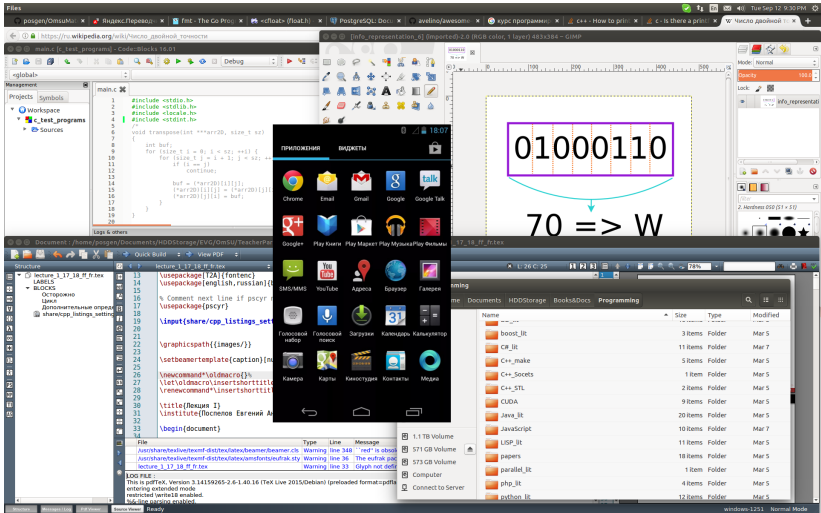


Лекция I

15 сентября 2017

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
1100110011010101010110001101010111010011
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

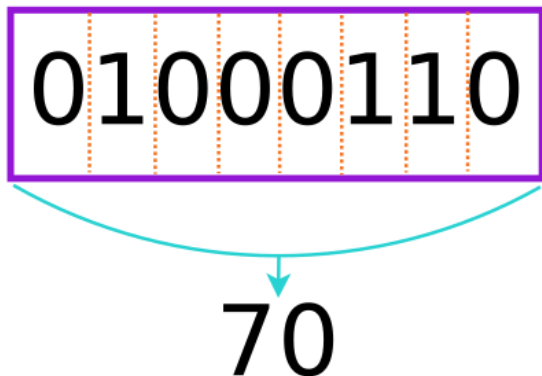
```
0101011101010100100101001000100001001001
0111001010101010010101001110010010010010
101010101010101011111010010011101101101010
0001110110100101101010101011010101010101
010101110110101010101010101010010101001001
1100110011010101010110001101010111010011
0101010100001101011100010100010100010101
```

Работа с информацией: что видит компьютер?

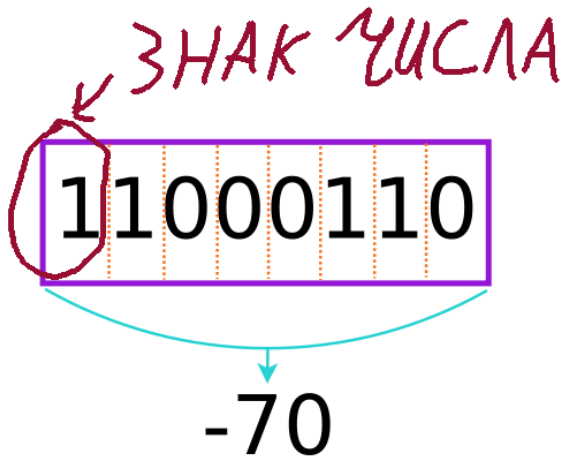
Минимальным блоком в современных ЭВМ является **байт**. На всех популярных ОС 1 байт состоит из 8 бит



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

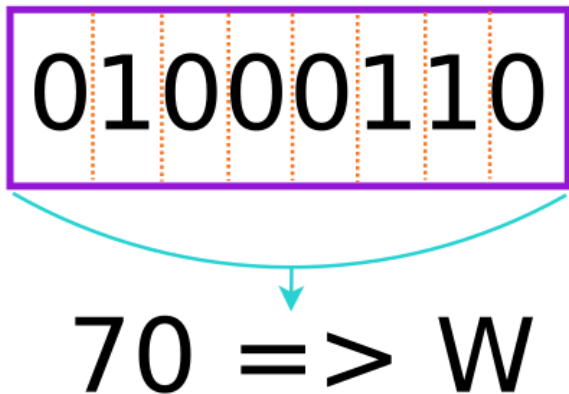


Целое число со знаком: первый бит отвечает за знак,
диапазон теперь - $[-128; 127]$



Работа с информацией: интерпретация байта

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



Переходим к конкретному языку
программирования

- Является **императивным, компилируемым, статически типизированным**
- Является мультипарадигменным: позволяет писать программы в **процедурном, объектно-ориентированном** и **обобщённом** стилях.
- Chrome, Firefox, Opera, Safari, IE Edge
- Используется для прикладных программ:
создания/редактирование изображений, работа с видео,
создание графических интерфейсов
- А также в науке: <https://root.cern.ch/> - набор библиотек для расчётов от ЦЕРНа

- Опять 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 Простые типы данных, переменные и операторы для работы с ними
- 2 Управление ходом выполнения программы (циклы, условные и безусловные переходы)
- 3 Способ обособления блоков кода, для многократного использования (функции)
- 4 Конструкции языка для создания пользовательских типов данных

Классификация типов данных

↙
Фундаментальные
(простые)

↓
Специальные

↘
Составные

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 конструкций), которые не могут быть использованы в качестве идентификаторов. Такие слова называются **ключевыми**, список:

**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**

1. Типы, переменные, операторы

Общий вид объявления и определения 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 простыми типами данных.

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

1. Простые 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

1. Простые типы, переменные, операторы

Целочисленные типы: объявление переменных

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

- 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

1. Простые типы, переменные, операторы

Целочисленные типы: присвоение значений

```
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. В этом случае,  
8 говорят об её определении (инициализации)  
9 */  
10 int good_rate = 16, bad_rate{-10}, karma;  
11 karma = good_rate + bad_rate;
```

Прежде, чем идти дальше

Выше показан пример использования оператора присвоения `=`, который используется для записи значений в переменную. Под *оператором* понимается некоторый символ (или специальная конструкция из символов), который производит действие 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.

Исключение

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

Кроме того, в примере 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**.

1. Простые типы, переменные, операторы

Целочисленные типы: 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;
```


1. Простые типы, переменные, операторы

Целочисленные типы: расширенные операторы присваивания

```
1 int balance = 5, rate = 10, total = 15;
2 // Оператор присваивания сначала вычисляет ↵
   правую часть
3 balance = balance + 1;
4 // Сокращённая запись
5 balance += 1;
6 // Также определены: -=, *=, /=, %=
7
8 // Инкремент/Декремент – увеличение/↵
   уменьшение значения переменной на единицу
9 rate++; // rate стал равным 11
10 ++rate; // –//– 12
11 rate--; // снова 11
12 --rate; // теперь 10
```

1. Простые типы, переменные, операторы

Целочисленные типы: тонкости инкремента/декремента

```
1 // Разница пре- и пост- инкремента
2 int balance = 5, total = 5;
3
4 cout << balance++ << endl; // напечатает на ←
    экране 5
5 // значение balance равно 6
6
7 cout << ++total << endl; // напечатает 6
8 // значение total равно 6
9
10 // Пример унарных операторов
11 int number = -6;
12 +number;
13 -number;
```

1. Простые типы, переменные, операторы

Целочисленные типы: побитовые операции

```
1 unsigned number = 4, next_number;  
2  
3 // Побитовый сдвиг вправо на 2 позиции  
4 next_number = number << 2;  
5 // Побитовый сдвиг влево на 3 позиции  
6 next_number = number >> 3;  
7 // Побитовое "И"  
8 next_number = number & 2;  
9 // Побитовое "ИЛИ"  
10 next_number = number | 3;  
11 // Побитовое исключающее "ИЛИ" (xor)  
12 next_number = number | 3;  
13  
14 // Расширенное присваивание  
15 next_number >>= 1;
```

1. Простые типы, переменные, операторы

Целочисленные типы: оператор **sizeof**

```
1 int i_num = 0b01100010;
2 long l_num = 0345;
3 size_t sz_num = 0xff11c;
4
5 cout << "Размер int: " << sizeof(int);
6 cout << "\nРазмер long: " << sizeof(l_num);
```

1. Простые 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  
8 // Отрицание:  
9 result = !falsey; // result равен true
```

1. Простые типы, переменные, операторы

```
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 |

1. Простые типы, переменные, операторы

Логический тип: совместимость с языком C

```
1 bool result;  
2  
3 result = 0;  
4 // result равен false  
5  
6 result = 1;  
7 // result равен true
```

1. Простые 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
```


1. Простые типы, переменные, операторы

Действительные числа или числа с плавающей запятой

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

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

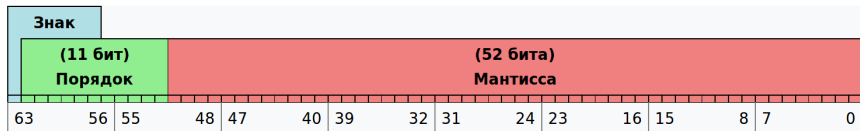
long double - двойной точности (≈ 15 -16 знаков после запятой), расширенный диапазон: 10^{4932} .

Стандартом языка определяется только относительная зависимость размеров: `sizeof(float) <= sizeof(double) <= sizeof(long double)`

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

1. Простые типы, переменные, операторы

Действительные числа: как представлены в памяти



Конкретное число вычисляется как (общая идея):

$$(-1)^{\text{знак}} \times 1.\text{Мантисса} \times 2^{\text{Порядок}}$$

Что интересно, формат чисел с плавающей запятой стандартизирован: IEEE 754.

В точности числа не учитывается десятичная экспонента:

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

1. Типы данных, переменные, операторы

Действительные числа: 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 1
```

1. Простые типы, переменные, операторы

Действительные числа: 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 1
```

1. Простые типы, переменные, операторы

Действительные числа: сравнение

Предупреждение

Сравнение действительных чисел неоднозначно

```
1 double first_rate = 0.4, second_rate = 0.4;  
2  
3 int is_equal = (first_rate == second_rate);  
4 // что будет в is_equal — непонятно
```

1. Простые типы, переменные, операторы

Действительные числа: подход в сторону правильного сравнения

```
1 #include <cmath>
2
3 double first_rate = 0.4, second_rate = 0.8 /↵
   2;
4
5 // Определяем для себя приемлемую точность
6 double eps = 0.0000001;
7
8 // abs — вычисляет модуль аргумента
9 int is_equal = ( abs(first_rate - ↵
   second_rate) < eps );
10 // есть уверенность в is_equal
```

1. Простые типы, переменные, операторы

Действительные и целые числа: взаимные 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 выражении есть хотя бы одно действительное число, результат выражения будет преобразован к его типу

1. Простые типы, переменные, операторы

Действительные и целые числа: взаимные преобразования.

```
1 double rate1 = 4.57;
2 int main_part = rate1;
3 // Здесь main_part равна 4
4
5 main_part += 3;
6 rate1 = main_part;
7 // rate1 примерно равен 7.0
```


2. Управление ходом выполнения программы

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     cout << "Как-то странно\n";  
5     cout << "эта инструкция не относится к ←  
        оператору ветвления";
```

2. Управление ходом выполнения программы

Условный переход: пример

```
1 int max_score = 3, min_score = 5;
2 // Так гораздо понятнее:
3 if ( max_score < min_score ) {
4     cout << "Как-то странно\n";
5 }
6 cout << "Эта инструкция не относится к ←
    оператору ветвления";
7
8 int current_score = 4;
9 if ( (current_score >= min_score) && (←
    current_score <= max_score) ) {
10     cout << "Всё ок\n";
11 } else {
12     cout << "Какая-то аномалия\n";
13 }
```

2. Управление ходом выполнения программы

Условный переход: ещё пример

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

2. Управление ходом выполнения программы

Тернарный оператор ?:

<логическое выражение>

<?> <выражение, если истинно>

<:> <выражение, когда ложно>;

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

2. Управление ходом выполнения программы

Конструкция **switch**

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

2. Управление ходом выполнения программы

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

2. Управление ходом выполнения программы

- Фактически **switch** производит сопоставления результата выражения с значениями в "ветках" **case**
- Его недостатком является то, что результат выражения должен быть приводим к числу

2. Управление ходом выполнения программы

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

2. Управление ходом выполнения программы

Циклы **while** и **do ... while**

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

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

2. Управление ходом выполнения программы

Цикл **while**

```
1 int counter = 10;  
2  
3 while ( counter > 0 ) {  
4     cout << counter * counter << "\n";  
5     counter--;  
6 }
```

2. Управление ходом выполнения программы

Цикл **do ... while**

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

2. Управление ходом выполнения программы

Бесконечный цикл на примере **while**

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

2. Управление ходом выполнения программы

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.

2. Управление ходом выполнения программы

Цикл **for**

```
1 #include <cmath>
2
3 // Бесконечный цикл
4 for (;;) {
5     // что-то полезное
6 }
7
8 // Вывод значений квадратного корня для ↵
   чисел от 0 до 9
9 for (int i = 0; i < 10; i++) {
10     cout << sqrt(i) << "\n";
11 }
```

2. Управление ходом выполнения программы

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

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

Где получить информацию о C++?

Литература

- Бьярне Страуструп "Программирование. Принципы и практика с использованием C++ 2016 г., Вильямс
- Татьяна Павловская, "C/C++. Процедурное и объектно-ориентированное программирование 2014 г., Питер

Альтернатива онлайн

- <http://en.cppreference.com/w/cpp/language> (
<http://ru.cppreference.com/w/cpp/language>)
- <http://www.cplusplus.com/reference/>
- <https://github.com/posgen/OmsuMaterials>