

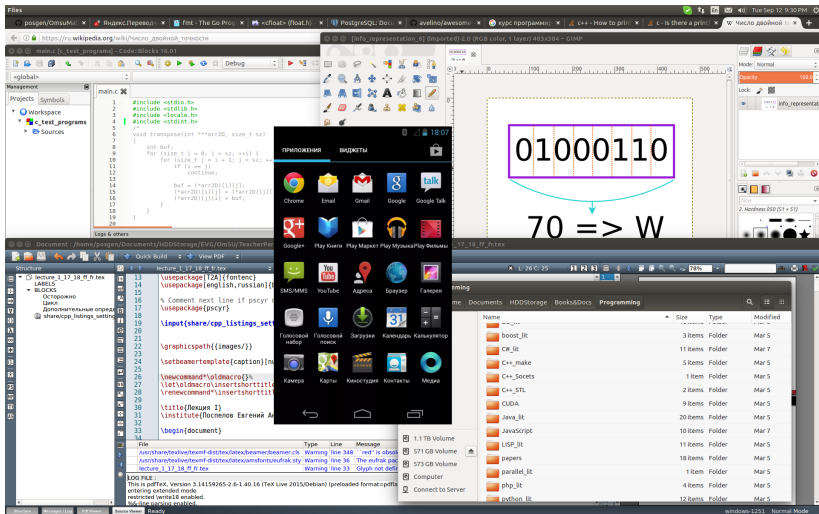
I

posevg@yandex.ru

12 февраля 2020

До изучения языка программирования:
взгляд снизу на типы данных

Работа с информацией: что видим мы?



Работа с информацией: что видит компьютер?

Современные процессоры работают с потоком **бит** - элементарных ячеек для хранения информации, принимающих, как правило, только два значения

```
0101011101010100100101001000100001001001
0111001010101010010101001110010010010010
1010101010101011111010010011101101101010
0001110110100101110101010101101010101010
0101011101101010101010101010010101001001
11001100110101010101100011010101111010011
0101010100001101011100010100010100010101
```

Работа с информацией: что видит компьютер?

Поток бит делится на блоки различной длины. Учитывая определение бита, каждый блок представляет собой некоторое число в двоичной системе исчисления

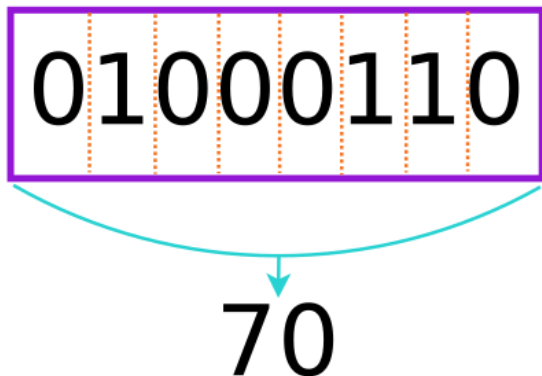
```
0101011101010100100101001000100001001001
011100101010101010010101001110010010010010
101010101010101011111010010011101101101010
0001110110100101101010101011010101010101
01010111011010101010101010101010010101001001
1100110011010101010110001101010111010011
0101010100001101011100010100010100010101
```

Работа с информацией: что видит компьютер?

Минимальным блоком в современных ЭВМ является **байт**. На всех популярных ОС 1 байт состоит из 8 бит

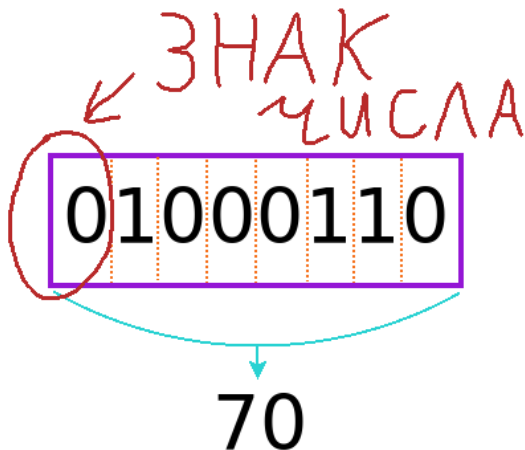


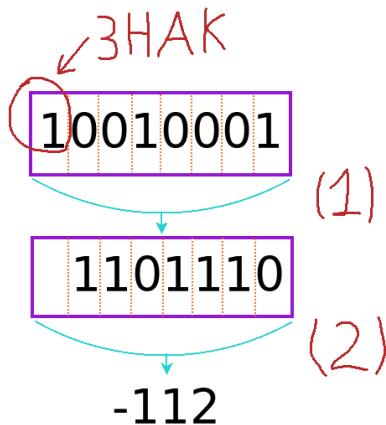
Целое число: восьми бит хватит для хранения 256 чисел в диапазоне $[0; 255]$



Работа с информацией: интерпретация байта

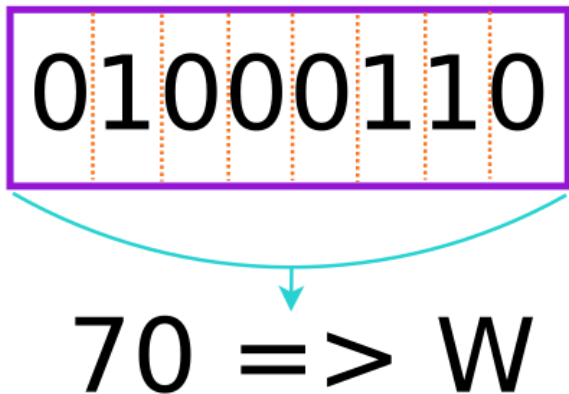
Целое число со знаком: первый бит отвечает за знак, диапазон теперь - $[-128; 127]$





Когда бит знака равен единице - число отрицательно. **(1)** - инвертирование значащих битов, **(2)** - получение отрицательного числа в десятичной системе исчисления.

Некоторый символ: вводится таблица соответствия между числом и набором текстовых символов



Определение

Тип данных в языках программирования представляет собой:

- заданное хранилище (storage): кратная числу байт область в памяти;
- конечное множество допустимых значений и их интерпретацию;
- заданные операции для работы с допустимыми значениями.

Переменная конкретного типа данных при её использовании в программе *всегда* занимает ограниченный блок байт в оперативной памяти ЭВМ.

Переходим к конкретному языку
программирования

Где получить информацию о C++?

Литература

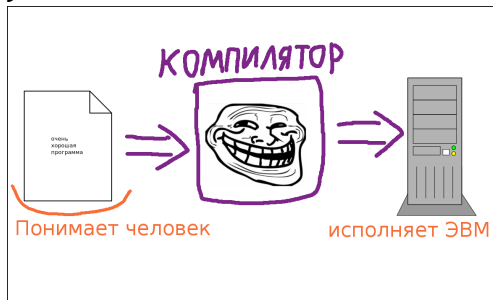
- 1 Алекс Эллайн, "C++. От ламера до программера 2015 г., Питер ("Jumping into C++")
- 2 Татьяна Павловская, "C/C++. Процедурное и объектно-ориентированное программирование 2014 г., Питер
- 3 Бьярне Страуструп "Программирование. Принципы и практика с использованием C++ 2016 г., Вильямс
- 4 Питер Готшлинг, "Современный C++ для программистов, инженеров и учённых 2016 г., Вильямс

Альтернатива онлайн

- 1 github.com/posgen/OmsuMaterials
- 2 en.cppreference.com/w/cpp (ru.cppreference.com/w/)
- 3 www.cplusplus.com/reference/
- 4 <https://www.cprogramming.com/tutorial/c++-tutorial.html>

Является

- 1 **императивным:** для написания программы используются операторы и управляющие конструкции языка программирования;
- 2 **компилируемым:**



- 3 **статически типизированным:** типы всех переменных программы известны в момент компиляции.

- Позволяет писать программы в **процедурном**, **объектно-ориентированном** и **обобщённом** стилях.
- На C++ созданы: Chrome, Firefox, Opera, Safari, IE Edge
- Используется для прикладных программ:
создания/редактирование изображений, работа с видео,
создание графических интерфейсов
- Обработка больших объёмов данных
- Машинное обучение: фреймворк **tensorflow**

- Компания Bell Labs, Бьярне Страуструп (Bjarne Strastrup)
- 1985, первый коммерческий релиз языка C++



- 1998 - первый стандарт ISO C++98
- На данный момент выпущены C++03, C++11, C++14, C++17

Какие особенности определяют язык программирования?

- 1 *Фундаментальные* типы данных и операторы для работы с переменными этих типов.
- 2 Управление ходом выполнения программы (циклы, условные и безусловные переходы).
- 3 Способ обособления блоков кода, для неоднократного использования (функции).
- 4 *Специальные* типы данных и способы работы с ними.
- 5 Конструкции языка для создания *пользовательских* типов данных.

Математика:

$$force(x, y) = 2x + 4y \sin(x)$$

$$a = 5.5$$

$$b = -8.34$$

$$res = force(a, b)$$

Пример использования C++

Математика:

$$force(x, y) = 2x + 4y\sin(x)$$

$$a = 5.5$$

$$b = -8.34$$

$$res = force(a, b)$$

C++:

```
1 double force(double x, double y)
2 { return 2 * x + 4 * y * sin(x); }
3
4 double a = 5.5,
5         b = -8.34;
6
7 double res = force(a, b);
```

Определение

Идентификатором в языке программирования называется непрерывная последовательность символов, которые используются для именования переменных, функций, пользовательских типов данных.

В языке C++ в состав идентификатора могут входить только **буквы, цифры и символ нижнего подчёркивания "_"**. При этом начинаться каждый идентификатор должен только с **буквы или символа подчёркивания**.

Прежде, чем идти дальше

Кроме того, в языках программирования существует определённый набор слов (в широком смысле - символьных последовательностей), которые не могут быть использованы в качестве идентификаторов. Такие слова называются **ключевыми**, вот они:

**alignas alignof and and_eq asm auto bitand bitor bool break
case catch char char16_t char32_t class compl const
constexpr const_cast continue decltype default delete do
double dynamic_cast else enum explicit export extern false
float for friend goto if inline int long mutable namespace
new noexcept not not_eq nullptr operator or or_eq private
protected public register reinterpret_cast return short
signed sizeof static static_assert static_cast struct switch
template this thread_local throw true try typedef typeid
typename union unsigned using virtual void volatile wchar_t
while xor xor_eq union unsigned using virtual void volatile
wchar_t while xor xor_eq**

Прежде, чем идти дальше

Комментарии - произвольный текст в файле с исходным кодом, который игнорируется компилятором и никак не влияет на ход программы.

```
1 // Это однострочный комментарий
2
3 /*
4 Пример
5     многострочного
6     комментария
7 */
```

Предупреждение! Многострочные комментарии не могут быть вложенными

Типы, переменные, операторы

Общий вид создания переменной в C++:

<тип> <идентификатор>;

Эта форма называется **объявлением** переменной.

Первый тип данных для знакомства: **int**

- предназначен для хранения целых чисел *со знаком*;
- диапазон значений (на большинстве 64-битных ОС тип **int** занимает 4 байта): $[-2_147_483_648; 2_147_483_647]$
- При объявлении переменной под неё выделяется место в оперативной памяти
- Однако C++ не даёт никаких гарантий относительно того, какое значение (целое число) окажется в объявленной переменной (фактически, не гарантируется, что в выделенном ОС блоке памяти сохранено конкретное число)

Рассматриваемый тип данных: **int**

Пример объявления переменных:

```
1 // Переменные конкретного типа можно объявлять по одной
2 int counter;
3 int velocity;
4
5 // А можно и несколько одновременно, через запятую
6 int width, height, area;
```


Типы, переменные, операторы

Рассматриваемый тип данных: **int**

Пример объявления переменных:

```
1 // Переменные конкретного типа можно объявлять по одной
2 int counter;
3 int velocity;
4
5 // А можно и несколько одновременно, через запятую
6 int width, height, area;
```

Изменение значений, хранящихся в переменных, происходит с помощью **операторов**.

Определение

Оператором в языках программирования называется символьная конструкция, которая производит действия над одним или более объектом.

Операторы в C++ делятся на:

- 1 **унарные** - требуется один объект для совершения действия, ставится после оператора;
- 2 **бинарные** - два объекта, по одному до и после оператора.

+1

Также в языке программирования присутствует единственный **тернарный** оператор - он состоит из двух символов и требует три объекта для своей работы. Будет рассмотрен в разделе о ходе выполнения программы.

Типы, переменные, операторы

Тип **int**: присвоение значений с помощью оператора «=»

```
1 int acceleration_rate;  
2 // Присвоение переменной начального значения  
3 acceleration_rate = -7;  
4  
5 /*  
6 Присвоение можно (и нужно!) делать  
7 при объявлении переменной.  
8 */  
9 int good_rate = 16, bad_rate = -10, karma;  
10 karma = good_rate + bad_rate;
```

Определение

Присвоение значения переменной в момент её объявления называется её **инициализацией**.

Типы, переменные, операторы

Тип **int**: арифметические операции - «+, -, *, /, %»

```
1 int balance = 10, rate, total;
2
3 rate = balance - 8;
4 total = 2 * balance * (6 - rate);
5
6 // Целочисленное деление – дробная часть ←
   отсекается
7 rate = 7 / 2; // rate равен 3
8
9 // Взятие остатка от деления
10 rate = 7 % 2; // rate равен 1
11
12 // Ошибка Времени Выполнения:
13 // rate = 11 / 0;
```

Обратите внимание: при работе с целыми числами нельзя допускать деления на нуль, это ошибка, вызывающая немедленную остановку выполнения программы.

Типы, переменные, операторы

Все возможные виды объявления и их инициализации переменной:

- (1) [...] <тип> <идентификатор> [= <значение>];
- (2) [...] <тип> <идентификатор> [{ <значение> }];
- (3) [...] <тип> <идентификатор> [(<значение>)];

, где треугольные скобки означают обязательные части, квадратные - опциональные, троеточие - дополнительные характеристики переменной или указания компилятору.

Предупреждение! (3) форма не рекомендуется к использованию с фундаментальными типами данных.

Правило хорошего тона

По возможности, **каждая** переменная программы должна быть проинициализированна разумным начальным значением.

Примеры форм (1) и (2) инициализации переменных

```
1 int count = 8; // (1)
2 int rate {101}; // (2)
```

Правило хорошего тона

Для фундаментальных типов данных предпочтительной является форма (1).

Типы, переменные, операторы

Целочисленный тип	Размер на 64-битных ОС
<code>short int</code> <code>unsigned short int</code>	2 байта
<code>int</code> <code>unsigned int</code>	4 байта
<code>long int</code> <code>unsigned long int</code>	8 байт
<code>long long int</code> <code>unsigned long long int</code>	8 байт
<code>size_t</code>	8 байт, беззнаковый тип

- Перечёркнутый `int` - может быть пропущен
- Стандарт языка не определяет конкретного размера каждого типа, только соотношение размеров между ними:
`sizeof(short) <= sizeof(int) <= sizeof(long) <= ...`
- `size_t` - размер определяется максимальным числом оперативной памяти на данной архитектуре ЭВМ

В примерах с присвоением значений переменным были использованы конкретные числа (-7, 16, -10). Поскольку язык C++ является типизированным, то данные числа в момент компиляции должны также получить тип данных. Когда в программе встречается целое число, то компилятор пробует сначала поместить его в тип **int**, затем в **long**, потом в **long long**. А если не получилось - выдаст ошибку компиляции.

Определение

Значения конкретных типов данных, записанные в программе в явном виде, называются **литералами**.

Типы, переменные, операторы

Целочисленные типы: расширенные операторы присваивания, инкремент/декремент

```
1 int balance = 5, rate = 10;
2
3 // Оператор присваивания сначала вычисляет
4 // выражение справа
5 balance = balance + 1;
6 // Сокращённая запись
7 balance += 1;
8 /* Также определены: -=, *=, /=, %= */
9
10 // Инкремент/Декремент – увеличение/уменьшение ←
   значения переменной на единицу
11 rate++; // rate стал равным 11
12 ++rate; // --//– 12
13 rate--; // снова 11
14 --rate; // теперь 10
```

Типы, переменные, операторы

Целочисленные типы: тонкости инкремента/декремента

```
1 // Разница пре- и пост- инкремента
2 int balance = 5, total = 5;
3
4 printf("balance = %d\n", balance++);
5 // напечатает в консоли "balance = 5"
6 // значение balance равно 6
7
8 printf("total = %d\n", ++total);
9 // напечатает "total = 6"
10 // значение total равно 6
11
12 // Пример унарных операторов
13 int number = -6;
14 +number;
15 -number;
```

Типы, переменные, операторы

Целочисленные типы: побитовые операции (**только для положительных целочисленных значений**)

```
1 unsigned number = 4, next_number;
2
3 // Побитовый сдвиг вправо: number * (2^2)
4 next_number = number << 2;
5
6 // Побитовый сдвиг влево: number / (2^3)
7 next_number = number >> 3;
8
9 // Побитовое "И"
10 next_number = number & 2;
11
12 // Побитовое "ИЛИ"
13 next_number = number | 3;
14
15 // Побитовое исключающее "ИЛИ" (xor)
16 next_number = number ^ 3;
17 // Поддерживаются >>=, |=, ^= и аналоги
```

Типы, переменные, операторы

Целочисленные типы: оператор **sizeof**

```
1 // Двоичный литерал, C++14
2 int i_num = 0b01100010;
3 long l_num = 0345;           // Восьмиричный
4 size_t sz_num = 0xff11c;    // Шестнадцатиричный
5
6 printf("Size of int: %lu\n", sizeof(int));
7 printf("Size of long: %lu\n",
8        sizeof(l_num));
9 printf("Size of size_t: %lu\n",
10        sizeof(size_t));
```

Возможный вывод в консоли:

```
Size of int: 4
Size of long: 8
Size of size_t: 8
```

Определение

Адресом перменной называют номер первого байта связанного с ней блока памяти (хранилища) в памяти.

Унарный оператор взятия адреса - амперсанд **&**

```
1 int degree = 120;  
2 int rank = 8;  
3  
4 printf("Addr of \"degree\": %p\n", &degree);  
5 printf("Addr of \"rank\": %p\n", &rank);
```

Возможный вывод в консоли:

Addr of "degree": 0x7ffff89f4200

Addr of "rank": 0x7ffff89f4204

Типы, переменные, операторы

bool - логический (булев) тип данных, принимающий только два значения - **true** или **false**

Операция	Оператор
отрицание	!
логическое И	&&, and
логическое ИЛИ	, or

```
1 bool truth = true, falsey = false, result;
2
3 // Логическое "И"
4 result = truth && falsey; // result равен false
5 // Логическое "ИЛИ"
6 result = truth || falsey; // result равен true
7 // допустимо и так
8 result = truth or falsey; // result равен true
9
10 // Отрицание:
11 result = !falsey; // result равен true
```

```
1 bool var1, var2;
```

```
2 // Переменным присваиваем значения...
```

Таблица истинности

	var1	var2	Результат
&&	true	true	true
&&	true	false	false
&&	false	true	false
&&	false	false	false
	true	true	true
	true	false	true
	false	true	true
	false	false	false

Типы, переменные, операторы

Логический тип: совместимость с языком C

```
1 bool to_be_or_not = 0;  
2 // to_be_or_not равна false  
3 printf("bool to int (1): %d\n", to_be_or_not);  
4  
5 to_be_or_not = 1;  
6 // to_be_or_not равна true  
7 printf("bool to int (2): %d\n", to_be_or_not);  
8  
9 to_be_or_not = 118;  
10 printf("bool to int (3): %d\n", to_be_or_not);
```

Вывод в консоль:

```
bool to int (1): 0  
bool to int (2): 1  
bool to int (3): 1
```

Переменные типа **bool** могут участвовать в арифметических выражениях (**но не должны**)

Типы, переменные, операторы

Операторы сравнения

Операция	Оператор
равенство	<code>==</code>
неравенство	<code>!=</code>
больше	<code>></code>
меньше	<code><</code>
больше или равно	<code>>=</code>
меньше или равно	<code><=</code>

Все операторы сравнения в C++ возвращают значения типа **bool**.

```
1 int width = 5, height = 4;
2 bool status = width > height;
3 // status равен true
4
5 status = (width == height);
6 // status равен false
```

Типы, переменные, операторы

Числа с плавающей точкой

float - число с плавающей точкой одинарной точности ($\approx 7-8$ знаков после запятой). Максимальное значение - $\approx 10^{38}$, минимальное - $\approx 10^{-38}$.

double - число с плавающей точкой двойной точности ($\approx 15-16$ знаков после запятой). Максимальное значение - $\approx 10^{308}$, минимальное - $\approx 10^{-308}$.

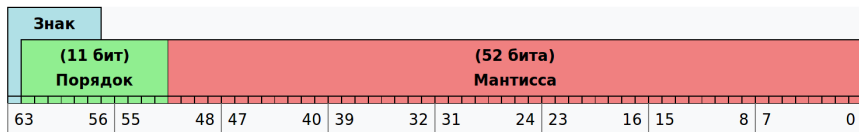
long double - двойной точности ($\approx 15-16$ знаков после запятой), расширенный диапазон: 10^{4932} .

Размер типов: `sizeof(float)` -> 32 бита, `sizeof(double)` -> 64 бита. Тип **long double** занимает как минимум 80 бит.

```
1 double speed = 5.5, distance;  
2 distance = speed / 3.0;  
3  
4 speed *= 4.0;
```

Типы, переменные, операторы

Числа с плавающей точкой: как представлены в памяти



Конкретное число вычисляется как:

$$(-1)^{\text{знак}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i} \right) 2^{\text{Порядок} - 1023}$$

, где b_i - i -ый бит мантииссы.

Формат чисел с плавающей запятой одинарной и двойной точности стандартизирован: IEEE 754.

В точности числа не учитывается десятичная экспонента:

$$1184 \rightarrow 1.184 \times 10^3$$

Числа с плавающей точкой: на нуль делить разрешается

```
1 #include <cmath>
2
3 double super_rate = 5.5;
4 super_rate /= 0.0;
5
6 bool is_infinity = isinf(super_rate);
7 // is_infinity равна true
```

Числа с плавающей точкой: понятие Not-A-Number (NaN)

```
1 #include <cmath>
2
3 // sqrt – вычисление квадратного корня
4 double super_rate = sqrt(-4.5);
5
6 bool is_nan = isnan(super_rate);
7 // is_nan равна true
```

Предупреждение

Сравнение чисел с плавающей точкой неоднозначно

```
1 double first_rate = 0.2 + 0.1 * 2;  
2 double second_rate = 0.2 * 2;  
3  
4 bool is_equal = (first_rate == second_rate);  
5 // что будет в is_equal — непонятно
```

Числа с плавающей точкой: подход в сторону правильного сравнения

```
1 #include <cmath>
2
3 double rate1 = 0.4, rate2 = 0.8 / 2;
4
5 // Определяем для себя приемлемую точность
6 double eps = 1.5E-8;
7
8 // abs — вычисляет модуль аргумента
9 bool is_equal = abs(rate1 - rate2) < eps;
10 // есть уверенность в is_equal: равна false
```

- не универсальный способ;
- для примера:

<https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/>

Числа: взаимные неявные преобразования.

- При работе с числами язык C++ осуществляет неявные преобразования целых значений в действительные и наоборот.
- Любое действительное число преобразуется в целое путём отбрасывания всей дробной части
- Любое целое значение преобразуется в действительное путём добавления нулей в дробную часть
- Если в арифметическом **операторе** есть хотя бы одно действительное число, результат оператора будет преобразован к действительному типу
- Если в арифметическом **операторе** операндами являются беззнаковое и знаковое целые значения, то число со знаком приводится к беззнаковому и результат тоже будет беззнаковый

Числа: использование в арифметических выражениях.

```
1 int five = 5, seven = 7;  
2 double expr = five / seven + 4.25;  
3 printf("expr = %f\n", expr);
```

Числа: использование в арифметических выражениях.

```
1 int five = 5, seven = 7;  
2 double expr = five / seven + 4.25;  
3 printf("expr = %f\n", expr);
```

Вывод на консоль:

```
expr = 4.250000
```

Типы, переменные, операторы

Числа: явные преобразования.

- использование типа + круглых скобок
- использование оператора **static_cast**

```
1 int five = 5, seven = 7;  
2 double expr2 = double(five) / seven + 4.25;  
3 printf("expr2 = %f\n", expr2);  
4  
5 double expr3 = static_cast<double>(five) / 7  
6               + 4.25;  
7 printf("expr3 = %f\n", expr3);
```

Вывод на консоль:

expr2 = 4.964286

expr3 = 4.964286

Далее - управляющие конструкции

Условный переход: общий синтаксис

```
if ( <логическое выражение> ) {  
    <набор инструкций>;  
} [else {  
    <набор инструкций>;  
}]
```

```
1 int max_score = 3, min_score = 5;  
2  
3 if ( max_score < min_score ) {  
4     printf("Не должно так быть\n");  
5 } else {  
6     printf("Мы этого и ждали\n");  
7 }
```

Управление ходом выполнения программы

Условный переход: отсутствие пары фигурных скобок

```
1 int max_score = 8, min_score = 2;
2 // Опасный синтаксис:
3 if ( max_score < min_score )
4     printf("Не должно так быть\n");
5     printf("Я не отношусь к условному переходу\n");
6
7 int current_score = 4;
8 // Логическое выражение может быть составным
9 if ( (current_score >= min_score) and (↔
    current_score <= max_score) ) {
10     printf("Всё ок\n");
11 } else {
12     printf("Какая-то аномалия\n");
13 }
```

Условный переход: ещё пример

```
1 int current_score = 7, last_score = 4;
2
3 if ( last_score == 4 ) {
4     current_score += 4;
5 } else if ( last_score == 5 ) {
6     current_score += 1;
7 } else if ( last_score == 6 ) {
8     current_score -= 10;
9 } else {
10     current_score = 0;
11 }
```

Тернарный оператор ?:

<логическое выражение>

<?> <выражение, когда истинно>

<:> <выражение, когда ложно>;

```
1 int max_score = 13, min_score = 5;  
2  
3 int result;  
4 result = (max_score - min_score) < 5 ) ?  $\leftarrow$   
    min_score * 2 : max_score / 2;
```


Конструкция **switch**

```
switch (<выражение>) {  
    case <значение_1>:  
        [инструкции]  
        <break>;  
    case <значение_2>:  
        [инструкции]  
        <break>;  
    ...  
    case <значение_N>:  
        [инструкции]  
        <break>;  
    [default:  
        [инструкции]  
    ]  
}
```

```
1 int rate, score;
2 // Вычисляем rate
3
4 switch ( rate ) {
5     case 2:
6         score = 1;
7         break;
8     case 5:
9         score = 2;
10        break;
11    case 8:
12        printf("Выпала восьмёрка!\n");
13        score = 3;
14        break;
15    default:
16        score = 5;
17 }
```

Управление ходом выполнения программы

```
1 score = 4;
2
3 switch ( score ) {
4     case 4:
5         printf("Вполне неплохо.\n");
6         // не ставим *break*
7     case 5:
8         printf("Совсем здорово!\n");
9         break;
10    case 6:
11        printf("Недостигаемая высота!\n");
12        break;
13    default:
14        printf("Недотянули.");
15 }
```

В примере, когда в переменной **score** будет четвёрка, то будут выполнены строки **5** и **8**

- Фактически **switch** производит сопоставления результата выражения с значениями в "ветках" **case**
- Его недостатком, во-первых является то, что результат выражения должен быть приводим к числу
- А во-вторых - необходимость не забыть поставить **break** в каждой ветке **case**

Цикл

Под **циклом** в программировании понимается обособленный набор повторяющихся N раз инструкций. Причём, $N \geq 0$

Дополнительные определения

- **Тело цикла** - набор повторяющихся инструкций
- **Итерация** (или проход цикла) - однократное выполнение всех инструкций из тела цикла

Циклы **while** и **do ... while**

```
while (<логическое выражение>) {  
    [инструкции];  
}
```

```
do {  
    [инструкции];  
} while (<логическое выражение>);
```

Цикл **do ... while**: печать квадратов чисел от 0 до 9 включительно, возрастающий порядок

```
1 int counter = 0;
2
3 do {
4     printf("%i\n", counter * counter);
5     counter++;
6 } while ( counter < 10 );
```

Цикл **while**: печать квадратов чисел от 10 до 1 включительно, убывающий порядок

```
1 int upper_limit = 10;
2 int multiplier = 1;
3
4 while ( multiplier <= upper_limit ) {
5     printf("2 x %d = %d\n", multiplier,
6           2 * multiplier);
7     multiplier++;
8 }
```


Бесконечный цикл на примере **while**

```
1 while( true ) {  
2     printf( "*" );  
3 }
```

Цикл **for**

```
for (    [инициализация];  
      [условие выхода];  
      [итеративное изменение]  
    ) {  
    [инструкции];  
}
```

- ❶ **Инициализация:** как правило, задание начальных значений переменных, которые будут использованы внутри цикла. Или определение новых.
- ❷ **Условие выхода:** логическое выражение, проверяемой **до начала** и после каждой итерации.
- ❸ **Итеративное изменение:** задание выражений (изменение счётчика цикла, как пример), которые выполняются **после каждой итерации**. Перечисляются через запятую.

Цикл **for**

```
1 #include <cmath>
2
3 // Вывод значений квадратного корня для ←
  чисел от 0 до 10
4 for (int i = 0; i <= 10; i++) {
5     printf("square root of %d is %f\n",
6           i, sqrt(i));
7 }
```

Цикл **for**

```
1 // Бесконечный цикл
2 for (;;) {
3     for (int i = 0; i <= 3; i++) {
4         switch (i) {
5             case 0:
6                 printf("\r-"); break;
7             case 1:
8                 printf("\r\\"); break;
9             case 2:
10                printf("\r|"); break;
11             case 3:
12                printf("\r/"); break;
13         }
14     }
15 }
```

Управление циклами **continue** и **break**

```
1 for (int counter = 0; ; ++counter) {  
2     if ( (counter % 2) == 0 ) {  
3         printf("%i\n", counter);  
4         continue;  
5     }  
6  
7     if ( (counter > 15) ) {  
8         break;  
9     }  
10  
11     printf("Итерация закончена\n");  
12 }
```