Структура программы в C++

Прежде чем приступить к написанию программ, необходимо изучить структуру программ на языке программирования C++. Своими словами, структура программ это разметка рабочей области (области кода) с целью чёткого определения основных блоков программ и синтаксиса. Структура программ несколько отличается в зависимости от среды программирования. Мы ориентируемся на IDE Microsoft Visual Studio, и по этому примеры программ будут показаны именно для MVS. Если вы используете другую IDE, то вам не составит труда перенести код из MVS в другие среды разработки, и вы поймете со временем, как это сделать.

Структура программ для Microsoft Visual Studio.

```
1  // struct_program.cpp: определяет точку входа для консольного приложения.
2  #include "stdafx.h"
3  //здесь подключаем все необходимые препроцессорные директивы
4  int main() { // начало главной функции с именем main
5  //здесь будет находится ваш программный код
6  }
```

В строке 1 говорится о точке входа для консольного приложения, это значит, что данную программу можно запустить через командную строку Windows указав имя программы, к примеру, такое struct_program.cpp. Строка 1 является однострочным комментарием, так как начинается с символов //, подробнее о комментариях будет рассказано в следующей статье. В строке 2 подключен заголовочный файл "stdafx.h". Данный файл похож на контейнер, так как в нем подключены основные препроцессорные директивы (те, что подключил компилятор, при создании консольного приложения), тут же могут быть подключены и вспомогательные (подключенные программистом).

include — директива препроцессора, т. е. сообщение препроцессору. Строки, начинающиеся с символа # обрабатываются препроцессором до компиляции программы.

Препроцессорные директивы также можно подключать и в строках, начиная после записи #include "stdafx.h" до начала главной функции. Причём такой способ подключения библиотек является основным, а использование "stdafx.h" — это дополнительная возможность подключения заголовочных файлов, которая есть только в MVS. С 4-й по 6-ю строки объявлена функция main. Строка 4 – это заголовок функции, который состоит из типа возвращаемых данных (в данном случае int), этой функцией, и имени функции, а также круглых скобок, в которых объявляются параметры функции.

int — целочисленный тип данных

Между фигурными скобочками размещается основной программный код, называемый еще телом функции. Это самая простая структура программы. Данная структура написана в Microsoft Visual Studio. Все выше сказанное остается справедливым и для других компиляторов, кроме строки 2.

Управляющие символы C++

Управляющие символы (или как их ещё называют — **escape-последовательность**) — символы которые выталкиваются в поток вывода, с целью форматирования вывода или печати некоторых управляющих знаков C++. Основной список управляющих символов языка программирования C++ представлен ниже (см. Таблица 1).

Таблица 1 — Управляющие символы C++

Символ	Описание
\r	возврат каретки в начало строки
\n	новая строка
\t	горизонтальная табуляция
\v	вертикальная табуляция
\»	двойные кавычки
\’	апостроф
****	обратный слеш
\0	нулевой символ
\?	знак вопроса
\a	сигнал бипера (спикера) компьютера

Все управляющие символы, при использовании, обрамляются двойными кавычками, если необходимо вывести какое-то сообщение, то управляющие символы можно записывать сразу в сообщении, в любом его месте. Ниже показан код программы, использующей управляющие символы.

```
1  // in_out.cpp: определяет точку входа для консольного приложения.
2
3  #include "stdafx.h"
4  #include <iostream>
5  using namespace std;
6
   int main()
```

```
7       {
8         cout << "\t\tcontrol characters C++"; // две табуляции и печать сообщения
9         cout << "\rcppstudio.com\n"; // возврат каретки на начало строки и печать сообщения
10        cout << "\'formatting\' output with \"escape characters\""; // одинарные и двойные ковычки
11        cout << "\a\a\a\a\a\a\a\a\a\a\a\a\a\a" <<endl; //звуковой сигнал биппера
12        system("pause");
13        return 0;
14      }
15
```

В **строке 9** в выходной поток поступают две табуляции \t\t, после чего печатается сообщение **control characters C++**. В **строке 10** управляющий символ **\r** возвращает каретку в начало строки и печатает сообщение **cppstudio.com**, причём данное сообщение займет место двух табуляций из строки 9. После этого каретка будет переведена на новую строку, так как в конце сообщения строки 10 стоит символ **\n**. В **строке 11** первое и последнее слова сообщения обрамлены одинарными и двойными кавычками соответственно. В **строке 12** в выходной поток сдвигаются управляющие символы **\a**, эти символы запускают спикер компьютера. Результат работы программы показан ниже (см. рисунок 1).

```
'formatting' output with "escape characterscppstudio.com   control characters C++
'formatting' output with "escape characters"
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 — Управляющие символы C++

В данной теме мы рассмотрели основные управляющие символы C++, чаще всего вы будете пользоваться символами **\t** и **\n**. Управляющие символы C++ — это не основной способ форматированного вывода, но наиболее простой и наиболее часто используемый.

Арифметические операции C++

Думаю, понятно, для чего нужны арифметические операции, арифметика в программировании намного проще чем в математике. Нас интересуют следующие арифметические операции в C++:

+ — сложение;

- — вычитание;

***** — умножение;

/ — деление;

% — остаток от деления.

Ниже представлен программный код использующий арифметические операции в C++.

```
1
        // arithmetic.cpp: определяет точку входа для консольного приложения.
2
3
        #include "stdafx.h"
4
        #include <iostream>
5
        using namespace std;
6
7
        int _tmain(int argc, char* argv[])
8
        {
9
           double sum, razn, pow, div; // объявление переменных через запятую
10
           double a1; // отдельное объявление переменной a1
11
           double a2; // отдельное объявление переменной a2
12
           cout << "Vvedite pervoe chislo: ";
13
           cin >> a1;
14
           cout << "Vvedite vtoroe chislo: ";
15
           cin >> a2;
16
           sum  = a1 + a2;  // операция сложения
17
           razn = a1 - a2;  // операция вычитания
18
           pow  = a1 * a2;  // операция умножения
19
           div  = a1 / a2;  // операция деления
20
           cout << a1 << "+" << a2 << "=" << sum  << endl;
21
           cout << a1 << "-" << a2 << "=" << razn << endl;
22
           cout << a1 << "*" << a2 << "=" << pow  << endl;
23
           cout << a1 << "/" << a2 << "=" << div  << endl;
24
           system ("pause");
25
           return 0;
        }
26
```

В **строке 9** объявляются переменные с именами sum, razn, pow, div тип данных у которых double – вещественный тип данных (то есть эти переменные могут хранить такие числа: 0.99; 3.0; 21.6; — 43.15; 345.342).

Любые переменные можно использовать только после объявления. Переменные можно также и инициализировать при объявлении, пример:

```
1    double sum = 10;
```

Это значит, что мы объявили переменную с именем sum типа double и присвоили ей значение **10**. Переменные можно инициализировать и объявлять как через запятую так и по отдельности каждую (**строки 10 и 11**).

```
1    cin >> a1; // прочитать первое число в переменную a1.
1    cin >> a2; // прочитать второе число в переменную a2.
```

Символы >> называются операцией извлечения из потока. Данные символы используются вместе с оператором ввода **cin**.

```
1    sum  = a1 + a2;  // сложить два числа и записать их сумму в переменную sum
2    razn = a1 - a2; // вычесть из первого числа второе и записать их разность в переменную razn
3    pow  = a1 * a2;  // умножить два числа и записать их произведение в переменную pow
4    div  = a1 / a2;  // разделить первое число на второе и записать их частное в переменную div
```

С **20-й по 23-ю строки** мы выводим результаты наших арифметических операций. Также как и в математике в языке программирования C++ с помощью скобочек формируется порядок вычислений в сложных выражениях, например: **((a+b)*c)-d**
Первое действие: **a+b**;

Второе действие: умножение на **c**;

Третье действие: вычитание **d**;

Результат работы программы, выполняющей четыре арифметические операции **+**, **-**, *****, **/**, над двумя введенными числами, показан ниже (см.Рисунок 1).

```
Vvedite pervoe chislo: 10
Vvedite vtoroe chislo: 5
10+5=15
10-5=5
10*5=50
10/5=2
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 — Арифметические операции C++

Рассмотрим еще одну арифметическую операцию % — остаток от деления, для этого изучим подробно следующий код программы:

```
/ ost_division.cpp: определяет точку входа для консольного приложения.

#include "stdafx.h"
```

```
#include <iostream>
using namespace std;

int _tmain(int argc, char* argv[])
{
   cout << "8  % 4  = " << 8 % 4   << endl; // выполнение операции "остаток от деления"
   cout << "7  % 4  = " << 7 % 4   << endl;
   cout << "2  % 4  = " << 2 % 4   << endl;
   cout << "43 % 10 = " << 43 % 10 << endl;
   system("pause");
   return 0;
}
```

Результат работы программы, выполняющей операцию «**остаток от деления**» смотреть на рисунке 2.

```
% 4  = 0
7  % 4  = 3
2  % 4  = 2
43 % 10 = 3
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2 — Арифметические операции C++

- 1-й случай: четверка может два раза поместиться в восьмерке остаток будет равен 0;
- 2-й случай: четверка может один раз поместиться в семерке и остаток будет равен 3;
- 3-й случай: четверка в двойке поместиться не может по этому остаток будет равен 2;
- 4-й случай: десять может четыре раза поместиться в сорока трех, и остаток будет равен 3;

Рассмотрев 4-ре случая использования операции — остаток от деления, надеюсь, вам стал понятен принцип работы этой операции. Если же нет, то поэкспериментируйте с программой, изменяйте числа и увидите как меняется результат.

Типы данных C++

В данном разделе будут рассмотрены основные типы данных в C++, эти типы данных ещё называются встроенными. Язык программирования C++ является расширяемым языком программирования. Понятие расширяемый означает то, что кроме встроенных типов данных, можно создавать свои типы данных. Поэтому в C++ существует огромное количество типов данных. Мы будем изучать только основные из них.

Таблица 1 — Типы данных C++

Тип	**байт**	**Диапазон принимаемых значений**
целочисленный (логический) тип данных		
bool	1	0 / 255
целочисленный (символьный) тип данных		
char	1	0 / 255
целочисленные типы данных		
short int	2	-32 768 / 32 767
unsigned short int	2	0 / 65 535
int	4	-2 147 483 648 / 2 147 483 647
unsigned int	4	0 / 4 294 967 295
long int	4	-2 147 483 648 / 2 147 483 647
unsigned long int	4	0 / 4 294 967 295
типы данных с плавающей точкой		
float	4	-2 147 483 648.0 / 2 147 483 647.0
long float	8	-9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0
double	8	-9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0

В таблице 1 представлены основные типы данных в C++. Вся таблица делится на три столбца. В первом столбце указывается зарезервированное слово, которое будет определять, каждое свой, тип данных. Во втором столбце указывается количество байт, которое отводится под переменную с соответствующим типом данных. В третьем столбце показан диапазон допустимых значений. Обратите внимание на то, что в таблице все типы данных расположены от меньшего к большему.

Тип данных bool

Первый в таблице — это тип данных bool — целочисленный тип данных, так как диапазон допустимых значений — целые числа от 0 до 255. Но как Вы уже заметили, в круглых скобочках написано — логический тип данных, и это тоже верно. Так как bool используется исключительно для хранения результатов логических выражений. У логического выражения может быть один из двух результатов true или false. true — если логическое выражение истинно, false — если логическое выражение ложно.

Но так как диапазон допустимых значений типа данных bool от 0 до 255, то необходимо было как-то сопоставить данный диапазон с определёнными в языке программирования логическими константами true и false. Таким образом, константе true эквивалентны все числа от 1 до 255 включительно, тогда как константе false эквивалентно только одно целое число — 0. Рассмотрим программу с использованием типа данных bool.

```
// data_type.cpp: определяет точку входа для консольного приложения.

#include "stdafx.h"
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
   bool boolean = 25; // переменная типа bool с именем boolean
   if ( boolean ) // условие оператора if
      cout << "true = "  << boolean << endl; // выполнится в случае истинности условия
   else
      cout << "false = " << boolean << endl; // выполнится в случае, если условие ложно
   system("pause");
   return 0;
}
```

В **строке 9** объявлена переменная типа bool, которая инициализирована значением 25. Теоретически после **строки 9**, в переменной boolean должно содержаться число 25, но на самом деле в этой переменной содержится число 1. Как я уже говорил, число 0 — это ложное значение, число 1 — это истинное значение. Суть в том, что в переменной типа bool могут содержаться два

значения — 0 (ложь) или 1 (истина). Тогда как под тип данных bool отводится целый байт, а это значит, что переменная типа bool может содержать числа от 0 до 255. Для определения ложного и истинного значений необходимо всего два значения 0 и 1. Возникает вопрос: «Для чего остальные 253 значения?».

Исходя из этой ситуации, договорились использовать числа от 2 до 255 как эквивалент числу 1, то есть истина. Вот именно по этому в переменной boolean содержится число 25 а не 1. В **строках 10 - 13** объявлен оператор условного выбора if, который передает управление оператору в **строке 11**, если условие истинно, и оператору в **строке 13**, если условие ложно. Результат работы программы смотреть на рисунке 1.

```
true = 1
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 — Тип данных bool

Тип данных char

Тип данных char — это целочисленный тип данных, который используется для представления символов. То есть, каждому символу соответствует определённое число из диапазона [0;255]. Тип данных char также ещё называют символьным типом данных, так как графическое представление символов в C++ возможно благодаря char. Для представления символов в C++ типу данных char отводится один байт, в одном байте — 8 бит, тогда возведем двойку в степень 8 и получим значение 256 — количество символов, которое можно закодировать. Таким образом, используя тип данных char можно отобразить любой из 256 символов. Все закодированные символы представлены в таблице ASCII.

ASCII (от англ. American Standard Code for Information Interchange) — американский стандартный код для обмена информацией.

Рассмотрим программу с использованием типа данных char.

```
1   // symbols.cpp: определяет точку входа для консольного приложения.
2
3   #include "stdafx.h"
4   #include <iostream>
5   using namespace std;
6
    int main(int argc, char* argv[])
```

```
7       {
8           char symbol = 'a'; // объявление переменной типа char и инициализация её символом 'a'
9           cout << "symbol = " << symbol << endl; // печать символа, содержащегося в переменной symbol
10          char string[] = "cppstudio.com"; // объявление символьного массива (строки)
11          cout << "string = " << string << endl; // печать строки
12          system("pause");
13          return 0;
14      }
15
```

Итак, в **строке 9** объявлена переменная с именем symbol, ей присвоено значение символа 'a' (**ASCII код**). В **строке 10** оператор cout печатает символ, содержащийся в переменной symbol. В **строке 11** объявлен строковый массив с именем string, причём размер массива задан неявно. В строковый массив сохранена строка "cppstudio.com". Обратите внимание на то, что, когда мы сохраняли символ в переменную типа char, то после знака равно мы ставили одинарные кавычки, в которых и записывали символ. При инициализации строкового массива некоторой строкой, после знака равно ставятся двойные кавычки, в которых и записывается некоторая строка. Как и обычный символ, строки выводятся с помощью оператора cout, **строка 12**. Результат работы программы показан на рисунке 2.

```
symbol = a
string = cppstudio.com
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2 — Тип данных char

Целочисленные типы данных

Целочисленные типы данных используются для представления чисел. В таблице 1 их аж шесть штук: short int, unsigned short int, int, unsigned int, long int, unsigned long int**.** Все они имеют свой собственный размер занимаемой памяти и диапазоном принимаемых значений. В зависимости от компилятора, размер занимаемой памяти и диапазон принимаемых значений могут изменяться. В таблице 1 все диапазоны принимаемых значений и размеры занимаемой памяти взяты для компилятора MVS2010. Причём все типы данных в таблице 1 расположены в порядке возрастания размера занимаемой памяти и диапазона принимаемых значений. Диапазон принимаемых значений, так или иначе, зависит от размера занимаемой памяти. Соответственно, чем больше размер занимаемой памяти, тем больше диапазон принимаемых значений. Также

диапазон принимаемых значений меняется в случае, если тип данных объявляется с приставкой unsigned — без знака. Приставка unsigned говорит о том, что тип данных не может хранить знаковые значения, тогда и диапазон положительных значений увеличивается в два раза, например, типы данных short int и unsigned short int.

Приставки целочисленных типов данных:

short — приставка укорачивает тип данных, к которому применяется, путём уменьшения размера занимаемой памяти;

long — приставка удлиняет тип данных, к которому применяется, путём увеличения размера занимаемой памяти;

unsigned (без знака) — приставка увеличивает диапазон положительных значений в два раза, при этом диапазон отрицательных значений в таком типе данных храниться не может.

Так, что, по сути, мы имеем один целочисленный тип для представления целых чисел — это тип данных int. Благодаря приставкам short, long, unsigned появляется некоторое разнообразие типов данных int, различающихся размером занимаемой памяти и (или) диапазоном принимаемых значений.

Типы данных с плавающей точкой

В C++ существуют два типа данных с плавающей точкой: float и double. Типы данных с плавающей точкой предназначены для хранения чисел с плавающей точкой. Типы данных float и double могут хранить как положительные, так и отрицательные числа с плавающей точкой. У типа данных float размер занимаемой памяти в два раза меньше, чем у типа данных double, а значит и диапазон принимаемых значений тоже меньше. Если тип данных float объявить с приставкой long, то диапазон принимаемых значений станет равен диапазону принимаемых значений типа данных double. В основном, типы данных с плавающей точкой нужны для решения задач с высокой точностью вычислений, например, операции с деньгами.

Итак, мы рассмотрели главные моменты, касающиеся основных типов данных в C++. Осталось только показать, откуда взялись все эти диапазоны принимаемых значений и размеры занимаемой памяти. А для этого разработаем программу, которая будет вычислять основные характеристики всех, выше рассмотренных, типов данных.

```
1    // data_types.cpp: определяет точку входа для консольного приложения.
2
```

```
#include "stdafx.h"
#include <iostream>
// библиотека манипулирования вводом/выводом
#include <iomanip>
// заголовочный файл математических функций
#include <cmath>
using namespace std;


int main(int argc, char* argv[])
{
   cout << "     data type      "   << "byte"                    << "      "    << "    max value  " << endl // заголовки столбцов
        << "bool             =  " << sizeof(bool)              << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных bool*/                    << (pow(2,sizeof(bool) * 8.0) - 1)
      << "char             =  " << sizeof(char)              << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных char*/                    << (pow(2,sizeof(char) * 8.0) - 1)
      << "short int        =  " << sizeof(short int)         << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных short int*/               << (pow(2,sizeof(short int) * 8.0 - 1) - 1
        << "unsigned short int =  " << sizeof(unsigned short int)  << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных unsigned short int*/      << (pow(2,sizeof(unsigned short int
        << "int              =  " << sizeof(int)               << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных int*/                     << (pow(2,sizeof(int) * 8.0 - 1) - 1)
        << "unsigned int     =  " << sizeof(unsigned int)      << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных unsigned int*/            << (pow(2,sizeof(unsigned int) * 8.0)
        << "long int         =  " << sizeof(long int)          << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных long int*/                << (pow(2,sizeof(long int) * 8.0 - 1) - 1
        << "unsigned long int  =  " << sizeof(unsigned long int)   << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных undigned long int*/       << (pow(2,sizeof(unsigned long int
        << "float            =  " << sizeof(float)             << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных float*/                   << (pow(2,sizeof(float) * 8.0 - 1) - 1)
        << "double           =  " << sizeof(double)            << "        " << fixed << setprecision(2)
/*вычисляем максимальное значение для типа данных double*/                  << (pow(2,sizeof(double) * 8.0 - 1) - 1
   system("pause");
```

```
34          return 0;
35      }
36
```

Данная программа выложена для того, чтобы Вы смогли просмотреть характеристики типов данных в своей системе. Не стоит разбираться в коде, так как в программе используются управляющие операторы, которые Вам, вероятнее всего, ещё не известны. Для поверхностного ознакомления с кодом программы, ниже поясню некоторые моменты. Оператор sizeof() вычисляет количество байт, отводимое под тип данных или переменную. Функция pow(x,y) возводит значение x в степень y, данная функция доступна из заголовочного файла <cmath>. Манипуляторы fixed и setprecision() доступны из заголовочного файла <iomanip>. Первый манипулятор — fixed**,** передаёт в поток вывода значения в фиксированной форме. Манипулятор setprecision(n) отображает n знаков после запятой. Максимальное значение некоторого типа данных вычисляется по такой формуле:

```
1 max_val_type = 2^(b * 8 - 1) - 1; // для типов данных с отрицательными и положительными числами
2 // где, b - количество байт выделяемое в памяти под переменную с таким типом данных
3 // умножаем на 8, так как в одном байте 8 бит
4 // вычитаем 1 в скобочках, так как диапазон чисел надо разделить надвое для положительных и отрицательных значений
5 // вычитаем 1 в конце, так как диапазон чисел начинается с нуля
6
7 // типы данных с приставкой unsigned
8 max_val_type = 2^(b * 8 ) - 1; // для типов данных только с положительными числами
9 // пояснения к формуле аналогичные, только в скобочка не вычитается единица
```

Пример работы программы можно увидеть на рисунке 3. В первом столбце показаны основные типы данных в C++, во втором столбце размер памяти, отводимый под каждый тип данных и в третьем столбце — максимальное значение, которое может содержать соответствующий тип данных. Минимальное значение находится аналогично максимальному. В типах данных с приставкой unsigned минимальное значение равно 0.

```
   data type      byte         max value
bool              =  1        255.00
char              =  1        255.00
short int         =  2        32767.00
unsigned short int =  2       65535.00
int               =  4        2147483647.00
unsigned int      =  4        4294967295.00
long int          =  4        2147483647.00
unsigned long int  =  4       4294967295.00
float             =  4        2147483647.00
double            =  8        92233720368547758O8.00
Для продолжения нажмите любую клавишу . . .
```

Рисунок 3 — Типы данных C++

Если, например, переменной типа short int присвоить значение 33000, то произойдет переполнение разрядной сетки, так как максимальное значение в переменной типа short int это 32767. То есть в переменной типа short int сохранится какое-то другое значение, скорее всего будет отрицательным. Раз уж мы затронули тип данных int, стоит отметить, что можно опускать ключевое слово int и писать, например, просто short. Компилятор будет интерпретировать такую запись как short int. Тоже самое относится и к приставкам long и unsigned. Например:

```
1  // сокращённая запись типа данных int
2  short a1; //  тоже самое, что и short int
3  long a1; // тоже самое, что и long int
4  unsigned a1; // тоже самое, что и unsigned int
5  unsigned short a1; // тоже самое, что и unsigned short int
```

Операции присваивания в C++

Для сокращённой записи выражений в языке программирования C++ есть специальные операции, которые называются операциями присваивания. Рассмотрим фрагмент кода, с использованием операции присваивания.

```
1  int value = 256;
2  value = value + 256; // обычное выражение с использованием двух операций:  = и +
3  value += 256; // сокращённое эквивалентное выражение с использованием операции
   присваивания
```

В строке 2 переменной value присваивается значение 512, полученное в результате суммы значения содержащегося в переменной value с числом 256. В строке 3 выражение выполняет аналогичную операцию, что и в строке 2, но выражение записано в упрощённом виде. В этом выражении присутствует операция присваивания со знаком плюс +=. Таким образом, операция += суммирует значение переменной value со значением, которое находится правее: 256, и присваивает результат суммы этой же переменной. Как видно из примера оператор в строке 3 короче оператора в строке 2, хоть и выполняет аналогичную операцию. Так что, если некоторую переменную нужно изменить, то рекомендуется использовать операции присваивания.

В C++ существует пять операций присваивания, не считая основную операцию присваивания: =.

1. += операция присваивания-сложения;
2. -= операция присваивания-вычитания;
3. *= операция присваивания-умножения;
4. /= операция присваивания-деления;
5. %= операция присваивания-остатка от деления;

Договоримся называть операции присваивания через дефис, чтобы было понятно о какой именно операции идёт речь. В таблице 1 наглядно показаны примеры использования операторов присваивания в языке программирования C++.

Операции присваивания в C++

Операция	**Обозначение**	**Пример**	**Экв.пример**	**Пояснение**
операция присваивания-сложения	**+=**	**var += 16**	**var = var + 16**	Прибавляем к значению переменной var число 16, результат суммирования сохраняется в переменной var
операция присваивания-вычитания	**-=**	**var -= 16**	**var = var — 16**	Вычитаем из переменной var число 16, результат вычитания сохраняется в переменной var
операция присваивания-умножения	***=**	**var *= 16**	**var = var * 16**	Умножаем значение переменной var в 16 раз, результат умножения присваивается переменной var
операция присваивания-деления	**/=**	**var /= 16**	**var = var / 16**	Делим значение переменной var на 16, результат деления присваивается переменной var
операция присваивания-остатка от деления	**%=**	**var %= 16**	**var = var % 16**	Находим остаток от деления и сохраняем его в переменной var

Разработаем программу, которая будет использовать операции присваивания.

```
1
2
3       // assignment.cpp: определяет точку входа для консольного приложения.
4
5       #include "stdafx.h"
6       #include <iostream>
7       using namespace std;
8
9       int main(int argc, char* argv[])
10      {
11         int value = 256;
12         cout << "value = " << value << endl;
13         value += 256; // сокращённое выражение с использованием операции присваивания - сложения
14         cout << "value += 256; >> " << value << endl;
15         value -= 256; // сокращённое выражение с использованием операции присваивания - вычитания
16         cout << "value -= 256; >> " << value << endl;
17         value *= 2; // сокращённое выражение с использованием операции присваивания - умножения
18         cout << "value *= 2;   >> " << value << endl;
19         value /= 8; // сокращённое выражение с использованием операции присваивания - деления
20         cout << "value /= 8;   >> " << value << endl;
21         system("pause");
           return 0;
        }
```

Для начала в строке 9 была объявлена переменная value, и инициализирована значением 256. В строках 11, 13, 15, 17, прописаны операции присваивания – сложения, вычитания, умножения и деления соответственно. После выполнения каждой операции присваивания оператор cout печатает результат. Результат работы программы (см. Рисунок 1).

```
value = 256
value += 256; >> 512
value -= 256; >> 256
value *= 2;   >> 512
value /= 8;   >> 64
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 – Операции присваивания в C++

На рисунке 1 наглядно показаны примеры выполнения операций присваивания, а также показан результат выполнения соответствующей операции присваивания.

Операции инкремента и декремента в C++

Инкремент ++ – это увеличение на единицу. Декремент -- – это уменьшение на единицу. Операции декремента и инкремента с лёгкостью заменяются арифметическими операциями или операциями присваивания. Но использовать операции инкремента и декремента намного удобнее.

```
1       //синтаксис операций инкремента и декремента
2       ++/*имя переменной*/;  // префиксный инкремент (преинкремент)
```

```
3   /*имя переменной*/++; // постфиксный инкремент (постинкремент)
4   --/*имя переменной*/;  // префиксный декремент (предекремент)
5   /*имя переменной*/--; // постфиксный декремент (постдекремент)
```

Синтаксис использования операций инкремента и декремента таков, что перед или после имени переменной ставится операция инкремента или декремента. Когда операция инкремента или декремента ставится перед именем переменной, то такая операция называется префиксным инкрементом (сокращённо — преинкрементом) или префиксным декрементом (сокращённо — предекрементом). А если операция инкремента или декремента ставится после имени переменной, то такая операция называется операцией постфиксного инкремента (сокращённо — постинкремент) или постфиксного декремента (сокращённо — постдекремент). При использовании операции преинкремента значение переменной, сначала, увеличивается на 1, а затем используется в выражении. При использовании операции постинкремента значение переменной сначала используется в выражении, а потом увеличивается на 1. При использовании операции предекремента, значение переменной, сначала, уменьшается на 1, а затем используется в выражении. При использовании операции постдекремента, значение переменной, сначала, используется в выражении, а потом уменьшается на 1. В таблице 1 показаны примеры выражений с использованием операций инкремента и декремента, а также приведена их краткая характеристика.

Операции инкремента и декремента в C++

Операция	**Обозначение**	**Пример**	**Краткое пояснение**
преинкремент	++	**cout << ++value;**	Значение в переменной value увеличивается, после чего оператор cout печатает это значение
предекремент	—	**cout << —value;**	Значение в переменной value уменьшается, после чего оператор cout печатает это значение
постинкремент	++	**cout << value++;**	Оператор cout печатает значение переменной value, затем увеличивает это значение на 1

Операции инкремента и декремента в C++

Операция	Обозначение	Пример	Краткое пояснение
постдекремент	—	**cout << value—;**	Оператор cout печатает значение переменной value, затем уменьшает это значение на 1

В примерах используется оператор cout, чтобы показать, например, в чём различие постфиксных и префиксных операций. Вместо оператора cout можно использовать любой другой, в зависимости от того, какую необходимо выполнить задачу. Разработаем программу на основе выражений из таблицы, которая наглядно покажет, как себя будут вести операции инкремента и декремента.

```
1   // increment_decrement.cpp: определяет точку входа для консольного приложения.
2
3   #include "stdafx.h"
4   #include <iostream>
5   using namespace std;
6
7   int main(int argc, char* argv[])
8   {
9      int value = 2011;
10     cout << "value   = " << value   << endl; // начальное значение
11     cout << "++value = " << ++value << endl; // операция преинкремента
12     cout << "value++ = " << value++ << endl; // операция постинкремента
13     cout << "value   = " << value   << endl; // конечное значение в переменной value после выполнения операции пости
14     cout << "--value = " << --value << endl; // операция предекремента
15     cout << "value-- = " << value-- << endl; // операция постдекремента
16     cout << "value   = " << value   << endl; // конечное значение в переменной value после выполнения операции постд
17     system("pause");
18     return 0;
19  }
```

Код программы простой, поэтому пояснений к нему не будет, тем более, что в комментариях всё расписано. Результат работы программы (см. Рисунок 1).

Рисунок 1 — Операции инкремента и декремента

Из рисунка 1 видно, что первым делом печатается значение переменной value и оно равно 2011 . Далее выполняется операции преинкремента, которая, в свою очередь, увеличивает на 1 значение переменной value, теперь в value содержится значение 2012. В строке 12 выполняется операция постинкремента, таким образом, сначала печатается старое значение переменной value – 2012, а потом значение в value увеличивается на 1. Оператор cout в строке 13, как раз, подтверждает изменение значения в переменной value, оно равно 2013. В строке 14 выполняется операции предекремента, а значит, значение в переменной value уменьшается на 1, и оператор cout печатает это значение, теперь value = 2012. В **строке 15** выполняется операция постдекремента, но сначала оператор **cout** печатает значение value, затем операция постдекремента уменьшает значение в переменной value. Теперь value = 2011, это подтверждает оператор cout в **строке 16**.

Как было выше сказано, операции инкремента и декремента можно заменить эквивалентными операциями, например:

```
1  /*операция*/ ++value; /*эквивалентна операции*/  value += 1;
2  /*операция*/ --value; /*эквивалентна операции*/  value -= 1;
3  /*операция*/ ++value; /*эквивалентна операции*/  value = value +1;
4  /*операция*/ --value; /*эквивалентна операции*/  value = value - 1;
```

Но зачем всё усложнять, если можно использовать операции инкремента и декремента, вместо операций присваивания. С точки зрения оформления программного кода, так даже будет правильнее.

Оператор выбора if else

В некоторых источниках говорится, что оператор выбора if else — самостоятельный оператор. Но это не так, if else — это всего лишь форма записи оператора выбора if. Оператор if else позволяет определить программисту действие, когда условие истинно и альтернативное действие, когда условие ложно. Тогда как if позволял определить действие при истинном условии.

Синтаксис записи оператора выбора if else:

```
1  if (/*проверяемое условие*/)
2  {
3    /*тело оператора выбора 1*/;
```

```
4   } else
5       {
6         /*тело оператора выбора 2*/;
7       }
```

Читается так: «Если проверяемое условие истинно, то выполняется **тело оператора выбора 1**, иначе (то есть проверяемое условие ложно) выполняется **тело оператора выбора 2**«. Обратите внимание на то, как записан оператор if else. Слово else специально сдвинуто вправо для того чтобы программный код был понятен и его было удобно читать.

Рассмотрим задачу с предыдущей темы, с использованием if else. Напомню условие задачи: «Даны два числа, необходимо их сравнить».

```
    // if_else.cpp: определяет точку входа для консольного приложения.
1
2
    #include "stdafx.h"
3
    #include <iostream>
4
    using namespace std;
5
6
    int main(int argc, char* argv[])
7
    {
8       int a, b;
9       cout << "Vvedite pervoe chislo: ";
10      cin >> a;
11      cout << "Vvedite vtoroe chislo: ";
12      cin >> b;
13      if ( a >= b) // если а больше либо равно b, то
14      {
15       cout << a << " >= " << b << endl;
16      } else  // иначе
17          {
18           cout << a << " <= " << b << endl;
19          }
20      system("pause");
        return 0;
```

```
}
```

В данном коде нас интересуют **строки 14-20**. Эти строки читаются так: если a (первое число) больше либо равно b (второе число), то выполнить оператор вывода в **строке 16**

```
1       cout << a << " >= " << b << endl;
```

иначе выполнить оператор вывода в **строке 19**

```
1       cout << a << " <= " << b << endl;
```

В данном ифе мы используем операции соотношений >= и <=. Условие перехода не совсем правильно, так как условие будет ложно только в том случае, если первое число будет меньше второго, во всех остальных случаях условие истинно. Значит, **строку 19** нужно записать так

```
1       cout << a << " < " << b << endl; // в кавычках записать не меньше или равно, а просто меньше.
```

А вот так сработала программа (см. Рисунок 1).

```
Vvedite pervoe chislo: 15
Vvedite vtoroe chislo: -4
15 >= -4
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 — Оператор выбора if else

Покажу еще один пример использования операторов выбора if else (так называемые вложенные операторы if else для множественного выбора).

Условие задачи:
Составить алгоритм находящий значение y, если y=x, при x<0; y=0, при 0<=x<30; $y=x^2$, при x>=30;

```
1       // inif_else.cpp: определяет точку входа для консольного приложения.
2
3       #include "stdafx.h"
4       #include <iostream>
5       using namespace std;
6
7       int main(int argc, char* argv[])
8       {
9          int x, y;
10         cout << "Vvedite x: ";
11         cin >> x;
12         if (x < 0)
13         {
14          y = x; // выполняется, если x меньше нуля
15         } else
                {
```

```
16              if ( (x >= 0) && (x < 30) )
17              {
18                y = 0; // выполняется, если x больше либо равно нуля и меньше 30
19              } else
20                    {
21                     if (x >= 30)
22                     {
23                      y = x * x; // выполняется, если x больше либо равен 30
24                     }
25                    }
26            }
27      cout << "y=" << y << endl;
28      system("pause");
        return 0;
    }
```

В данной задаче возможны три случая:1-й случай: x < 0;

2-й случай: x лежит в пределах от 0 (включая 0) до 30;

3-й случай: x больше или равен 30.

Заметьте новшество!! В **17 строке** такую запись: if ((x >= 0) && (x< 30))**,** я использовал символы && — это логическое И. Операция логического **И** && необходима для объединения нескольких простых условий в одно составное. В нашем случае необходимо проверить истинность двух условий: первое – x >= 0, второе – x < 30. Все проверяемое условие будет истинно, если истинны два простых условия. В математике правильной записью считается такая запись: 0 <= x < 30, а в C++ правильной записью считается вот такая запись: (x >= 0) && (x < 30) или такая 0 <= x && x < 30. Кстати круглые скобочки () && () не обязательны, так как условия простые, но для уверенности, я прописываю, всегда, данные скобочки и вам советую.

Разбор частного случая:

Допустим, пользователь ввел число 31. Начиная со **строки 12**, выполняется проверка условий. Читается так: «Если x (31 в нашем случае) < 0, то выполнить оператор в **строке 14**». Но так как 31 > 0 условие ложно мы переходим к слову else (иначе) **строка 15**. Дальше проверяем, входит ли число 31 в заданный интервал. Читается так: если x>=0 и x<30 **то выполнить оператор в строке 19**. Но так как число 31 не входит в заданный интервал, то условие ложно.

Подробно **строка 17**: программа сначала проверит первое простое условие x >= 0 – оно истинно, а если первое истинно, то программа перейдет к проверке второго простого условия x < 30 – оно ложно. Следовательно всё составное условие ложно, ведь в составном условии у нас используется логическая операция &&, а это значит, что все составное условие истинно только в том случае, когда истинны оба простых условия. Переходим к else (иначе), здесь у нас последний if, (**строка 22**). Выполняется проверка x >= 30. Читается так: Если x >= 30 то выполнить оператор, находящийся в строке 24. Наконец-то условие истинно, итак выполнился оператор в **строке 24**. И **строка 28** печатает получившееся значение. Ну, все, рассмотрели программу по мельчайшим деталям. Результат работы программы, в случае, если пользователь ввел число 31 (см. Рисунок 2)

```
Vvedite x: 31
y=961
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2 — Оператор выбора if else

Логические операции в C++

Так как в предыдущей статье, я впервые использовал логическую операцию, расскажу, какие они бывают, сколько их и как ими пользоваться.

В C++ существует три логические операции:

1. Логическая операция И &&, нам уже известная;
2. Логическая операция ИЛИ ||;
3. Логическая операция НЕ ! или логическое отрицание.

Логические операции образуют сложное (составное) условие из нескольких простых (два или более) условий. Эти операции упрощают структуру программного кода в несколько раз. Да, можно обойтись и без них, но тогда количество ифов увеличивается в несколько раз, в зависимости от условия. В следующей таблице кратко охарактеризованы все логические операции в языке программирования C++, для построения логических условий.

Таблица 1 — Логические операции C++

Операции	**Обозначение**	**Условие**	**Краткое описание**
И	**&&**	a == 3 **&&** b > 4	Составное условие истинно, если истинны оба простых условия

Таблица 1 — Логические операции C++

Операции	Обозначение	Условие	Краткое описание
ИЛИ	||	a == 3 || b > 4	Составное условие истинно, если истинно, хотя бы одно из простых условий
НЕ	!	!(a == 3)	Условие истинно, если a не равно 3

Сейчас следует понять разницу между логической операцией И и логической операцией ИЛИ , чтобы в дальнейшем не путаться. Пришло время познакомиться с типом данных bool –логический тип данных. Данный тип данных может принимать два значения: true (истина) и false (ложь). Проверяемое условие в операторах выбора имеет тип данных bool. Рассмотрим принцип работы следующей программы, и все будет понятно со всеми этими логическими операциями.

```
1
2
3       // or_and_not.cpp: определяет точку входа для консольного приложения.
4
5       #include "stdafx.h"
6       #include <iostream>
7       using namespace std;
8
9       int main(int argc, char* argv[])
10      {
11         bool a1 = true, a2 = false; // объявление логических переменных
12         bool a3 = true, a4 = false;
13         cout << "Tablica istinnosti log operacii &&" << endl;
14         cout << "true  && false: " << ( a1 && a2 )   << endl // логическое И
15              << "false && true: "  << ( a2 && a1 )   << endl
16              << "true  && true: "  << ( a1 && a3 )   << endl
17              << "false && false: " << ( a2 && a4 )   << endl;
18         cout << "Tablica istinnosti log operacii ||" << endl;
19         cout << "true  || false: " << ( a1 || a2 )   << endl // логическое ИЛИ
20              << "false || true: "  << ( a2 || a1 )   << endl
21              << "true  || true: "  << ( a1 || a3 )   << endl
22              << "false || false: " << ( a2 || a4 )   << endl;
23         cout << "Tablica istinnosti log operacii !"  << endl;
24         cout << "!true: " << ( ! a1 ) << endl // логическое НЕ
25              << "!false: "<< ( ! a2 ) << endl;
26         system("pause");
           return 0;
       }
```

Строки 9 и 10 вам должны быть понятны, так как здесь инициализируются переменные типа bool. Причем каждой переменной присваивается

значение true или false. Начиная с 9-й строки и заканчивая 20-й, показано использование логических операций. Результат работы программы (см. Рисунок 1).

```
Tablica istinnosti log operacii &&
true  && false: 0
false && true: 0
true  && true: 1
false && false: 0
Tablica istinnosti log operacii ||
true  || false: 1
false || true: 1
true  || true: 1
false || false: 0
Tablica istinnosti log operacii !
!true: 0
!false: 1
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 — Логические операции C++

Цикл for в C++

Цикл — многократное прохождение по одному и тому же коду программы. Циклы необходимы программисту для многократного выполнения одного и того же кода, пока истинно какое-то условие. Если условие всегда истинно, то такой цикл называется бесконечным, у такого цикла нет точки выхода.

В языке программирования C++ существуют такие циклы:

- цикл for
- цикл while
- цикл do while

Тело цикла будет выполняться до тех пор, пока условие будет истинно(т. е. true).

```
1    // форма записи оператора цикла for:
2
3    for (/*выражение1*/; /*выражение2*/; /*выражение3*/ )
4    {
5    /*один оператор или блок операторов*/;
6    }
```

Если в теле цикла for должен выполнится один оператор, тогда фигурные скобки можно опустить:

```
1    for (/*выражение1*/; /*выражение2*/; /*выражение3*/)
2    /*один оператор*/;
```

Рассмотрим подробно три выражения записанные в круглых скобках цикла for. Выражение 1 — объявление (и) или инициализация, ранее

объявленной, переменной-счетчика, которая будет отвечать за истинность условия в цикле for. Пример:

```
1    int counter = 0; // отсюда видно, что была объявлена переменная counter типа int и и
     значением 0
```

Переменная-счетчик всегда должна иметь целочисленный тип данных. Если переменная была объявлена в цикле (все равно в каком), по завершении цикла эта переменная будет уничтожена.

Разница между объявлением и инициализацией переменной:

```
1    counter; // объявление переменой count
2
3    counter = 9; // инициализация целочисленной  переменой count значением  9.
```

Выражение 2 — это условие продолжения цикла for, оно проверяется на истинность.

```
1    counter < 10; // условие истинно пока count строго меньше десяти!
```

Выражение 3 изменяет значение переменной-счетчика. Без выражения 3 цикл считается бесконечным, так как изменение содержимого переменной count выполняться не будет, и если изначально условие было истинным, то цикл будет бесконечным, иначе программа даже не войдет в цикл. Выражения 1, 2, 3 отделяются друг от друга обязательными разделителями, точкой с запятой. Тело цикла обрамляется фигурными скобочками, если тело цикла состоит из одного оператора ,то фигурные скобочки не нужны. Под изменением значения переменной подразумевается уменьшение или приращение значения , например:

```
1    for ( int counter = 0; counter < 15; counter++) // выполняется приращение переменной counter с шагом 1 от 0 до 15
```

- ++ это операция инкремента, увеличение значения переменной на единицу;
- -- это операция декремента, уменьшение значения переменной на единицу.

Очень часто неправильно интерпретируется запись пределов в цикле for, в нашем случае приращение переменной counter выполняется с шагом 1 от 0 до 15. Обратите внимание на конечный предел. В условии продолжения цикла стоит знак отношения строго меньше, а значит, когда значение в переменной counter будет равно 14, выполнится выход из цикла. Ниже показан пример работы программы (см. Рисунок 1).

```
1    // for.cpp: определяет точку входа для консольного приложения.
2
3    #include "stdafx.h"
4    #include <iostream>
5    using namespace std;
6
7    int main(int argc, char* argv[])
     {
```

```
8           for (int counter = 0; counter < 15; counter ++ ) // начало цикла
9               cout << " " << counter; // тело цикла
10          cout << endl;
11          system("pause");
12          return 0;
13      }
14
```

В строках 9, 10 записан цикл for, причём без фигурных скобочек. А раз отсутствуют фигурные скобочки, значит, телом цикла является следующий оператор, после заголовка цикла, в нашем случае — это оператор вывода:

```
1       cout << " " << counter; // тело цикла
```

Заголовком цикла for является **строка 9**:

```
1       for (int counter = 0; counter < 15; counter ++ ) // заголовок цикла
```

Фигурные скобочки можно не опускать, это кому как удобно. Но в данном случае без фигурных скобочек запись кода более компактна, чем со скобочками.

Рисунок 1 — Цикл for в C++

```
1       for ( int counter = 14; counter >= 0; counter --) // декремент переменной counter от 14 д
```

Ниже показан результат работы программы с использованием декремента, для уменьшения значения переменой-счетчика (см. Рисунок 2).

```
1
2       // for_dec.cpp: определяет точку входа для консольного приложения.
3
4       #include "stdafx.h"
5       #include <iostream>
6       using namespace std;
7
8       int main(int argc, char* argv[])
9       {
10          for (int count = 14; count >= 0; count --) // заголовок цикла
11              cout << " " << count; // тело цикла
12          cout << endl;
13          system("pause");
14          return 0;
15      }
16
```

В заголовке цикла for в условии продолжения цикла мы использовали нестрогий знак отношения >=, чтобы последним значением переменой-счётчика был 0.

Рисунок 2 — Цикл for в C++

Шаг в цикле for может быть отличным от единицы, а точнее, любым целым числом.

```
1    for ( int counter = 0; counter <= 20; counter += 5) // приращение переменной counter о
     5
```

В этом случае переменная counter будет равняться 5, 10, 15, и 20, после чего выполнится выход из цикла for.

```
1    for ( int counter = 20; counter >= 0; counter -= 5) // изменение переменной counter от 2
```

Далее показан код программы, которая подсчитывает количество чётных чисел в интервале от 0 до 50 включительно.

```
1
2       // for_example.cpp: определяет точку входа для консольного приложения.
3
4       #include "stdafx.h"
5       #include <iostream>
6       using namespace std;
7
8       int main(int argc, char* argv[])
9       {
10          int counter_even = 0;
11          for (int count = 2; count <= 50; count += 2) // заголовок цикла
12              counter_even ++; // подсчёт чётных чисел
13          cout << "number of even numbers = " << counter_even << endl;
14          system("pause");
15          return 0;
        }
```

В строке 9 объявлена и инициализирована нулём переменная для хранения количества чётных чисел. В заголовке цикла for в строке 10 переменная counter инициализирована двойкой, так как двойка — первое чётное число, начиная с 0. Шаг цикла равен двум, таким образом следующее чётное число это 4. В строке 11 переменная counter_even инкрементируется на каждом этапе цикла for. Результат работы программы показан ниже (см. Рисунок 3).

Рисунок 3 — Цикл for в C++