

Это учебное пособие создавалась с огромным желанием помочь педагогам и начинающим программистам в изучении основ языка программирования С++. А главное, как можно понятней и короче осветить основные темы с использованием практических примеров. Поэтому вы не найдете здесь историю программирования С++. Хотя, об этом тоже нужно знать. Упор сделан на то, чтобы дать больше ценной информации - что, где, как и зачем надо писать, чтобы программа выполняла ту задачу, которая перед вами стоит.

# **Введение**

Условно создание компьютерных программ делят на этапы, которые повторяются в случае, если результаты не соответствуют условиям задачи.

Давайте коротко рассмотрим их:

1. **Постановка задачи**

Необходимо на основе полученного условия задачи провести его детальный анализ - определить (уточнить) какие будут использоваться исходные данные, их источники, их допустимые и недопустимые значения. Сформулировать какие данные должны быть на выходе, в том числе в каком виде представлены, на каких устройствах должны быть отображены. Если в условии задачи есть какие-либо неоднозначные формулировки, их обязательно нужно уточнить.

1. **Математическое моделирование**

Подбирается необходимой математический аппарат, при решении сложных задач требуется построить необходимые математические модели, которые описываются на языке математики и логики.

1. **Построение алгоритма**

**Алгоритм** - это определенная последовательность действий, которая позволяет на основе входных данных получить необходимые выходные данные, т.е. решить поставленную задачу.

1. **Программирование**

Оно предполагает, что раннее составленный алгоритм необходимо записать на языке программирования.

**Язык программирования** представляет собой формальную знаковую систему, при помощи которой записывается код.

**Система (среда) программирования** - это совокупность программных средств, которые настроены на совместную работу и применяются программистом для разработки компьютерных программ.

Как правило, любая среда программирования имеет набор программных средств:

* Текстовый редактор;
* Программа-транслятор;
* Программа-отладчик.

1. **Ввод данных**
2. **Отладка и тестирование**

**Отладка** – устранение ошибок как синтаксических, так и логических.

**Тестирование** – проверка на правильность выполнения всех запрограммированных задач. Для простых достаточно приготовить набор входных/выходных данных (выходные вычисляются вручную). Для сложных программных систем разрабатываются test-case (папка, документ).

1. **Анализ результатов**

На этом этапе выявляются соответствие полученных данных, выдаваемых программой с теми, которые планировались на этапе «постановки задачи», если не соответствует, то все сначала.

# **Первая программа**

В качестве компилятора советую использовать: [Online C++ Compiler - online editor (onlinegdb.com)](https://www.onlinegdb.com/online_c++_compiler).

Напишите следующий код сами. Пишите вручную!

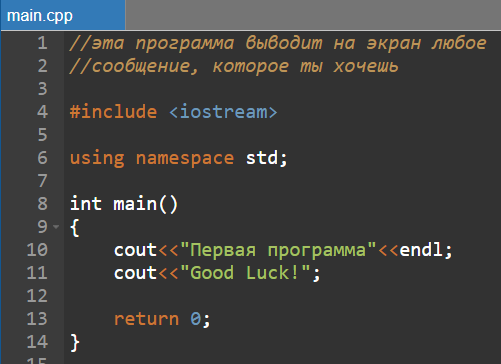


Рисунок 1 - Код программы

В 1-й и 2-й строках расположены комментарии к нашему коду. Весь текст, который расположен за двойным слэш – **//** – полностью игнорируется компилятором. Он его попросту не видит. Так мы можем в любом месте нашего кода оставлять комментарии. Это похоже на заметку. Комментарии бывают и многострочными. 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 препроцессора. Директива - это библиотека. В данном случае это библиотека ввода/вывода, т.е. отвечает за вывод на экран монитора и за ввод с клавиатуры пользователем.

В строке 6 – это, так называемое, подключение пространства имен **std** в котором хранятся слова-команды, которые мы будем использовать при написании каждой программы.

В строках 8 – 14 располагается главная функция: **main ( )**. После открываются фигурные скобки, где находится блок операторов или “тело” программы - **{ /\*код программы\*/ }**.

Перемещаемся в 10-11 строки – тут мы ввели команду **cout** (зарезервированное слово из пространства имен **std, consoleOut -** консоль “на выход”), которая отвечает за вывод данных на экран, оператор **<<** за которым следует текст. Текст необходимо помещать в кавычки для того, чтобы компилятор понимал, что это не команда, а обычная строка. Когда текст написан, кавычки закрываются и снова пишем оператор **<<** за которым идет 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.

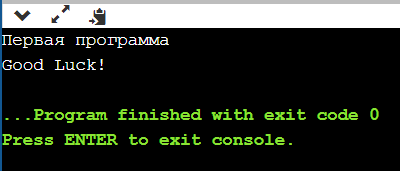
В 13 строке команда **return 0;** Она говорит компилятору, что по завершении программы надо вернуть значение 0. Особо не задумывайтесь пока о том, что это значит. Воспринимайте это, как некое правило, которое надо писать в каждой программе.

Для того, чтобы запустить программу необходимо нажать F9 или кнопку Run.



Рисунок 2 - Запуск программы

Результат программы вы увидите внизу:

  
Рисунок 3 - Результат 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.

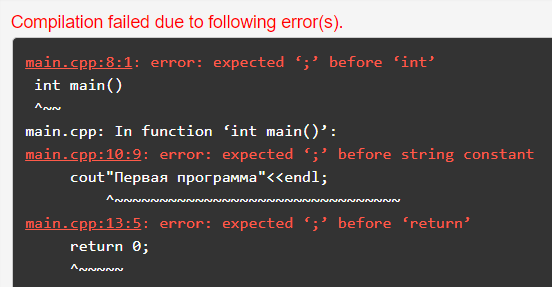


Рисунок 4 - Пример вывода ошибок

# **Типы данных, переменные и константы в С++**

**Типы данных.** Зачем же они нам нужны?! Допустим нам надо написать программу, которая выводит на экран данные о возрасте, весе и росте человека. Т. е. возраст — это целое число, вес и рост — это вещественное. Чтобы было более понятно рассмотрим это в программе ниже.

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

Таблица 1 - Типы данных

|  |  |  |
| --- | --- | --- |
| 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: ‘s’, ‘@’ или ‘7’ (как символ). Одинарные кавычки обязательны. |
| Логический тип: | | |
| **bool** | 1 байт | может принимать только два значения **true** (истина) и **false** (ложь) |

**Переменные и константы.** И переменная, и константа – это определённые области в оперативной памяти, которые имеют имя. Переменная может менять своё значение в ходе выполнения программы, а константа — это неизменяемая переменная, которая определяется один раз и изменяться уже не может.

Имена переменных необходимо задавать в соответствии с определенными правилами:

* Может содержать большие и маленькие буквы английского алфавита, цифры и нижние подчеркивания: **My\_Box**.
* Переменная не может начинаться с цифры, только так: **Number\_1**.
* Старайтесь задавать переменной логический смысл, например, переменная, отвечающая за коробку назвать **Box**, а не **b**.

**Объявление и инициализация переменных и констант.** Сначала указывают **тип\_данных**, затем задают **имя переменной.** Поставить знак = означает присвоить значение. К примеру, создадим переменную, которая будет хранить значение количества коробок: **int amount\_of\_box = 3;**.

Инициализация — это присваивание значения переменной. Желательно инициализировать переменные в самом начале. Если не знаете ее будущее значение, то просто укажите **0**: **int box = 0;.**

У констант **(const)** значение должно быть присвоено сразу при создании: **const int box = 2;**.

Пишем программу:

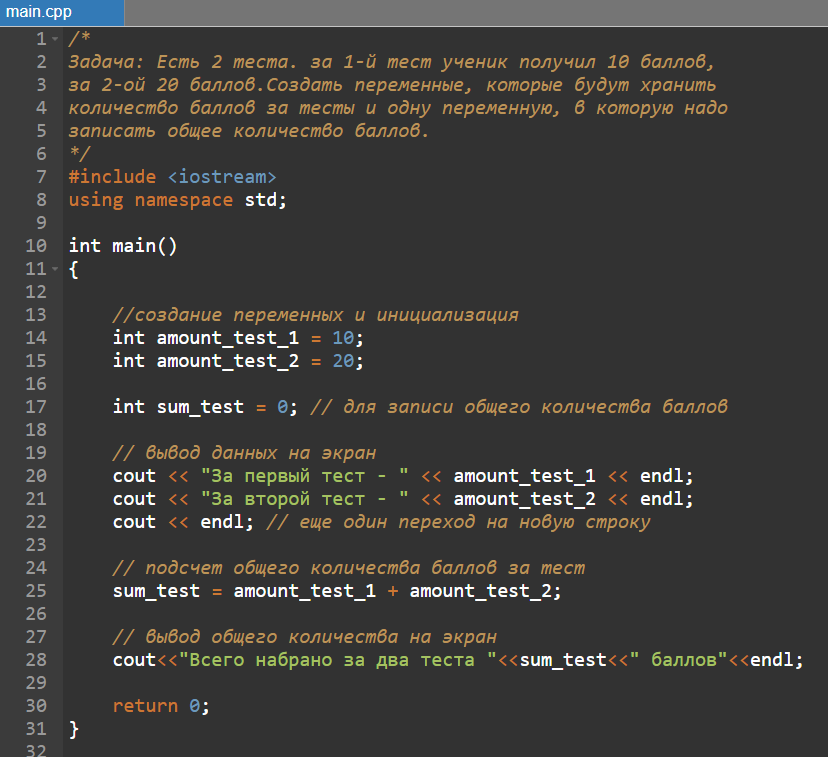


Рисунок 5 - Код программы

Результат программы:

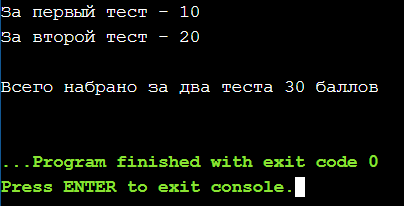


Рисунок 6 - Результат программы

# **Запомни!**

* Сначала **тип\_данных**, а потом **имя\_переменной**;
* **Имя\_переменной** задает программист
* Инициализировать переменную, т.е. задавать ей значение, желательно сразу: **тип\_данных имя\_переменной = значение;**;
* Константы объявляются следующим образом: **const тип\_данных имя\_переменной = значение;**;
* Переменная может также содержать результат вычисления: **sum\_box = box + 3;**;
* Регистр букв очень важен в объявлении переменных. Переменная **box** и **Box** это две разные переменные!
* Значения вещественных переменных пишутся через **точку**.

# **Задачки**

1. Поход в магазин. Объявить четыре переменные, с помощью которых можно будет посчитать общую сумму покупки трёх товаров. Например: молока, сметаны и сока.  
   Цены товаров за штуку: молоко - 43.90; сметана - 65.70; сок - 51.99.
2. Объявить две переменные типа **int** и одну переменную типа **double**. Первым двум задать любые целые значения, а третьей - 1.5. Вывести на экран: первую переменную (просто ее значение); 2-ю переменную, умноженную на 3-ю и 3-ю переменную, поделенную на 10. В итоге должно получиться три вывода на экран.

# **Вывод данных на экран и ввод данных с клавиатуры**

**Вспомним**

Команда **cout<<** отвечает за вывод информации на экран. Чтобы выполнить переход на новую строку, необходимо дописать в конце строки **<<endl;.** Форма вывода текста и переменной на экран выглядит следующим образом:

**cout <<”Значение переменной box = “ << box << endl;**

Также есть еще несколько специальных символьных последовательностей, которые помогают разнообразить вывод информации на экран монитора:

\n - аналог **endl**, только пишется в кавычках (“\nТекст\n”)

\t - табуляция при выводе текста на экран (горизонтальная)

\v – вертикальная табуляция

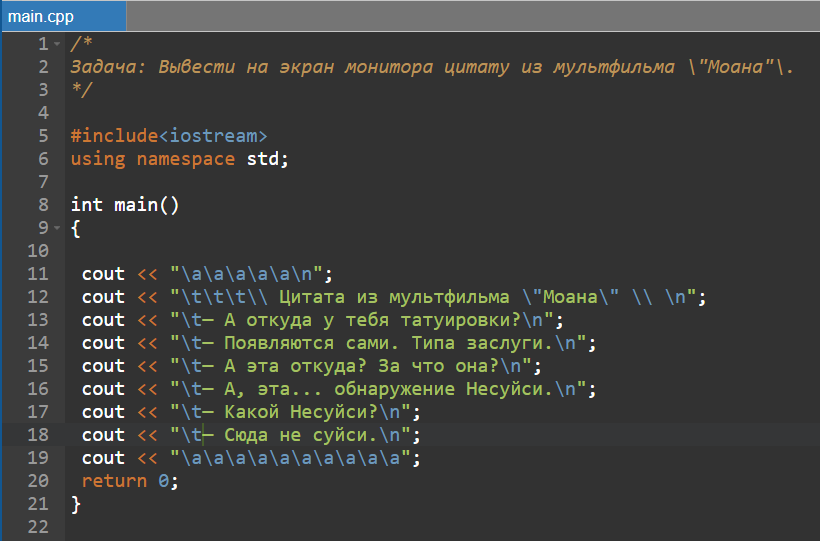
\” - добавление в текст двойных кавычек

\’ - добавление в текст одинарных кавычек

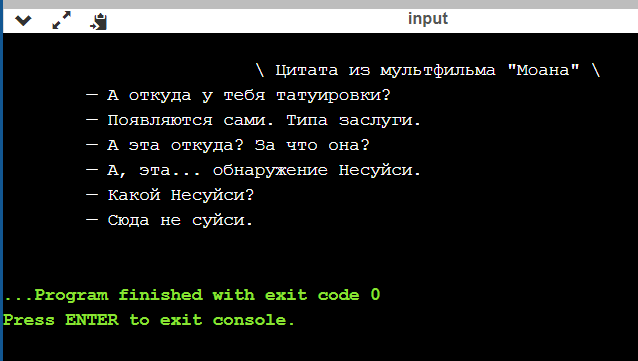
\\ - добавление в текст бэк-слэша (\)

\a - короткий звук

Рассмотрим на примере:

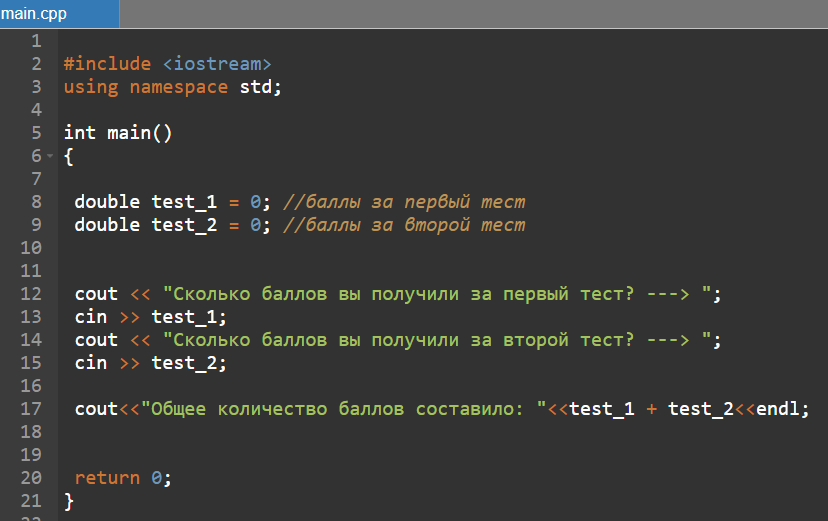
  
Рисунок 7 - Код программы

Результат выполнения программы:

  
Рисунок 8 - Результат программы

Ранее, при инициализации переменной, мы присваивали ей значение сами, а что, если пользователь должен ввести значение переменной?! Тогда организовывать ввод данных нужно при помощи команды **cin**. Выглядит это следующим образом: **cin >> имя\_переменной**.

Рассмотрим на примере:

  
Рисунок 9 - Код программы

После ввода переменной необходимо нажимать **Enter**, после этого автоматически продолжится выполнение программы. Команда **endl** в **cin** не нужна!

Результат программы:

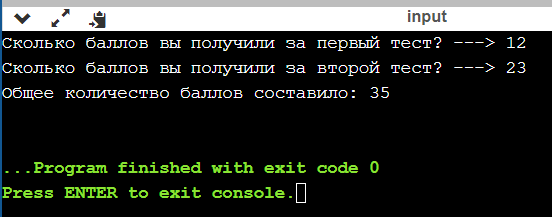


Рисунок 10 - Результат программы

* **Задачки**  
    
  1. Задайте три переменные с разными типами данных (int, double, char) и предложите пользователю ввести в них значения. После ввода данных, выведите их на экран монитора.  
    
  2. Выведите стихотворение М.Ю. Лермонтова “Парус” на экран монитора. Название стих-я посередине основного текста, между четверостишиями делать разрыв (две команды endl), внизу справа добавить автора.  
    
  3. Пользователь должен ввести 2 числа. Вам надо показать на экран произведение этих чисел, сумму и среднее арифметическое введенных чисел.

# **Арифметические операции в С++**

В повседневной жизни мы постоянно используем арифметику. Программирование также не обходится без арифметики. Арифметика в программировании такая же, как и в школьной математике.

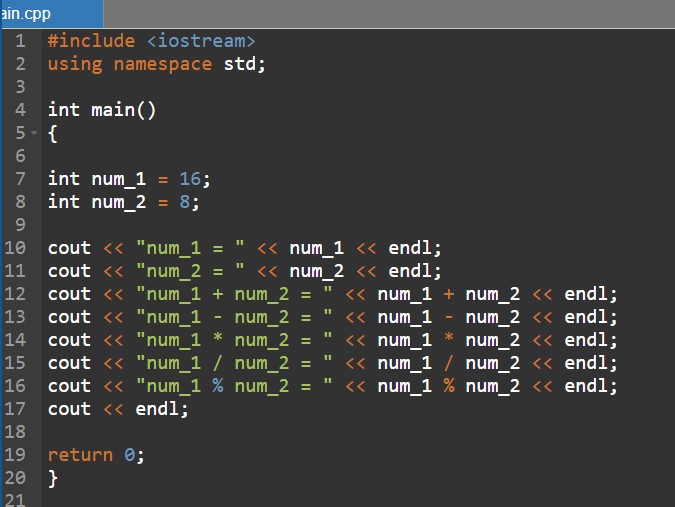
Рассмотрим арифметические операции:

Таблица 2 - Арифметические операции

|  |  |
| --- | --- |
| Оператор | Операция |
| **+** | Сложение данных |
| **-** | Вычитание данных |
| **\*** | Умножение данных |
| **/** | Деление данных |
| **%** | Деление данных по модулю |

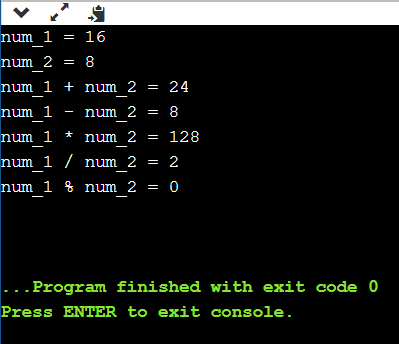
И тут сразу возникает вопрос: как это по модулю? Легко, вот вам пример: если нам необходимо поделить по модулю 9 на 4 (9 % 4), результат будет равен 1 (это остаток – то, что на 4 уже не делится на цело). Еще примеры: 20 % 8 = 4 (8 помещается в 20-ти 2 раза: 8 \* 2 = 16, 20 – 16 = 4 остаток от деления). По модулю делить на 0 нельзя!

Рассмотрим пример программы:

  
Рисунок 11 - Код программы

Мы указали целый тип int, поэтому в результате программы будут выведены только целые числа.

Рассмотрим пример программы:

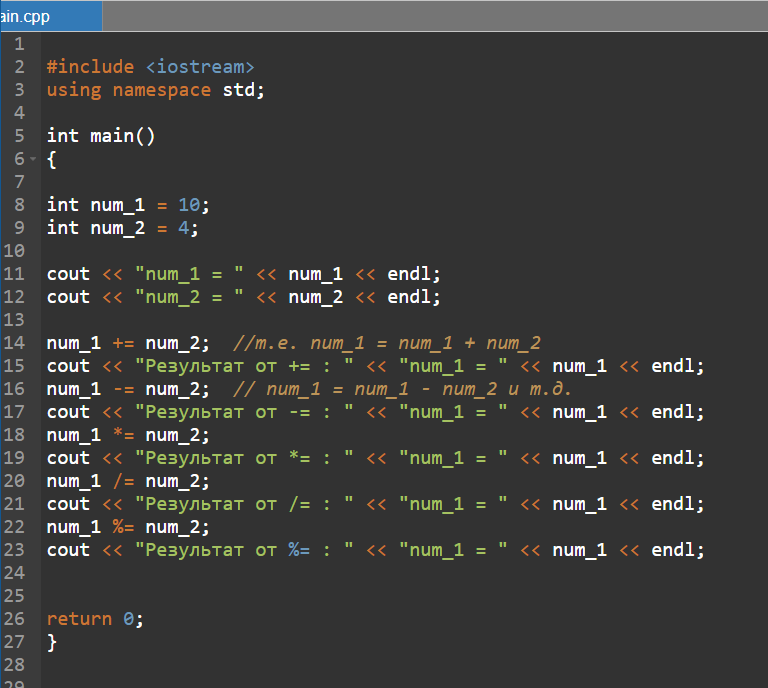
   
Рисунок 12 - Результат программы

Помимо обычных операторов есть еще комбинированные операторы, которые одновременно выполняют роль присваивания значений переменных. Все эти операторы называются **бинарными**.

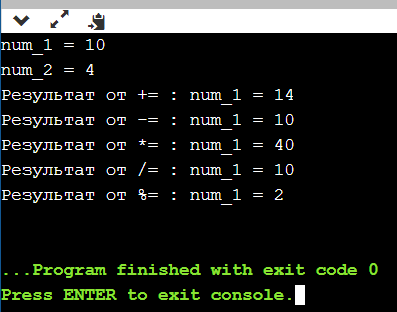
Таблица 3 - Комбинированные операторы

|  |  |
| --- | --- |
| Комбинированный оператор | Операция |
| **+=** | Сложение данных с присваиванием |
| **-=** | Вычитание данных с присваиванием |
| **\*=** | Умножение данных с присваиванием |
| **/=** | Деление данных с присваиванием |
| **%=** | Деление данных по модулю с присваиванием |

Рассмотрим пример:

   
Рисунок 13 - Код программы

Результат:

   
Рисунок 14 - Результат программы

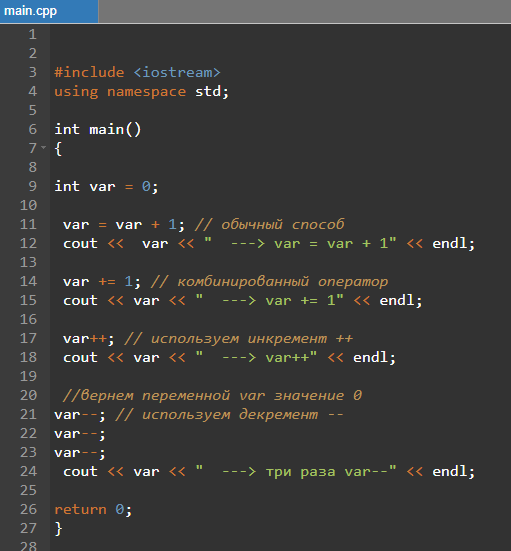
* **Задачки**

1. Напишите программу-калькулятор на нахождение суммы, например: **y = a + b;**.
2. Распространённая задача: Дано четырехзначное число (к примеру, 3456), вывести на экран в обратном порядке цифры, из которых это число состоит. То есть мы должны увидеть на экране 6543. Подсказка: чтобы взять из числа отдельные цифры, надо применять деление по модулю на 10.
3. Напишите программу для уравнения:  
    .

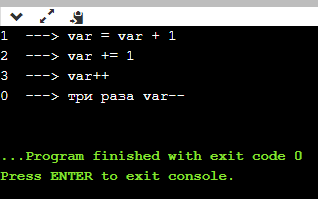
# **Инкремент и декремент в С++**

Помимо бинарных операторов, которые мы с вами рассматривали раннее, существуют и унарные операторы. Называются они инкремент (++) и декремент (--). Их роль состоит в том, чтобы увеличить или уменьшить значение переменной на единицу, при этом перезаписав значение. Унарные операторы используют всего одну переменную (операнд).

Рассмотрим пример, в котором будет изменяться значение переменной **var** тремя различными способами:

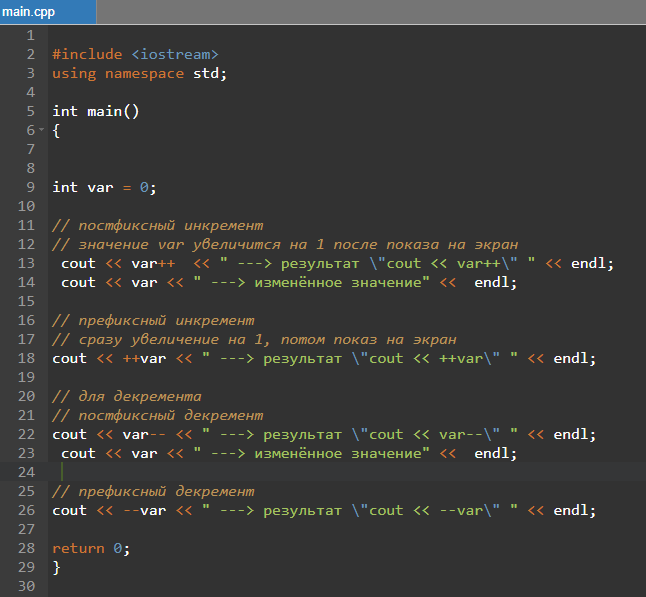
  
Рисунок 15 – Код программы

Результат программы:

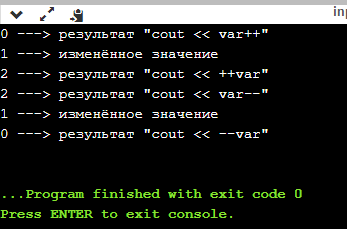
  
Рисунок 16 – Результат программы

Унарные операторы могут иметь **постфиксную** (когда оператор расположен сразу за переменной) и **префиксную** (оператор расположен перед переменной) форму. Эти формы 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.

Рассмотрим на следующий пример:

  
Рисунок 17 – Код программы

Результат программы:

  
Рисунок 18 – Результат программы

**Логические операции в С++**

В прошлых темах мы с вами разбирали арифметические операции, комбинированные, унарные операторы. Но кроме вычислений, также нужно уметь и сравнивать значения переменных друг с другом, для этого нам необходимы логические операции.

Когда мы сравниваем между собой какие-либо значения переменных, то получаем результат – истина(true\1) или ложь (false\0).

Рассмотрим операторы, которые используются в логических операциях:

|  |  |
| --- | --- |
| ***Операторы сравнения*** | |
| **Оператор** | **Пример и результат** |
| **>** (больше) | cout << (2>6);  0\false |
| **<** (меньше) | cout << (2<6);  1\true |
| **>=** (больше или равно) | cout << (2>=6);  0\false |
| **<=** (меньше или равно) | cout << (2<=6);  1\true |
| ***Операторы равенства*** | |
| **==** (равно) | cout << (2==6);  0\false |
| **!=**(не равно) | cout << (2!=6);  1\true |

Вопрос мы всегда задаем слева направо, например: «значение слева ‘больше’ чем значение справа?». В результате такой проверки, программа принимает одно из двух возможных значений – истина или ложь.

Также, рассмотрим с вами операции объединения, с которыми вы могли сталкиваться на уроках информатики. Они могут сравнивать не только два значения, но и создавать некие условия или объединять несколько условий сравнения в одно.

|  |  |
| --- | --- |
| ***Отрицательная инверсия и логические операции объединения*** | |
| **!** | **НЕ** (инверсия) |
| **&&** | **И** (конъюнкция) |
| **||** | **ИЛИ** (дизъюнкция) |

**Инверсия.** Логическое отрицание, при котором условие становится противоположно исходному условию.

|  |  |
| --- | --- |
| **А** | **!А** |
| 0 | 1 |
| 1 | 0 |

**Конъюнкция.** Логическое умножение, при котором составное условие истинно тогда и только тогда, когда истинны все входящие в него простые условия.

|  |  |  |
| --- | --- | --- |
| **А** | **В** | **А&&B** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

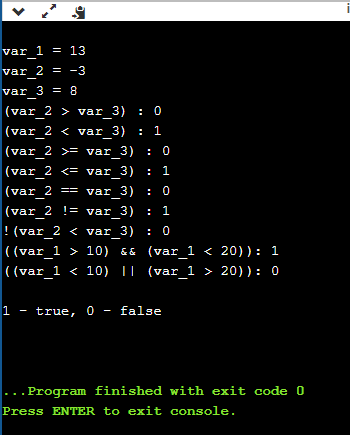
**Дизъюнкция.** Логическое сложение, при котором составное условие ложно тогда, когда ложны все входящие в него простые условия.

|  |  |  |
| --- | --- | --- |
| **А** | **В** | **А||B** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Рассмотрим все в одном коде:

Рисунок 19 – Код программы

Результат программы:

  
Рисунок 20 – Результат программы

* **Задачки**

1. Написать программу, которая указана в примере к данному параграфу, только пусть пользователь сам вводит значения.

В этом параграфе мы рассмотрели с вами основные логические операции в С++. Возможно некоторые моменты еще непонятны и трудны в понимании, но это не страшно. Больше практики и будет проще!

В следующем учебном пособие мы с вами продолжим изучать программирование и познакомимся со многими интересными темами и примерами.