**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПИЩЕВЫХ ПРОИЗВОДСТВ»**

**(ФГБОУ ВО «МГУПП»)**

Лекция № 1: Программирование

Тема: «*Основы программирования*»

Содержание:

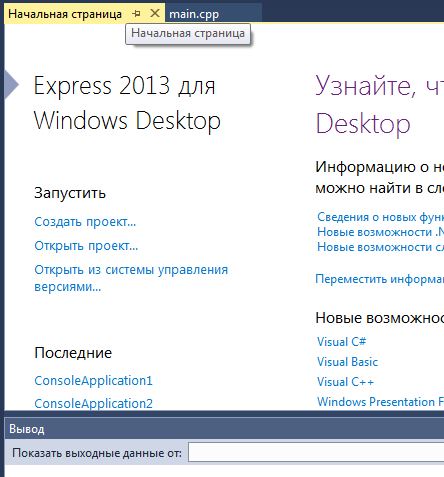
1. Написание первой программы на языке C++.
2. Типы данных, переменные и константы в C++.
3. Вывод данных на экран. Ввод данных с клавиатуры.
4. Арифметические операции.
5. Инкремент и декремент.
6. Логические операции.

Москва 2022

**Написание первой программы на языке С++**

К сожалению, компьютер не может нас понимать – он понимает только один язык – машинный код. Но так как программировать таким образом просто невозможно, было создано множество языков программирования. Одним из них является язык С++. Этот язык имеет определённое количество специальных зарезервированных логичных слов, с помощью которых в Microsoft Visual Studio (как и в любой другой среде разработки) можно писать команды для нашего компьютера. А далее уже с помощью встроенного компилятора, этот код будет преобразовываться в машинный код, который будет обработан, а нам будет показан результат работы программы.

Ну что же, приступим к написанию программы. Для этого вам надо открыть ваш созданный проект. Заходите в Microsoft Visual Studio. Перед вами открывается Начальная страница.

[](https://purecodecpp.com/wp-content/uploads/2014/05/Snimok.jpg)

Вот там, где написано Последние, должен находиться недавно созданный вами проект – жмите на него. Если вы еще не создавали проектов – создайте. Начинайте писать следующий код. Настоятельно рекомендую не копировать код, а набирать вручную.

|  |  |
| --- | --- |
|  |  |

Разберемся теперь, что мы написали. В 1-й и 2-й строках расположены комментарии к нашему коду. Весь текст строки, который расположен за двойным слэш –  **//**  – полностью игнорируется компилятором. Он его попросту не видит. Так мы можем в любом месте нашего кода оставлять комментарии – любую информацию, которую посчитаем нужной. Что-то вроде заметки, чтобы не забыть или обратить внимание.  Комментарии бывают и многострочными, так как если комментарий большой, бывает неудобно   перед началом каждой строки ставить двойной слэш.  Чтобы создать многострочный комментарий, надо весь текст или код, который необходимо игнорировать, поместить между **/\***   и    **\*/**

|  |  |
| --- | --- |
|  |  |

В строке 4 программы мы видим **#include <iostream>** – это директива препроцессору. Пока не будем вникать – просто пишите так в каждой новой программе.  То же касается и строки 5 – это, так называемое, подключение пространства имен **std** в котором хранятся слова-команды, которые мы будем использовать при написании каждой программы.

В строках 7 – 14 располагается главная функция: **main ( ) { /\*код программы\*/ }** При запуске программы она всегда выполняется первой и в ней между двумя фигурными скобками **{ }** находятся наши команды.  Вот запись в строке 9**–**функция **setlocale(LC\_ALL, “rus”);**– позволит корректно отображать на   экране знаки кириллицы. Прописывайте эту функцию в те программы, где вам будет необходимо выводить на экран русский текст.

Перемещаемся в строку 11 – тут мы ввели команду **cout** (зарезервированное слово из пространства имен **std**), которая отвечает за вывод данных на экран, оператор **<<** за которым следует текст. Текст обязательно надо помещать в кавычки. Так мы показываем компилятору, что это не команда для него, а обычная строка. В строке пишите любой текст. Когда текст написан, кавычки закрываются и снова пишем оператор **<<** за которым снова команда компилятору – **endl;**  (переход на следующую строку).  Точку с запятой **;** ставить обязательно. Она говорит компилятору, что команда закончилась и можно переходить к выполнению следующей команды.

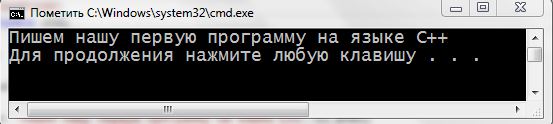
В конце блока функции **main()**видим команду **return 0;**  Она говорит компилятору, что по завершении программы надо вернуть значение 0. Тоже особо не задумывайтесь пока о том, что это значит.  Воспринимайте это, как некое правило – так надо писать в каждой программе. Любая наша программа в следующих уроках обязательно будет содержать такие записи:

|  |  |
| --- | --- |
|  |  |

Используйте её как шаблон, перед написанием новых программ.

Как же запустить нашу программу, чтобы полюбоваться на экране тем текстом, который был размещен в кавычках исходного кода? Для этого нажмите сочетание клавиш Ctrl + Shift + B – начнется компиляция программы.

Если вы допустили какие-то ошибки в коде, отладчик их обнаружит и сообщит вам об этом. Внизу окна вы увидите список этих ошибок. Их надо исправить.  Ну а если ошибок нет в нижней строке окна мы увидим Сборка: успешно: 1, с ошибками: 0 и т.д.  После этого нажимаем клавиши Ctrl + F5 и увидим в открывшемся окне сообщение то, что просили:

[](https://purecodecpp.com/wp-content/uploads/2014/05/444.jpg)

Программа отработала и выполнила наши указания нормально.

# Типы данных, переменные и константы в С++

Следующая тема, которая входит в основы программирования – это типы данных, переменные и константы. В этой части мы разберёмся с тем, что же такое типы данных и зачем они нам понадобятся в программировании, а также узнаем о переменных и константах.

***Типы данных.*** Сначала ответим “зачем” существуют типы данных. Смотрите, допустим нам надо написать программу, которая выводит на экран данные о возрасте, весе и росте человека. Но, чтобы программа могла обращаться к этим данным, их надо хранить где-то в оперативной памяти компьютера, а уже потом “попросить” показать их. Вот для этого нам и нужны типы данных – чтобы компилятор “понял” сколько памяти надо выделить для данных и что в этой памяти будет храниться (целое число (возраст), вещественное (вес и рост), символы и т.д.).

Типы данных можно разбить на три группы: числовые, символьные и логические. Для обозначения типов данных используются специальные зарезервированные (ключевые) слова.  Ниже мы их рассмотрим и укажем сколько памяти для них выделяется и какие данные они могут хранить.

Числовые типы данных: для хранения целых чисел (0, 33, -27 и т.д.)

**int** (4 байта) –   хранит   числа в диапазоне   от   -2 147 483 648 до 2 147 483 647

**short** (2 байта) – хранит числа в диапазоне от -32 768 до 32 767

**long** (4 байта) –   хранит   числа   в   диапазоне   от   -2 147 483 648 до 2 147 483 647

Числовые типы данных: для хранения вещественных чисел (с плавающей точкой: –435.332, 54.77, 3.0)

**float**(4 байта) – хранит дробные числа с точностью до 7 знаков после запятой

**double**(8 байт) – хранит дробные числа с точностью до 15 знаков после запятой

Символьный тип: для хранения одного символа

**char** (1 байт) – хранит один символ. Например: ‘f’, ‘+’ или ‘4’ (как символ). Одинарные кавычки обязательны.

Типа данных для хранения строк в С++ нет. Позже мы с вами познакомимся с тем, каким образом можно хранить строки в памяти.

Логический тип:

**bool**(1 байт) – может принимать только два значения **true** (истина) и **false** (ложь)

Сразу стоит обратить внимание, что все зарезервированные слова под типы данных пишутся не заглавными, а строчными (маленькими) буквами. Если вы объявите тип **bool**, как Bool – компилятор выдаст сообщение об ошибке. В редакторе, при написании кода, все зарезервированные слова С++ подсвечиваются синим цветом (это касается всех зарезервированных слов). Так что, если вы вписали тип, а слово не стало синим – проверьте правильность написания.

***Переменные и константы.***  Их названия говорят сами за себя. И переменная, и константа – это определённые области в оперативной памяти, которые имеют имя (имя задаёт программист). Только переменная может менять своё значение в ходе выполнения программы, константа же определяется один раз и изменяться уже не может.

Время, к примеру, это переменная величина. Сейчас, допустим, 3 часа дня, а через 2 часа уже будет 5 часов вечера.  А вот количество часов в сутках – это уже константа.

И переменным, и константам при объявлении обязательно надо присвоить имя. Существуют строгие правила, согласно которым даются имена:

* имя переменной может содержать маленькие и большие буквы английского алфавита, цифры, и ‘**\_’** (нижнее подчеркивание). С нижним подчеркиванием имена читаются намного легче. Сравните **amountofapples** и **amount\_of\_apples.**
* имя не может начинаться с цифры! **amount\_of\_apples1** – можно, 1**amount\_of\_apples**– не можно))
* именем не может быть зарезервированное слово (**int, bool** …). Если вы придумали имя переменной, ввели, а его подсветило синим цветом, значит такое слово в С++ является зарезервированным. Придумайте новое имя.
* имя переменной должно быть осмысленным (логичным тому, что будет хранить данная переменная). Т.е. вряд ли кто-то догадается, что в переменной **x** хранится значение количества яблок. Логичнее назвать такую переменную **amount\_of\_apples.**

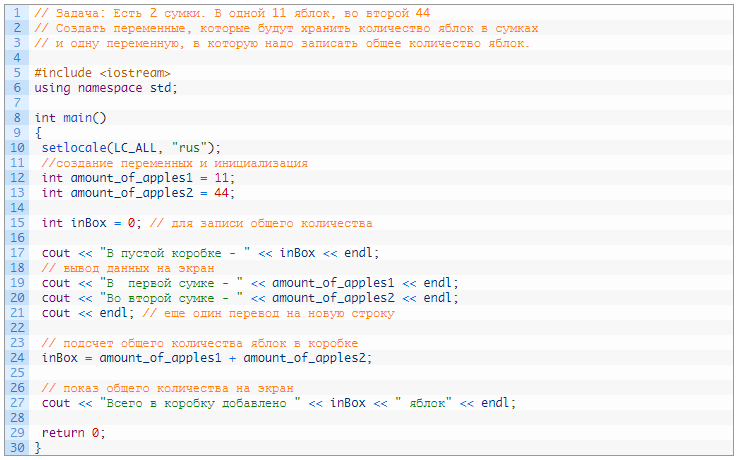
***Объявление и инициализация переменных и констант.***

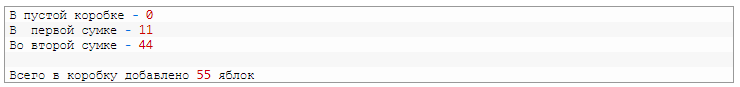
Как же можно создать переменную и указать какое значение она будет хранить? Для этого нам необходимо сначала указать тип\_данных, затем дать имя переменной (придерживаясь правил, рассмотренных выше), поставить знак **=** (означает присвоить) и указать значение. К примеру создадим переменную, которая будет хранить значение количества яблок: **int amount\_of\_apples = 7;**.

Присваивание значения переменной при её создании называется **инициализацией**. Желательно всегда инициализировать переменные при создании, даже если вы не знаете какое значение переменная примет в ходе работы программы. В этом случае инициализируйте её нулём: **int amount\_of\_apples = 0;**. Дело в том, что оперативная память свободной не бывает. И если вы просто создали переменную **int amount\_of\_apples;**и не присвоили ей никакого значения, она всё равно будет хранить какие-то остаточные данные от предыдущих программ. Поэтому, хотя и не является ошибкой сначала объявить переменную, а ниже присвоить ей значение, желательно присвоить значение этой переменно сразу, тем самым очистив её от “мусора”.  Кстати, в MVS 2013, если вы попытаетесь вывести на экран значение не инициализированной переменной, компилятор выдаст ошибку.

Что касается констант, то значение им должно быть присвоено сразу при создании. Например, определим константу, которая будет хранить количество дней в неделе: **const int** **daysInWeek = 7; чтобы** дать понять компилятору, что это константа, а не обычная переменная, перед типом данных обязательно использовать ключевое слово **const**.

Так, пожалуй, приступим к рассмотрению нескольких примеров, чтобы понять и запомнить то, о чём мы тут говорили. Не забывайте, набирать код – практикуйтесь.

В строке 15 переменной **inBox** присвоено значение **0**, так как мы не знаем какое значение туда запишется.  И в строке 17, мы показываем начальное значение на экран.  Как видите, чтобы показать значение переменной на экран, достаточно обратиться к ней по имени: **cout << inBox;**  Обратите внимание, как работает **cout**. С помощью оператора **<<** , мы можем чередовать показ текста и показ значения переменной. Так мы делали в строках 17, 19, 20 и 27. В строке 24 в переменную inBox записывается сумма переменных: **amount\_of\_apples1 + amount\_of\_apples2**.  То есть, компилятор, сначала сложит значения этих переменных, а потом запишет сумму в **inBox** . И чтобы убедиться, что всё посчиталось и записалось – снова выводим значение **inBox** на экран – строка 27. Запускаем программу (сначала Ctrl + Shift + B, если ошибок нет то далее Ctrl + F5 подробней [здесь](https://purecodecpp.com/archives/123)). Вот что мы увидим:

Выделим основное, что необходимо запомнить:

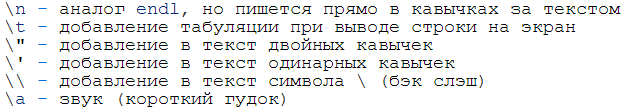
* + тип данных указывается перед именем переменной и определяет какие данные в ней будут храниться (число, символ… ) и сколько памяти необходимо под них выделить.
  + имя переменной дает программист, соблюдая определённые правила (указаны выше).
  + чтобы создать (объявить) переменную надо указать её тип и дать имя. Переменную желательно сразу инициализировать (присвоить значение при создании): **тип имя = значение;**
  + чтобы объявить константу необходимо использовать ключевое слово **const**и обязательно сразу присвоить значение: const**тип имя = значение;**
  + переменным можно присвоить не только определённое значение, а и результат какого-то вычисления: **amount\_of\_apples1 = amount\_of\_apples2 + 33;**
  + объявлять переменные желательно в начале main-функции.А при необходимости еще и оставить комментарий, что они будут хранить.
  + регистр букв имени имеет значение. Имена Apple и apple обозначают разные переменные.

# Вывод данных на экран и ввод данных с клавиатуры

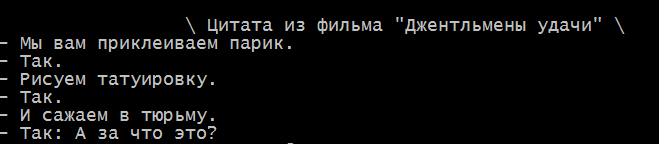
Сначала поговорим о выводе данных на монитор, так как вы, отчасти, уже с ним сталкивались ранее. Как вы помните, при помощи ключевого слова **cout,** мы можем вывести на монитор строку, которую пишем в “кавычках”, либо значение переменной, обратившись к ней по имени. Так же вы знакомы с оператором **endl**, который позволяет перевести вывод на следующую строку.  И еще один важный момент – мы можем комбинировать вывод данных используя оператор **<<** несколько раз после **cout**:

**cout << “Значение переменной var\_name  =  ” <<  var\_name  <<  endl;**

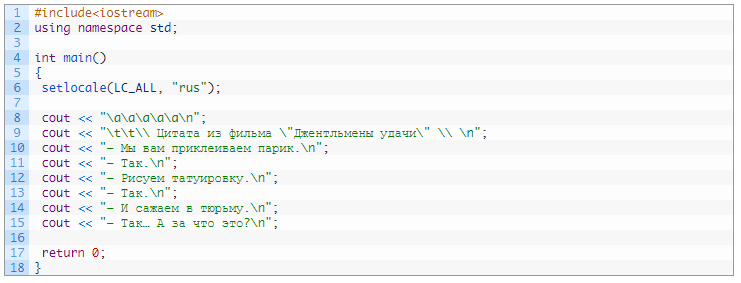
Теперь мы познакомимся еще с несколькими специальными символьными последовательностями, которые помогут нам манипулировать выводом данных на экран. Вот перечень:

[](https://purecodecpp.com/wp-content/uploads/2014/05/vyivod-dannyih.png)рис.1 – символьные последовательности, вывод данных

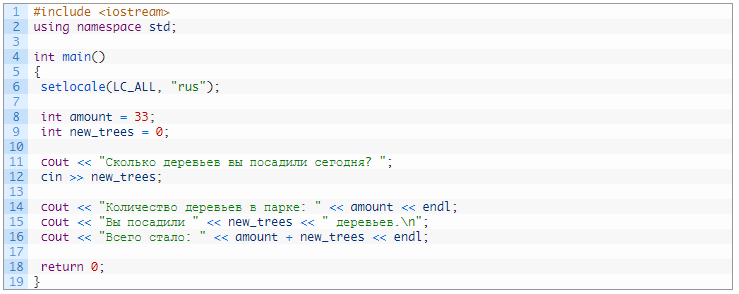
Зачем использовать обратный слэш? Рассмотрим на примере. Нам надо вывести на экран цитату из фильма. Сразу пусть прозвучит сигнал, который привлечет внимание пользователя на экран. По центру разместим заголовок и название фильма, из которого цитируем, а ниже – цитату. На экране должно выглядеть так:

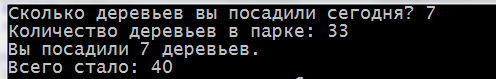
[](https://purecodecpp.com/wp-content/uploads/2014/05/vyivod-dannyih1.png)

Постарайтесь набрать код самостоятельно! Решение ниже:

Зачем нам нужен обратный слэш? Он говорит компилятору: “Внимание, за мной стоит не простой символ, а специальный! Он скажет тебе, что нужно сделать! :)”  Вот в строке 8, если бы мы не использовали с вами обратный слэш, на экран бы пошёл вывод пяти символов а. А так, мы услышим “прекрасную музыку”.  В строке 9 тоже интересно: две табуляции, далее надо вывести на экран один бэк слэш, а фактически надо написать два, ну и берём в кавычки название фильма. В конце каждой строки вместо манипулятора потока **endl**(**<< endl**) который позволяет  перейти на новую строку (фактически аналог кнопки Enter на  вашей клавиатуре) удобнее и короче, в данном случае, использовать управляющий  символ **\n.**

Теперь поговорим о вводе данных пользователем с клавиатуры. Мы уже знаем – чтобы присвоить значение переменной, её можно инициализировать при создании либо присвоить значение ниже по ходу программы: **имяПеременной = значение;**А сейчас мы научимся записывать в переменную значение, которое вводит пользователь с клавиатуры. Мы можем организовать ввод данных используя операторы **cin** и>**>**. Синтаксис, следующий: cin**>> имяПеременной**.  Рассмотрим эту возможность на примере:

Запустите программу и обратите внимание – программа выполнит команду 11-й строки и, дойдя до оператора cin**>>**, остановится и начнется ожидание действия от пользователя. Надо ввести значение и нажать **Enter**. Как только переменная получит значение, введенное с клавиатуры, программа продолжит выполнение.

[](https://purecodecpp.com/wp-content/uploads/2014/05/vvod-dannyih.png)

Стоит отметить, что **cin**понимает и различает типы переменных. И если ввести в **int** символ, а не цифру, значение переменной не изменится. А если ввести несоответствующее значение в переменную, которая не инициализирована при создании, на экран будет показан какой-то остаточный “мусор” из переменной. Чтобы вводить символы – переменные надо объявлять типа **char**.

**Арифметические операции в С++**

В современной жизни очень сложно обойтись без арифметических операций. Нам постоянно приходится что-то считать: складывать, умножать, вычитать, делить и т.д. Программирование – не исключение. Вам в 99.9% случаев придется ими пользоваться, при написании своих программ. Бояться их не стоит – все арифметические операции – просты, понятны и знакомы нам со школы.

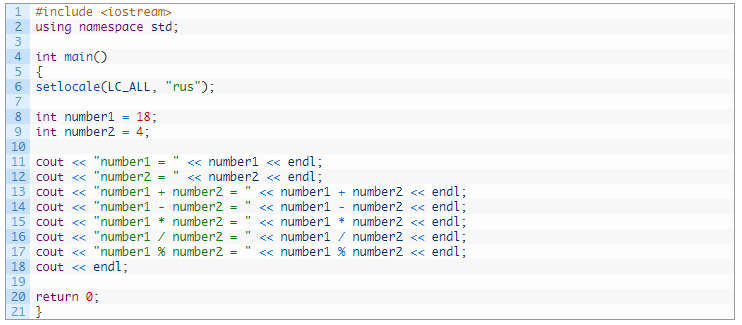
Рассмотрим арифметические операции в следующей таблице.

[](https://purecodecpp.com/wp-content/uploads/2014/06/arifmeticheskie-operacii5.jpg)Арифметические операции в С++ – рис. 1

Тут особое внимание следует уделить делению по модулю (%). Эта операция достаточно часто используется в решении определённых задач. Пример её применения: если нам необходимо поделить по модулю 9 на 4 (9 % 4), результат будет равен 1 (это остаток – то, что на 4 уже не делится на цело). Еще примеры: 20 % 8 = 4 (8 помещается в 20-ти 2 раза: 8 \* 2 = 16, 20 – 16 = 4 остаток от деления), 3 % 2 = 1, 99 % 10 = 9, 9 % 10 = 9. Важно:

* деление по модулю применяется только к целочисленным переменным;
* нельзя делить по модулю на 0;

Пример:

Результат компиляции:  
[](https://purecodecpp.com/wp-content/uploads/2014/06/arifmeticheskie_operacii11.jpg)

Тут вы видите, что при делении **num1** на **num2**, на экране появилась только целая часть – 4 (хотя точное значение 4.5). Дробная часть отсекается, так как переменные определены, как целочисленные – **int**. А в результате деления по модулю мы видим 2 – то, что осталось в остатке от деления 18 на 4.

Еще что хотелось бы рассмотреть в данной статье – это так называемые **комбинированные (или составные) операторы**. Помимо выполнения своей арифметической роли, они одновременно выполняют роль присваивания значения переменным. Вот список таких составных операторов:

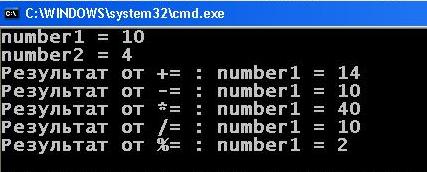
[](https://purecodecpp.com/wp-content/uploads/2014/06/arifmeticheskie-operacii4.jpg)Арифметические операции в С++ – рис. 2

Покажу на примере:



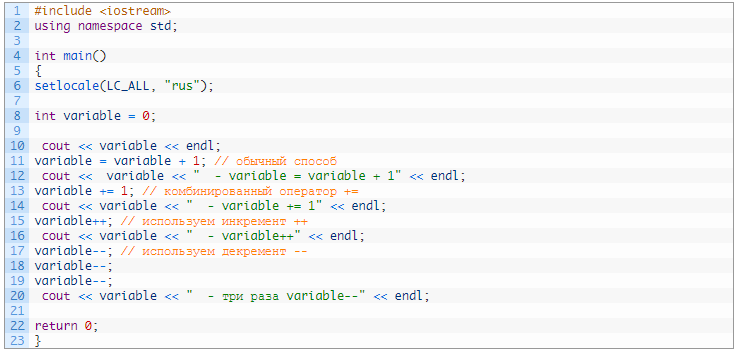
Хотя для кого-то, с первого взгляда, эти операторы могут показаться запутанными, поверьте, к ним очень быстро можно привыкнуть и успешно применять в своих программах. Надо только немного попрактиковаться с их применением. Ваш код при этом будет выглядеть компактнее. Так же, использование комбинированных операторов является признаком хорошего тона в программировании. Поэтому, хоть и не будет ошибкой в коде использование сложения отдельно от присваивания – **number1 = number1 + number2;**, предпочтительней использовать сокращенный вариант записи **number1 += number2;**

Результат:

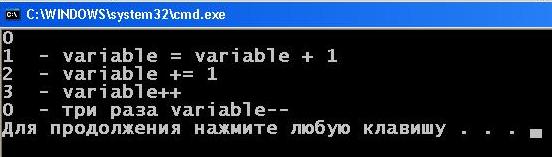
[](https://purecodecpp.com/wp-content/uploads/2014/06/arifmeticheskie_operacii2.jpg)

# Инкремент и декремент в С++

В этой теме мы научимся применять унарные операторы, для которых необходим один операнд. Называются они **инкремент** (+**+**) и **декремент** (**––**). Роль этих операторов в том, чтобы изменить (увеличить или уменьшить соответственно) значение переменной на единицу, при этом значение будет перезаписано. Рассмотрим пример, в котором будем изменять значение переменной **variable** на единицу тремя различными способами:

После каждой операции в строках 11, 13 и 15. к значению переменной **variable** прибавляется единица. Как вы видите, самая короткая запись – это запись с использованием инкремента. Ниже, в строках 17 – 19, мы трижды применили декремент и в итоге получим значение **variable** уменьшенное на 3.

Компилируем:

[](https://purecodecpp.com/wp-content/uploads/2014/06/inkrement-dekrement-2.jpg)

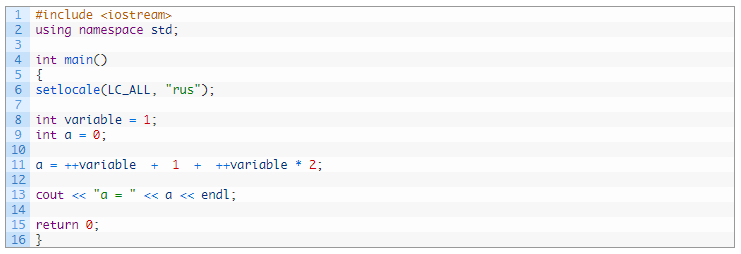
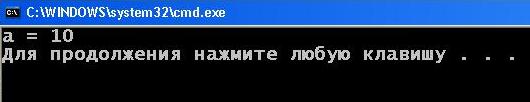
Оба этих унарных оператора могут иметь две формы: **постфиксную** (когда оператор расположен сразу за переменной) и **префиксную** (оператор расположен перед переменной). В рассмотренном примере мы применяли постфиксные формы инкремента и декремента. Чем же отличаются эти формы друг от друга? И первая и вторая формы этих операторов выполняют одинаковые роли – увеличение или уменьшение переменной на единицу. Но если в строке кода, где применяются инкремент или декремент, присутствуют еще какое-либо команды, тогда эти операторы по-разному себя ведут. Постфиксная форма будет применена после выполнения других команд. Посмотрим на следующем примере:

В результате выполнения кода строки 12, мы увидим на экране число 0. Так произойдет потому, что увеличение на единицу выполнится после вывода значения переменной **variable** на экран. Чтобы убедиться, что значение действительно изменилось – в строке 13 просим показать нам **variable**. А вот в случае применения префиксной формы инкремента (строка 16), переменная сразу перезапишется и уже только после этого произойдет вывод на экран. Для декремента всё то же самое, но с уменьшением на единицу.

Компилируем:

[](https://purecodecpp.com/wp-content/uploads/2014/06/inkrement-dekrement-3.jpg)

Вот посмотрите еще на этот пример:

Результат:  
[](https://purecodecpp.com/wp-content/uploads/2014/06/inkrement-dekrement-4.jpg)

Постарайтесь понять, почему в результате операций строки 11, переменная **a** стала равна 10.

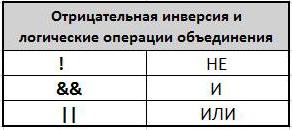
# Логические операции в С++

В этой части лекции будем разбираться с логическими операциями языка С++.  При сравнении между собой каких-либо переменных (значений), мы получаем результат – ложь (**false**) или истина (**true**). Например, мы сравниваем два значения: 3 < 8 – это истина, так как 3 действительно меньше 8.  А вот 10 > 50 – это ложь.

Рассмотрим операторы, которые применяются для логических операций:

[](https://purecodecpp.com/wp-content/uploads/2014/06/logicheskie-peracii.jpg)Логические операции – таб. 1

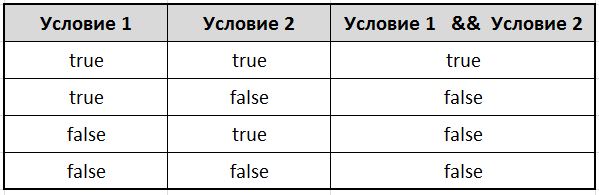
Все операторы из **таблицы 1 –**иоператоры сравнения и операторы равенства **–**являются бинарными. В программировании работа с ними сводится к тому, что мы сравниваем два операнда (два значения) между собой и видим их отношение друг к другу: один больше, а второй меньше, или наоборот, или один абсолютно равен второму, или не равен…   В результате этой проверки, как говорилось выше, в программу возвращается одно из двух возможных значений –   **false** (0) или **true** (1). Кстати, к истине в С++ относится любое значение, как положительное, так и отрицательное, кроме 0. **Важно** – не путайте операторы **=** (присваивание значения переменной) и **==** (логический оператор равенства). Это два абсолютно разных оператора.

[](https://purecodecpp.com/wp-content/uploads/2014/06/logicheskie-peracii2.jpg)

Логические операции -таб. 2

Логические операции объединения &**&** и   **||** также имеют бинарную форму. Они используются в программировании для того, чтобы можно было не только сравнивать два значения, а и создавать некие условия или объединять несколько условий сравнения или равенства в одно. Например, у нас есть переменная **variable** значение которой вводит пользователь. А мы хотим узнать – находится ли оно в диапазоне от 10-ти до 20-ти? Это так (**true**) или не так (**false**)? Проверим, используя логическое И (**&&**): **cout <<  ((variable > 10) && (variable < 20));** То есть, на экран покажется 1(**true**), только в том случае, если значение будет одновременно больше 10 И(**&&**)  меньше 20. Если пользователь введет 15 – увидим на экране 1. Если введет 21 – на экране 0, так как 21 уже вне диапазона.

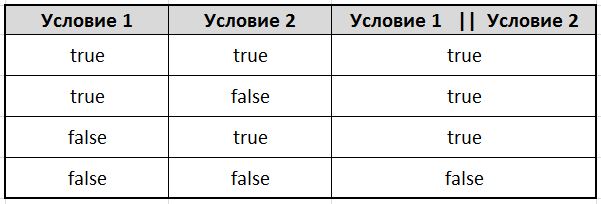
Результаты, которые возвращает комбинированное условие с логическим И (**&&**), можно представить в таблице:

[](https://purecodecpp.com/wp-content/uploads/2014/06/logicheskie-operacii-7.jpg)Логические операции – таб. 3

Применительно, к примеру рассмотренному выше: 15 одновременно и >10 и <20. Значит выполняются и Условие 1 и Условие 2. Итог – **true**. Если бы пользователь ввел значение 50, то выполняется и дает **true** только Условие 1 (>10), а Условие 2 (<20) – это уже **false**, так как оно не выполняется. Итог комбинированного условия –   **false**.

А что, если нам необходимо наоборот подтвердить, что введенное число не принадлежит заданному диапазону, а находится в диапазонах либо <10, либо >20? Тогда нам придется составить комбинированное   условие, используя логический оператор ИЛИ ( **||** ): **cout <<  ((variable < 10) || (variable > 20));** Тогда  если пользователь ведет 15 – увидим на экране 0, так как 15  не входит ни в диапазон от 9 до -∞, ни в диапазон от 21 до +∞. А если введет 22 – на экране увидим 1, так как 22 попадает под второе условие и входит в диапазон от 21 до +∞.

Результаты, которые возвращает комбинированное условие с логическим ИЛИ ( **||** ), можно так же представить в таблице:

[](https://purecodecpp.com/wp-content/uploads/2014/06/logicheskie-operacii-88.jpg)Логические операции – таб. 4

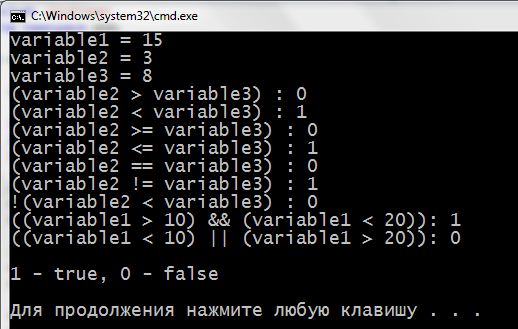
Что касается отрицательной инверсии – логическое **НЕ ( ! )** – этот оператор является унарным.  Он всегда “перекручивает” возвращаемый результат условия. Например, при выводе на экран результата сравнения **cout << (3 < 8);** – увидим 1, так как это истина. А используя перед условием логическое НЕ:  **cout << !(3 < 8);** истина условия (3 < 8) преобразуется в ложь. В таблице это выглядит так:

[](https://purecodecpp.com/wp-content/uploads/2014/06/logicheskie-operacii-9.jpg)

Логические операции – таб. 5

Соберем все рассмотренное в один код:

Результат:

[](https://purecodecpp.com/wp-content/uploads/2014/06/logicheskie-operacii-10.jpg)