 4 5

Андрей Иванов 5 3 3 3 3

Для работы с файлами мы воспользуемся библиотекой fstream, которая подключается в заголовочном файле с таким же именем.

**#include <fstream>**

*// Запись данных о студенте в файл*

**void** Students::save()

{

 std::ofstream **fout**("students.txt", std::ios::app);

 fout << Students::get\_name() << " "

 << Students::get\_last\_name() << " ";

 **for** (**int** i = 0; i < 5; ++i) {

 fout << Students::scores[i] << " ";

 }

 fout << std::endl;

 fout.close();

}

Переменная fout — это объект класса ofstream, который находится внутри библиотеки fstream. Класс ofstream используется для записи каких-либо данных во внешний файл. Кстати, у него тоже есть конструктор. Он принимает в качестве параметров имя выходного файла и режим записи.

В данном случае, мы используем режим добавления — std::ios:app (англ. append). После завершения работы с файлом, необходимо вызвать метод close() для того, чтобы закрыть файловый дескриптор.

Чтобы сохранить оценки студента, мы будем вызывать только что созданный метод save().

Students student = **new** Students("Василий", "Пупкин");

student->save();

## Деструктор Students

Логично было бы сохранять все оценки после того, как работа со студентом закончена. Для этого создадим **деструктор** класса Students, который будет вызывать метод save() перед уничтожением объекта.

*// Деструктор Students*

Students::~Students()

{

 Students::save();

}

Добавим прототипы деструктора и метода save() в **students.h**.

*/\* students.h \*/*

**#pragma once */\* Защита от двойного подключения заголовочного файла \*/***

**#include <string>**

**class** Students {

 **public**:

 *// Запись данных о студенте в файл*

 **void** **save**();

 *// Деструктор класса Students*

 ~Students();

 *// Конструктор класса Students*

 Students(std::string, std::string);

 *// Установка имени студента*

 **void** **set\_name**(std::string);

 *// Получение имени студента*

 std::string **get\_name**();

 *// Установка фамилии студента*

 **void** **set\_last\_name**(std::string);

 *// Получение фамилии студента*

 std::string **get\_last\_name**();

 *// Установка промежуточных оценок*

 **void** **set\_scores**(**int** []);

 *// Получение массива с промежуточными оценками*

 **int** \***get\_scores**();

 *// Получение строки с промежуточными оценками*

 std::string **get\_scores\_str**(**char**);

 *// Установка среднего балла*

 **void** **set\_average\_ball**(**float**);

 *// Получение среднего балла*

 **float** **get\_average\_ball**();

 **private**:

 *// Промежуточные оценки*

 **int** scores[5];

 *// Средний балл*

 **float** average\_ball;

 *// Имя*

 std::string name;

 *// Фамилия*

 std::string last\_name;

};

И определим эти функции в **students.cpp**.

*/\* students.cpp \*/*

**#include <string>**

**#include <fstream>**

**#include "students.h"**

*// Деструктор Students*

Students::~Students()

{

 Students::save();

}

*// Запись данных о студенте в файл*

**void** Students::save()

{

 std::ofstream **fout**("students.txt", std::ios::app);

 fout << Students::get\_name() << " "

 << Students::get\_last\_name() << " ";

 **for** (**int** i = 0; i < 5; ++i) {

 fout << Students::scores[i] << " ";

 }

 fout << std::endl;

 fout.close();

}

*// Конструктор Students*

Students::Students(std::string name, std::string last\_name)

{

 Students::set\_name(name);

 Students::set\_last\_name(last\_name);

}

*// Установка имени студента*

**void** Students::set\_name(std::string student\_name)

{

 Students::name = student\_name;

}

*// Получение имени студента*

std::string Students::get\_name()

{

 **return** Students::name;

}

*// Установка фамилии студента*

**void** Students::set\_last\_name(std::string student\_last\_name)

{

 Students::last\_name = student\_last\_name;

}

*// Получение фамилии студента*

std::string Students::get\_last\_name()

{

 **return** Students::last\_name;

}

*// Установка промежуточных оценок*

**void** Students::set\_scores(**int** scores[])

{

 **int** sum = 0;

 **for** (**int** i = 0; i < 5; ++i) {

 Students::scores[i] = scores[i];

 sum += scores[i];

 }

}

*// Получение массива с промежуточными оценками*

**int** \*Students::get\_scores()

{

 **return** Students::scores;

}

*// Установка среднего балла*

**void** Students::set\_average\_ball(**float** ball)

{

 Students::average\_ball = ball;

}

*// Получение среднего балла*

**float** Students::get\_average\_ball()

{

 **return** Students::average\_ball;

}

Содержимое **main.cpp** останется прежним. Скомпилируйте и запустите программу. Перед тем, как приложение завершит свою работу, в директории с исполняемым файлом будет создан новый текстовый файл с оценками — **students.txt**.



**Задание**

Вариант **Один для всех**.

Доработать и откомпилировать программу учёта успеваемости студентов, пользуясь понятием класс. Выполнить сохранение в файл.

Продемонстрируйте её работу.

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы с использованием понятия **класс** и его описание (алгоритм должен получиться не более 11 элементов).
4. Описание действий и результатов.
5. Выводы

# Лабораторная работа 8. Массивы

Переменная — это ячейка памяти компьютера, где может храниться одно единственное значение. **Массив** — это область памяти, где могут последовательно храниться несколько значений.

Возьмем группу студентов из десяти человек. У каждого из них есть фамилия. Создавать отдельную переменную для каждого студента не рационально, а работать с ними неудобно. Создадим массив, в котором будут храниться фамилии всех студентов.

## Пример инициализации массива

string students[10] = {

 "Авраменко", "Галкин", "Чайкин",

 "Петросян", "Миллер", "Ильичёв",

 "Эйлер", "Курт-Оглы", "Чуликов", "Чухраев"

};

## Описание синтаксиса

Массив создается почти так же, как и обычная переменная. Для хранения десяти фамилий нам нужен массив, состоящий из 10 элементов. Количество элементов массива задается при его объявлении и заключается в квадратные скобки.

Чтобы описать элементы массива сразу при его создании, можно использовать фигурные скобки. В фигурных скобках значения элементов массива перечисляются через запятую. В конце закрывающей фигурной скобки ставится точка с запятой.

Попробуем вывести наш массив на экран с помощью оператора **cout**.

**#include <iostream>**

**#include <string>**

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

{

 std::string students[10] = {

 "Авраменко", "Галкин", "Чайкин",

 "Петросян", "Миллер", "Ильичёв",

 "Эйлер", "Курт-Оглы", "Чуликов", "Чухраев"

 };

 std::cout << students << std::endl; *// вывести весь массив непосредственно*

 **return** 0;

}

Скомпилируйте этот код и проверьте результат. Теперь запустите программу еще раз и сравните с предыдущим результатом. В моей операционной системе вывод был следующим:

* Первый вывод: 0x7ffffe1e5820
* Второй вывод: 0x7fff7e1e5f90
* Третий вывод: 0x7ffffe1eeb40

Удивительно, выводится адрес этого массива в оперативной памяти, а никакие не «Иванов» и «Петров».

Дело в том, что при создании переменной, ей выделяется определенное место в памяти. Если мы объявляем переменную типа int, то на машинном уровне она описывается двумя параметрами — ее адресом и размером хранимых данных.

Массивы в памяти хранятся таким же образом. Массив типа int из 10 элементов описывается с помощью адреса его первого элемента и количества байт, которое может вместить этот массив. Если для хранения одного целого числа выделяется 4 байта, то для массива из десяти целых чисел будет выделено 40 байт.

Почему же при повторном запуске программы, адреса различаются? Это сделано для защиты от атак [переполнения буфера](http://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B1%D1%83%D1%84%D0%B5%D1%80%D0%B0). Такая технология называется [рандомизацией адресного пространства](http://ru.wikipedia.org/wiki/ASLR) и реализована в большинстве популярных ОС.

Попробуем вывести первый элемент массива — фамилию студента.

**#include <iostream>**

**#include <string>**

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

{

 std::string students[10] = {

 "Авраменко", "Галкин", "Чайкин",

 "Петросян", "Миллер", "Ильичёв",

 "Эйлер", "Курт-Оглы", "Чуликов", "Чухраев"

 };

 std::cout << students[0] << std::endl;

 **return** 0;

}

Смотрим, компилируем, запускаем. Убедились, что вывелся именно «Авраменко». Заметьте, что нумерация элементов массива в C++ начинается **с нуля**. Следовательно, фамилия первого студента находится в students[0], а фамилия последнего — в students[9].

В большинстве языков программирования нумерация элементов массива также начинается с нуля.

Попробуем вывести список всех студентов. Воспользуемся циклом.

## Вывод элементов массива через цикл

**#include <iostream>**

**#include <string>**

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

{

 std::string students[10] = {

 "Авраменко", "Галкин", "Чайкин",

 "Петросян", "Миллер", "Ильичёв",

 "Эйлер", "Курт-Оглы", "Чуликов", "Чухраев"

 };

 **for** (**int** i = 0; i < 10; i++) {

 std::cout << students[i] << std::endl;

 }

 **return** 0;

}

Если бы нам пришлось выводить массив из нескольких тысяч фамилий, то мы бы просто увеличили конечное значение счетчика цикла — строку for (...; i < 10; ...) заменили на for (...; i < 10000; ...).

Заметьте, что счетчик нашего цикла начинается с нуля, а заканчивается девяткой. Если вместо оператора строгого неравенства — i < 10 использовать оператор «меньше, либо равно» — i <= 10, то на последней итерации программа обратится к несуществующему элементу массива — students[10]. Это может привести к [ошибкам сегментации](http://ru.wikipedia.org/wiki/%D0%9E%D1%88%D0%B8%D0%B1%D0%BA%D0%B0_%D1%81%D0%B5%D0%B3%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D0%B8) и аварийному завершению программы.

Массив, как и любую переменную можно не заполнять значениями при объявлении.

## Объявление массива без инициализации

string students[10];

*// или*

string teachers[5];

Элементы такого массива обычно содержат в себе «мусор» из выделенной, но еще не инициализированной, памяти. Некоторые компиляторы, такие как GCC, заполняют все элементы массива нулями при его создании.

При создании статического массива, для указания его размера может использоваться только константа. Размер выделяемой памяти определяется на этапе компиляции и не может изменяться в процессе выполнения.

**int** n;

cin >> n;

string students[n]; */\* Неверно \*/*

Выделение памяти в процессе выполнения возможно при работе с [динамическими массивами](http://code-live.ru/post/cpp-dynamic-arrays/).

Заполним с клавиатуры пустой массив из 10 элементов.

## Заполнение массива с клавиатуры

**#include <iostream>**

**#include <string>**

**using** std::cout;

**using** std::cin;

**using** std::endl;

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

{

 **int** arr[10];

 *// Заполняем массив с клавиатуры*

 **for** (**int** i = 0; i < 10; i++) {

 cout << "[" << i + 1 << "]" << ": ";

 cin >> arr[i];

 }

 *// И выводим заполненный массив.*

 cout << "\nВаш массив: ";

 **for** (**int** i = 0; i < 10; ++i) {

 cout << arr[i] << " ";

 }

 cout << endl;

 **return** 0;

}

Скомпилируем программу и проверим ее работу.



Если у вас возникают проблемы при компиляции исходников— внимательно прочитайте ошибку компилятора, попробуйте проанализировать и исправить ее.

Массивы — очень важная вещь в программировании, не жалейте сил на их изучение.

**Задание**

Вариант **Чётный**.

***Создать массив типа int на 10 элементов и заполнить его случайными числами от 0 до 30. Вывести на экран. Далее перезаписать все числа, которые больше [номера\_варианта]: от исходного значения отнять 10.   Записывая новое значение, используйте***[***составные (комбинированные) операторы***](https://purecodecpp.com/archives/404)***.***

Вариант **Нечётный**.

***Заполнить массив из 50-ти элементов нечётными числами от 1 до 99. (используйте операцию***[***остаток от деления***](https://purecodecpp.com/archives/404)***, чтобы проверить число на чётность) и сравнить с [номером\_варианта]. Вывести массив, для каждого элемента текстом указать «чётный/нечётный» и «больше/меньше номера варианта».***

Примечание: рекомендуется ознакомиться с дополнительной информацией о [генерации случайных чисел](https://purecodecpp.com/archives/1012).

**Содержание отчёта**

1. Титульный лист
2. Цель
3. Алгоритм программы.
4. Описание действий и результатов.
5. Выводы

# Приложения

Решение проблем

Проблема №1

Как использовать русский язык в программах C++?

Ответ №1

Чтобы выводить кириллицу в C++ нужно подключить заголовочный файл

Windows.h:

#include <Windows.h>

И прописать следующие две строчки в функции main():

 SetConsoleCP(1251);

 SetConsoleOutputCP(1251);

Проблема №2

При выполнении программы появляется чёрное консольное окно, а

затем сразу пропадает.

Ответ №2

Некоторые компиляторы (например, Bloodshed’s Dev C++) автоматически

не задерживают консольное окно после того, как программа завершила

своё выполнение. Если проблема в компиляторе, то следующие два шага

решат эту проблему:

Во-первых, добавьте следующую строчку кода в верхнюю часть вашей

программы:

#include <iostream>

Во-вторых, добавьте следующий код в конец функции main() (прямо перед

оператором return):

std::cin.clear();

std::cin.ignore(32767, '\n');

std::cin.get();

Таким образом программа будет ожидать нажатия клавиши, чтобы

закрыть консольное окно. Вы получите дополнительное время, чтобы

хорошенько всё рассмотреть/изучить. После нажатия любой клавиши,

консольное окно закроется.

Другие решения, такие как system("pause");, могут работать только на

определённых операционных системах, поэтому вариант выше

предпочтительнее.

Примечание: Visual Studio не задерживает консольное окно, если

выполнение запущено с отладкой ("Отладка" > "Начать отладку" или

F5). Если вы хотите, чтобы была пауза - воспользуйтесь решением выше

или запустите программу без отладки ("Отладка" > "Запуск без отладки"

или Ctrl+F5).

Проблема №3

При компиляции программы я получаю следующую ошибку:

"c:vcprojectstest.cpp(263) :fatal error C1010: unexpected end of file while

looking for precompiled header directive".

Ответ №3

Эта ошибка свидетельствует о том, что вы забыли подключить

предварительно скомпилированный заголовок pch.h. Для устранения этой

проблемы, вам нужно просто добавить следующую строчку кода в самый

верх программы:

#include "pch.h"

Обратите внимание, в программах с несколькими файлами, каждый файл

должен начинаться именно с этой строчки кода. В качестве альтернативы,

вы можете просто отключить использование предварительно

скомпилированных заголовков.

Проблема №4

При использовании cin, cout или endl компилятор говорит, что cin,

cout или endl являются "undeclared identifier" (необъявленными

идентификаторами).

Ответ №4

Во-первых, убедитесь, что у вас присутствует следующая строчка кода в

верхней части вашей программы:

#include <iostream>

Во-вторых, убедитесь, что cin, cout или endl имеют префикс “std::”.

Например:

std::cout << "Hello world!" << std::endl;

Проблема №5

Избавляемся от ошибки "This function or variable may be unsafe" в Visual Studio

Ответ №5

Компилятор в Visual Studio сильно отличается от привычных большинству программистов GCC или CLANG, из-за чего при написании кода на C или C++ очень часто возникают неожиданные проблемы в виде ошибки использования стандартных функций, например, scanf, fopen, sscanf и тому подобным. Студия предлагает заменять функции на безопасные (повезёт, если нужно просто добавить \_s к функции с ошибкой, но нередко в этих функциях идёт иной набор аргументов, нежели в обычной программе).

1. Выберем пункт "Проект" в верхнем меню

2. В открывшемся списке щёлкнем по "Свойства название\_проекта"

3. В появившемся окне выберем Свойства конфигурации, C/C++, Препроцессор

4. В строке Определения препроцессора допишем в самый конец строку \_CRT\_SECURE\_NO\_WARNINGS

5. Нажмём ОК