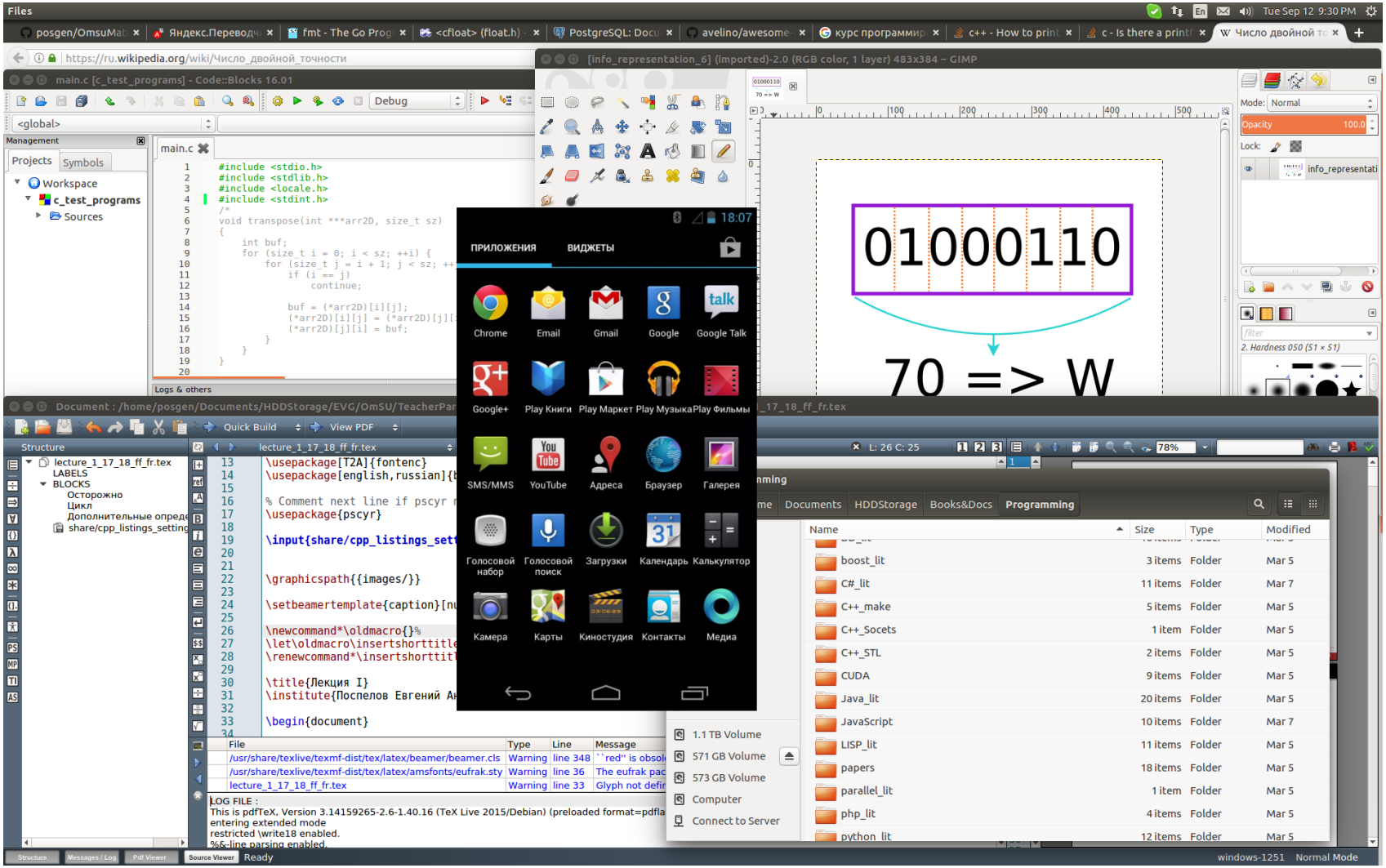
Лекция I

Поспелов Евгений Анатольевич

13 сентября 2017

До изучения языка программирования: взгляд снизу на типы данных

Работа с информацией: что видим мы?



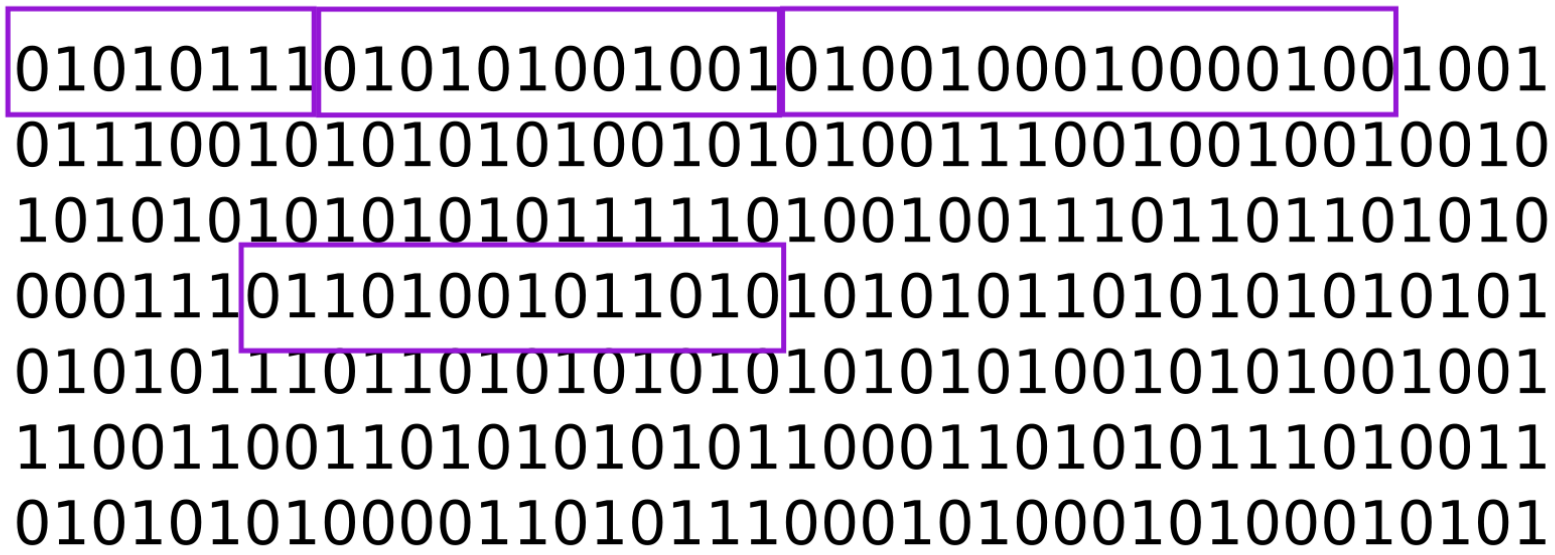
Работа с информацией: что видит компьютер?

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

```
01010111010101001001010010001000010010010
01110010101010100101010011100100100100010
10101010101010111110100100111011011010100
00011101101001011010101010110101010101010
01010111011010101010101010010101001001010
11001100110101010101100011010101110100110
01010101000011010111000101000101000010101
```

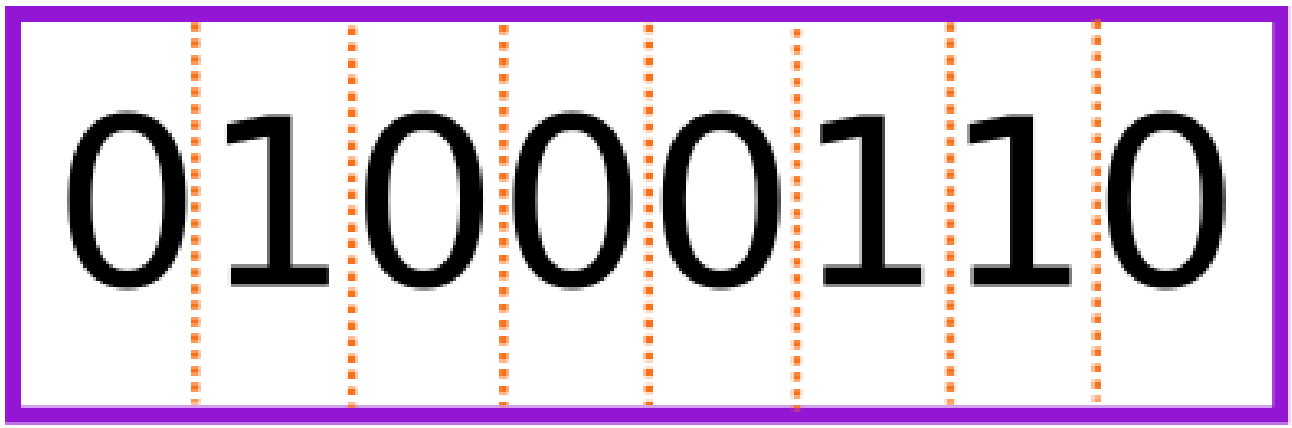
Работа с информацией: что видит компьютер?

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



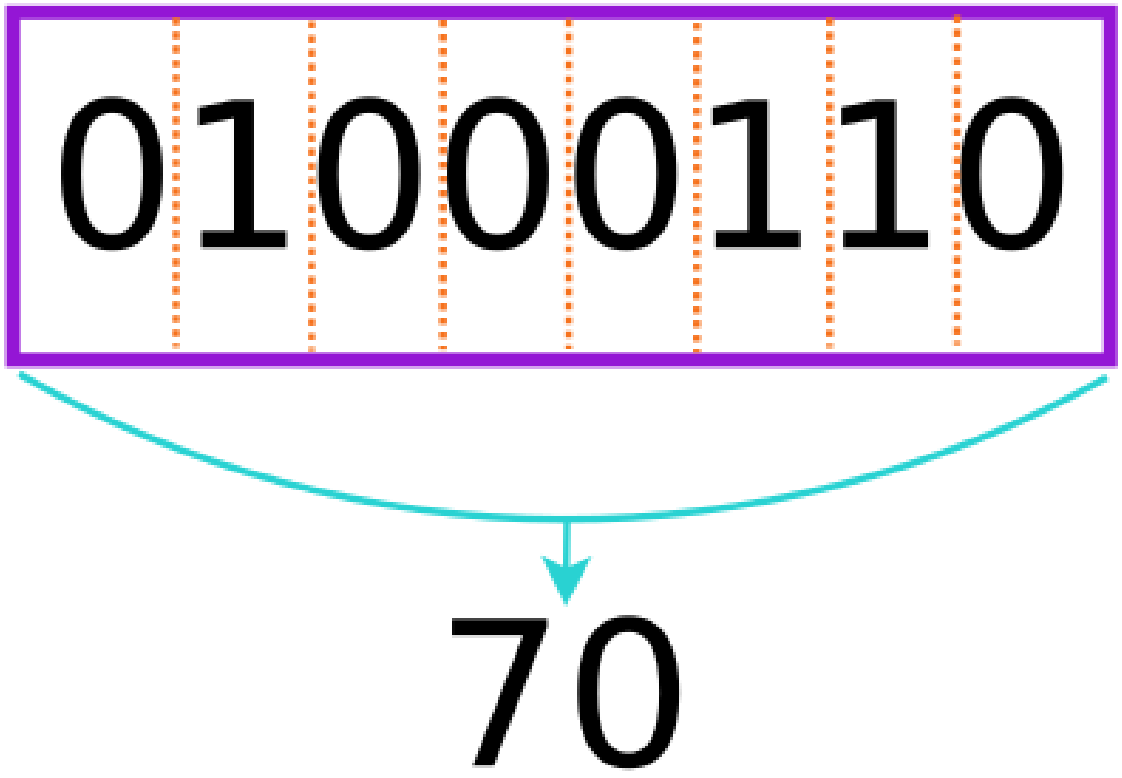
Работа с информацией: что видит компьютер?

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



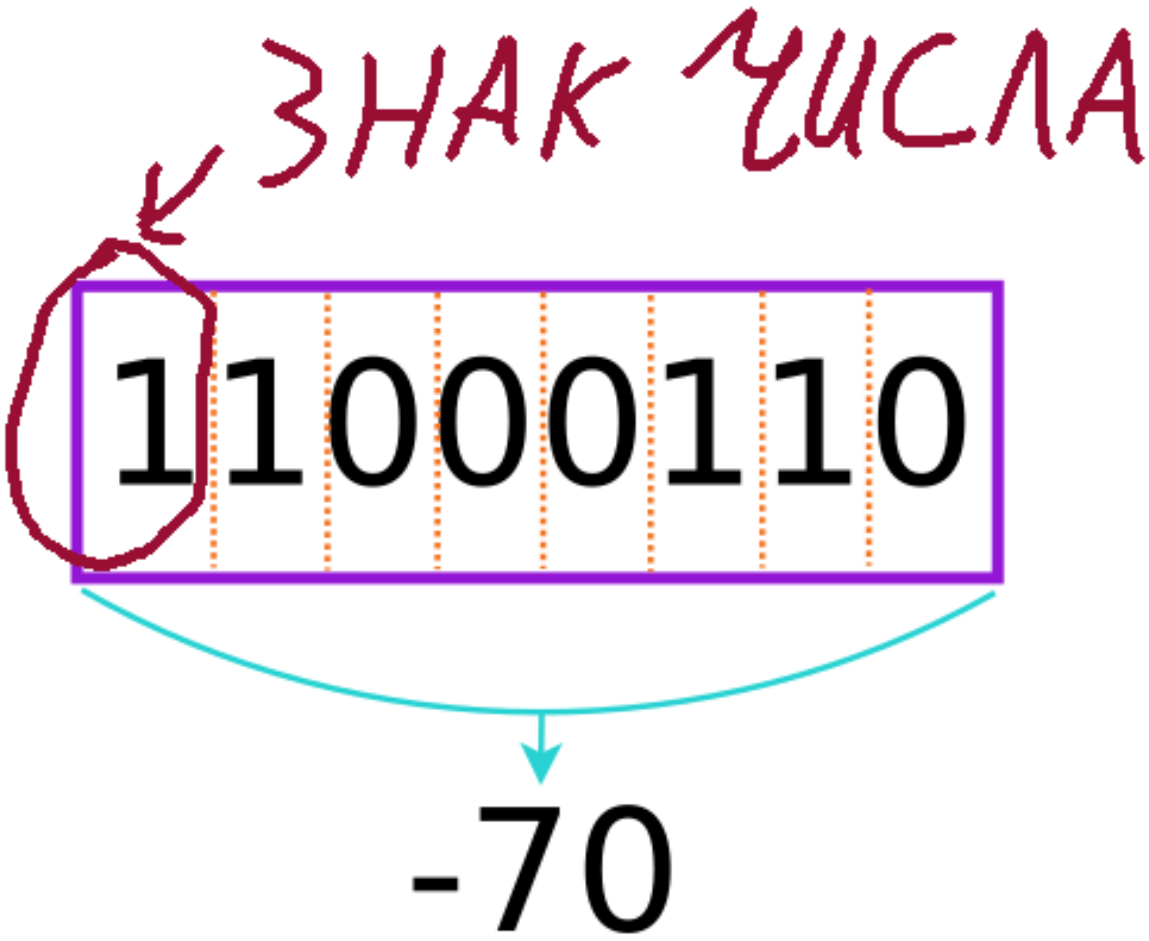
Работа с информацией: интерпретация байта

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



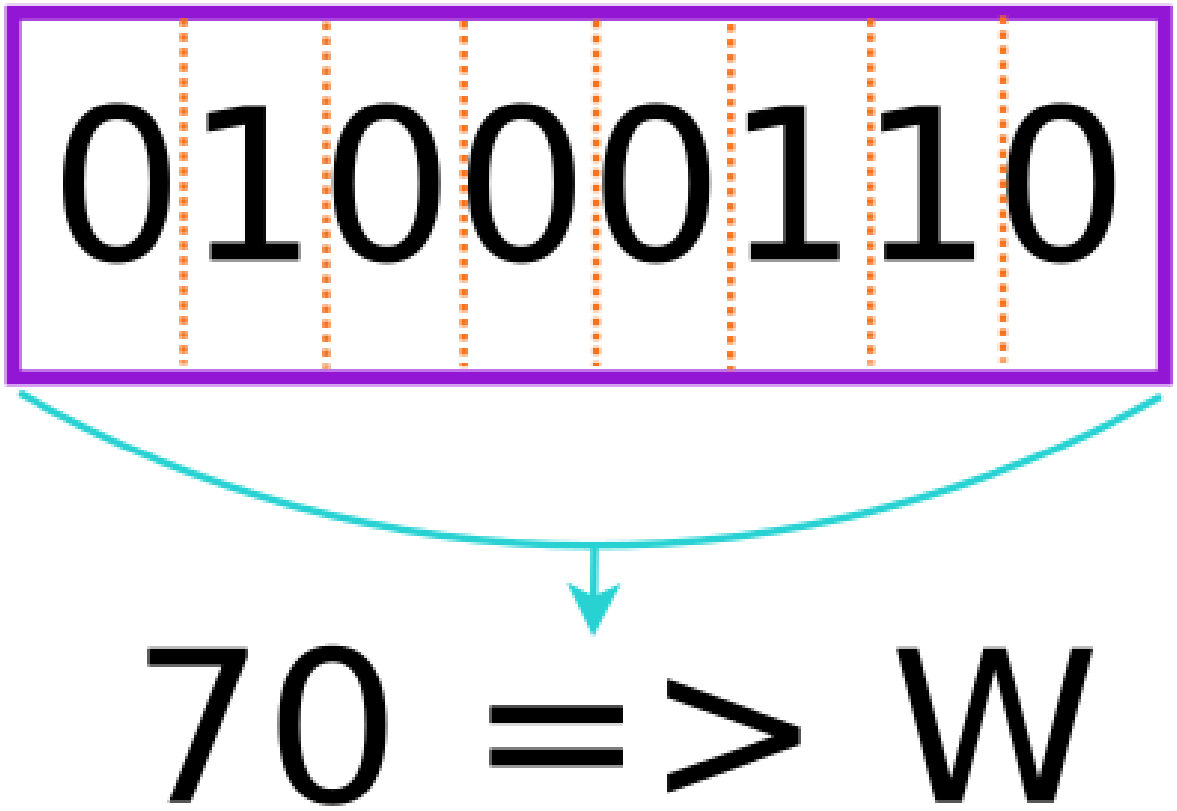
Работа с информацией: интерпретация байта

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



Работа с информацией: интерпретация байта

Некоторый символ: вводится некоторая таблица соответствия между числом и текстовым символом



Переходим к конкретному языку программирования

Язык программирования C

- Читается как "Си"
- Является **императивным**, **компилируемым**, **статически типизированным**
- Реализует **процедурную** парадигму программирования: выделение кода в *логически* отдельные фрагменты происходит только с помощью определения функций.
- Все современные ведущие ОС написаны на C (подходит для системного программирования)
- Используется для прикладных программ: создания/редактирование изображений, работа с видео, создание графических интерфейсов и других задачах
- А также в науке: **https://software.intel.com/en-us/mkl**

Язык программирования C: историческая справка

- Деннис Ритчи, Bell Labs: 1973 - публикация языка C

- Разработан для упрощения написания операционных систем
- 1989 - первый стандарт ANSI C89 (ISO C90)
- 2011 - текущий стандарт ISO C99

Какие особенности определяют язык программирования?

1. 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 (определение пользовательских функции)
4. Возможность создания пользовательских типов данных

Прежде, чем идти дальше

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**. Для языка C их список следующий:

auto break case char const continue default do double
else enum extern float for goto if int long register
return short signed sizeof static struct switch typedef
union unsigned void volatile while inline
_Bool _Complex _Imaginary

Общий вид объявления и определения переменной:

```
[...] <тип> <идентификатор> [= <значение>];
```

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

Прежде, чем идти дальше

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

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

Предупреждение. Многострочные комментарии не могут быть вложенными

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

- Слово **int** можно пропускать при использовании **short, unsigned, long** типов
- Стандарт языка не определяет конкретного размера каждого типа, он определяет только соотношение между ними: sizeof(**short**) <= sizeof(**int**) <= sizeof(**long**) <= sizeof(**long long**)

Целочисленные типы данных: 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;
```

Целочисленные типы данных: 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 объявлениии переменной. В этом случае,
8    говорят об её определении (инициализации)
9  */
10 int good_rate = 6, bad_rate = -10, karma;
11 karma = good_rate + bad_rate;
```

Прежде, чем идти дальше

Выше показан пример использовани оператора присвоения =, который используется для записи значений в переменную. Под *оператором* понимается некоторый символ (или специальная конструкция из символов), который производит действие. Для совершения которого оператору нужен один или более объект. Объектом в C выступает, почти во всех случаях, переменная. 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 расмотрен в разделе о ходе выполнения программы.

Прежде, чем идти дальше

Кроме того, в примере с привоением значений переменным были использованы конкретные числа (-7, 6, -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 сначало 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**.

Целочисленные типы данных: 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;
```

Целочисленные типы данных: расширенные операторы присваивания

```
1  int balance = 5, rate = 10, total = 15;
2  // Оператор присваивания сначало 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
```

Целочисленные типы данных: тонкости инкремента/декремента

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

Целочисленные типы данных: побитовые операции

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

Целочисленные типы данных: оператор **sizeof**

```
1 int i_num = 0b01100010;
2 long l_num = 0345;
3 size_t sz_num = 0xff11c;
4
5 printf("Размер int: %ld", sizeof(int));
6 printf("Размер long: %ld", sizeof(l_num));
```

Операторы сравнения

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

Все операторы сравнения в C возращают значения типа **int**: **1**, если условие выполняется и **0** - в противоположном случае.

```
1 int width = 5, height = 4;
2 int status = width > height;
3 // status равен 1
4
5 status = (width == height);
6 // status равен 0
```

Логические операторы

Операция	Оператор
отрицание	!
логическое И	&&
логическое ИЛИ	||

Логические операторы в C также возвращают **1** или **0**. При этом любое выражение, результат которого отличен от нуля трактуется как **истинное значение**, а нуль - как **ложное значение**.

```
1 int rate1 = 5, rate2 = 4, status;
2
3 status = !rate1;
4 // status равен 0
5
6 status = !rate2;
7 // status равен 1
```

```
1 int var1, var2;
```

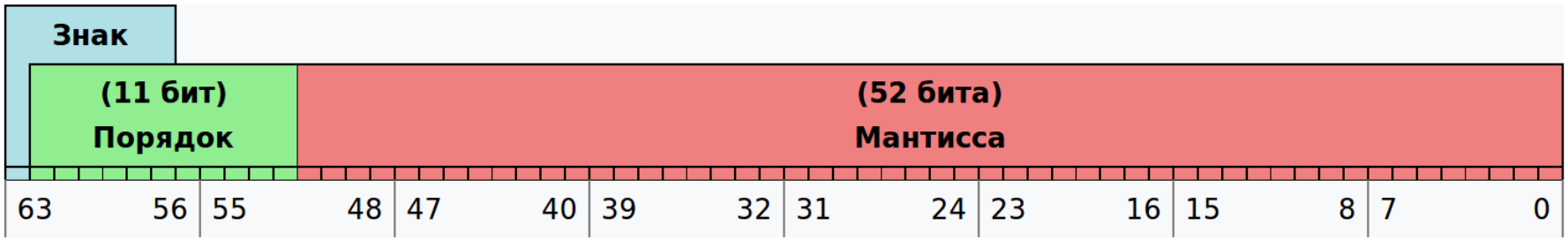
Таблица истинности

	var1	var2	Результат
&&	$!=0$	$!=0$	1
&&	$!=0$	$==0$	0
&&	$==0$	$!=0$	0
&&	$==0$	$==0$	0
||	$!=0$	$!=0$	1
||	$!=0$	$==0$	1
||	$==0$	$!=0$	1
||	$==0$	$==0$	0

Действительные числа или числа с плавающей запятой
float - действительное число одинарной точности ($\approx$ 7-8 знаков после запятой). Диапазон значений порядка 10^{38}.
double - действительное число двойной точности ($\approx$ 15 знаков после запятой). Диапазон значений порядка 10^{308}.
long double - двойной точности ($\approx$ 15 знаков после запятой), расширенный диапазон: 10^{4932}.
Размер типов зависит от компилятора и ОС, стандартом языка определяется только относительная зависимость: sizeof(**float**) <= sizeof(**double**) <= sizeof(**long double**)

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

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

Действительные числа: на нуль делить разрешается

```
1 #include <math.h>
2
3 double super_rate = 5.5;
4 super_rate /= 0.0;
5
6 int is_infinity = isinf(super_rate);
7 // is_infinity равна 1
```

Действительные числа: понятие Not-A-Number (NaN)

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

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 == first_rate);
4 // что будет в is_equal — непонятно
```

Действительные числа: подход в сторону правильного сравнения

```
1 #include <math.h>
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 // fabs – Вычисляет модуль аргумента
9 int is_equal = ( fabs(first_rate - ↩
     second_rate) < eps );
10 // есть уверенность в is_equal
```

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

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
```

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("Как-то странно\n");
5 printf("эта инструкция не относится к ↩
    оператору ветвления");
```

Условный переход: пример

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

Условный переход: ещё один 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 }
```

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; )
```

Конструкция **switch**

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

```
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     printf("Выпала восьмёрка!\n");
12     score = 3;
13     break;
14   default:
15     score = 5;
16 }
```

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

Что можно почитать?

Литература

- В.В. Подбельский, С.С. Фомин, "Курс программирования на языке Си 2015 г., ДМК-Пресс

Альтернатива онлайн

- https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf
- http://en.cppreference.com/w/c/language (http://ru.cppreference.com/w/c/language)
- http://www.cplusplus.com/reference/
- https://github.com/posgen/OmsuMaterials