Лекция 2

Введение в C/C++.

Системы счисления

Базовые типы данных

Важен не язык - важно умение мыслить!
Языки программирования по синтаксису и семантике во многом похожи!
Ищите смысл и проникайтесь алгоритмами, а не непонятно как работающими «заготовками»!

```
for (int i = 0; i < n; i++)  s += i; // C/C++
```

Изотерика :–)

Тернарная операция **?:** выполняется над тремя операндами

```
1   int max = a > b ? a : b;
2   ++i + ++i;  //при i=1, выражение =6!
3   (x = 7) + (c = 10 + x);
4   ++(n = (k = 5) % 2);
5   if (-1 > (unsigned int) 1)  *p++;
```

https://neolurk.org/wiki/%2B%2Bi_%2B_%2B%2Bi

Истина!!!

Основные элементы ЯП

Алфавит – набор символов или групп символов, рассматриваемых как единое целое, с помощью которых составляется текст программы.

Оператор (инструкция, *statement*) – основная конструкция ЯП, определяющая конкретное действие.

Синтаксис – правила построения конструкций ЯП с помощью алфавита и операторов языка.

Семантика – смысл и правила использования этих конструкций.

Найдите ошибки:

```
#include <stdio.h>
int main()
{  int a, b;
   scanf("%d %d", a, b);
   printf("a = %d, b = %d\n", a, b);
   return 0;
}
```

→ `scanf("%d %d", &a, &b);`

```
int main()
{  int a, b;
   printf("a = %d, b = %d\n", a, b);
   scanf_s("%d %d", &a, &b);
   return 0;
}
```

поменять местами

```
printf("\n%c, %d, %f, %s", x, y, z);
```

Как создаются исполняемые файлы с расширением .exe

- Создание исходного файла с помощью программы текстового редактора **name.cpp**
- Препроцессорная обработка – программа передается препроцессору, который выполняет директивы, содержащиеся в ее коде (***директивы начинаются со знака #***)
- Компиляция (лексический, синтаксический, семантический анализ) - перевод исходного модуля в объектную программу на машинном языке или языке низкого уровня. **name.obj**
- Компоновка (сборка) программы – редактор связей связывает все объектные модули (файлы) в исполняемую программу (загрузочный модуль) **name.exe**
- Загрузка (запуск) программы на выполнение – загрузочный модуль программы с помощью программы загрузчика помещается в оперативную память
- Выполнение программы

Этапы компиляции и компоновки на C/C++

Можно в литературе более подробно ознакомиться с процессом компиляции программ

Он включает следующие этапы:

- Лексический анализ. Последовательность символов исходного файла преобразуется в последовательность лексем
- Синтаксический анализ. Последовательность лексем преобразуется в дерево разбора
- Семантический анализ
- Оптимизация
- Генерация кода

Основные этапы решения задачи на ЭВМ

1. Постановка задачи (разработка ТЗ)
2. Выбор метода решения
3. Разработка алгоритма решения задачи

Свойства алгоритма:

- Массовость
- Понятность
- Правильность
- Результативность

Не верно

Верно

C	
`x=1;`	`scanf("%d",&x);`
C++	
`x=1;`	`cin >> x;`

4. Разработка алгоритма в заданной среде программирования (IDE)
5. Отладка и тестирование программы
6. Анализ полученных результатов

Вопрос*: *Что такое

тестирование – процесс выявления ошибок;

отладка – поиск места ошибки с целью ее исправления.

Тестирование ПО – его исследование с целью получения информации о качестве продукта.

Набор тестов ⟶ сравнение результатов с эталоном с целью выявления ошибок.

```
//какой тест выявит ошибку в программе?
int main()
{  int a, b;
   scanf_s("%d %d", &a, &b);  //Ввод в b ноль
   printf("\n a/b = %d ", a/b);
   return 0;
}
```

Самая дорогая точка в программировании

Космический аппарат США Маринер-1 22.07.1962 должен был направиться к Венере, но был уничтожен из-за аварии через 293 секунды после старта.

Фрагмент кода на Фортране

DO 17 I = 1, 10 **DO 17 I = 1.10**

Скрытая ошибка!!!

Код интерпретирован как **DO17I = 1.10**,
т.е. переменной DO17I (пробельные символы языком не учитываются) присвоить значение 1.10

Необходимо быть уверенным, что основные компоненты программной системы реализованы правильно и не подведут в критический момент. Для этого при проектировании выполняется верификация и валидация системы.

Верификация – оценка, подтверждение корректности ПО аналитическими методами с целью гарантированности правильности программы, соответствия заданным условиям, ТЗ на ее проектирование. Сопровождает весь жизненный цикл ПО.

Валидация – процесс оценки того, насколько программа соответствует требованиям применения ее по назначению (требованиям заказчика).

Верификация задается вопросом, строим ли мы систему правильно, а **валидация** – строим ли мы правильную систему.

Идентификаторы

В общем виде **идентификатор** (сокращенно **id**) – это некоторое уникальное имя объекта.

В программировании **идентификаторы** – это имена переменных, констант, функций, пользовательских типов, классов и других объектов, определяемых пользователем.

```
number, x, summa, step_1
```

~~`1number, step*`~~

Идентификатор - это последовательность букв, цифр и символов подчёркивания, которая начинается с буквы или символа подчёркивания ("_") и не содержит пробелов.

Язык C/C++ чувствителен к регистру букв

`Summa` и **`summa`** - ***разные идентификаторы***.

Переменные и их типы

Переменная – именованная величина, значение которой может меняться.

- Имя переменной – идентификатор.
- Переменные должны быть объявлены до их использования.
- Переменные должны быть инициализированы до их использования.
- Строчные и прописные символы различаются

  `number ≠ Number`
- При объявлении переменной указывается ее **тип**

```
char ch;   //символьный 1 байт
int num;   //целый 4 байта
float y;   //вещественный 4 байта
```

Тип переменной определяет

- множество значений, которые она может принимать
- размер памяти для хранения значений данного типа
- множество операций, применимых к типу

```
'A'/'B' == 0
'B'/'A' == 1
```

Базовые (встроенные) типы данных

- Целый - `int`
- Символьный - `char`
- Вещественный (числа с плавающей точкой) - `float`
- Логический - `bool`

Кроме ***базовых (встроенных) типов данных***, можно создавать свои типы данных – ***пользовательские типы данных***.

Диапазон чисел типа `int` зависит от компьютера, на котором выполняется программа.

Когда процессоры были 16-битными, `int` было 2 байта.

На 32-разрядных и 64-разрядных системах `int` занимает 4 байта

```
printf("%d", sizeof(int));
```

Тип	байт	Диапазон принимаемых значений
целочисленный (логический) тип данных		
bool	1	true и false
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

Рассмотрим пример хранения значений типа short int

Это знаковый двухбайтовый тип данных, с помощью которого можно сохранить **? 2^{16}** значений (-32 768 ÷ 32 767)

$2^{16} = 65536$

Бит знака

-32768 . . 32767

32767

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

байт (8 бит) **байт**

-32768

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

-1

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Чему равно максимальное 2-х байтовое беззнаковое целое?

65535

Отрицательные числа хранятся в обратном коде

При вычислении выражения операнды приводятся к высшему типу, здесь к типу unsigned int (а это 4 байта)

```
if (-1 > (unsigned int) 1)  *p++;
```

1 байт приведется к 4х байтовому значению

4 байта

Как получить побитовое отрицательное значение?

1 → 00000001

-1 → 11111110 → (инвертируем)

\+ 1 → (добавляем 1)

11111111 → (значение -1, дополнительный код)

00000000000000000000000011111111 > 00000000000000000000000000000001

Истина!!!

Представление чисел в двоичной, восьмеричной и 16-ричной системах счислений

Память компьютера состоит из ячеек – разрядов. В разряд можно записать либо **0**, либо **1**. Разряд может хранить 1 бит информации. Но разряд слишком мал для работы с информацией.

1 байт – 8 бит	2^0
1 КБ – 1024 байт	2^{10}
1 МБ – 1024 КБ	2^{20}
1 ГБ – 1024 МБ	2^{30}
1 ТБ – 1024 ГБ	2^{40}
1 ПБ – 1024 ТБ	2^{50}
1 ЭБ – 1024 ПБ	2^{60}
1 ЗБ – 1024 ПЭ	2^{70}
1 ИБ – 1024 ЗБ	2^{80}

Терабайт петабайт эксабайт зеттабайт йоттабайт

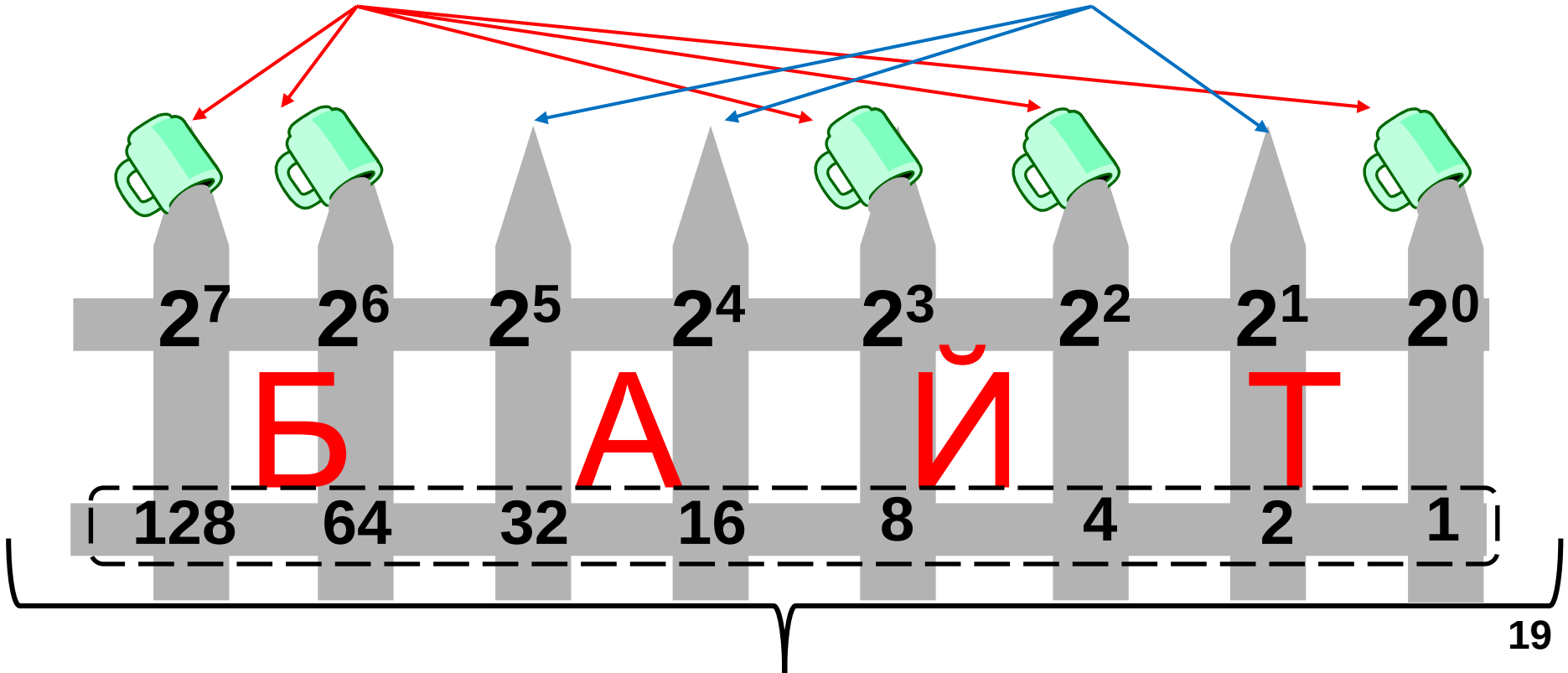
Байт - 8 бит

Нумерация битов в байте начинается с 0 справа налево

7	6	5	4	3	2	1	0
1	1	0	0	1	1	0	1

Разряды, в которых записана 1

Разряды, в которых записан 0



10 с.сч.	2 с.сч.	8 сист.сч.	16 сист.сч.
1	2^0 1	1	1
2	2^1 10	2	2
3	11	3	3
4	2^2 100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	2^3 1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C

10 с.сч.	2 с.сч.	8 сист.сч.	16 сист.сч.
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	2^4 10000	20	10

32	2^5	100000
64	2^6	1000000
128	2^7	10000000

! Нумерация битов в байте начинается с 0 справа налево

Заполнить таблицу:

10 сист.сч.	8 сист.сч.	16 сист.сч.	2 сист.сч.
15	17	F	0000 1111
100	144	64	0110 0100
303	457	12F	10 0101 0111

Максимальное беззнаковое однобайтное целое - . . . ?

255_{10}

1111 1111

377_8

1111 1111

FF_{16}

0111 1111 - . . . ?

$127_{10}, 177_8, 7F_{16}$

Максимальное беззнаковое двухбайтное целое?

65 535 **0xffff**

Максимальное беззнаковое 4-хбайтное целое?

4 294 967 295 **0xffffffff**

16-ричная система счисления гораздо проще!
Все ячейки памяти имеют адреса, и это именно 16-ричные значения

Д/З

Вывести на экран двоичный код числа.

Решение: например, целым переменным x1, x2, x3 присваиваем десятичное, восьмеричное или 16-ричные значения:

```
int x1 = 123;
int x2 = 0123;
int x3 = 0x123;
```

далее выводим их двоичное представление

Примеры описания переменных

```
char a = '+'; //инициализация переменной
int  i, j;
double eps = 0.001;
float f1, f2;
int  n(100); //второй способ инициализации
unsigned int weight;
bool k = true;
char FirstName[20];
```

- Имя переменной может содержать буквы, цифры и знак подчеркивания
- Имя переменной не может начинаться с цифры
- Большие и маленькие буквы различаются между собой

Спецификаторы формата printf(), scanf()

Специ-фикатор	Формат
%i, %d	ввод-вывод целого int
% c	символ
% s	строка символов
%e	число с плавающей точкой, экспоненциальная запись
%f	число с фиксированной точкой
%g	самая короткая запись вещественного числа
%u	десятичное целое без знака (unsigned)
%o	восьмеричное целое без знака (octal)
%x	шестнадцатеричное целое без знака (hexadecimal)
%p	Указатель (шестнадцатеричное целое без знака)

1. Целые константы

Десятичные целые	Восьмеричные целые	Шестнадцати-ричные целые
123, -123,	**0123,**	**0x123**

Выведем эти целые числа в формате decimal (%d) (десятичное):

```
printf("\n %d  %d  %d  %d\n", 123,-123,0123,0x123);
```

```
123    -123    83    291
```

```
printf("\n %d\t%x\t%o\n", 255,255,255);
```

```
Консоль отладки Microsoft Visua
255    ff    377
```

Эквивалентные представления целых однобайтовых чисел:

Десятичные со знаком	Десят. без знака unsigned	Восьмеричные	Шестнадцатиричные	Двоичные
15	15	017	0xF	00001111
16	16	020	0x10	00010000
1	1	1	0x1	00000001
-1	255	377	0xFF	11111111

```
signed char ch = -1;
printf("signed char ch= %hhd\n", ch);
printf("unsigned char ch= %hhu\n", ch);
```

```
signed char ch= -1
unsigned char ch= 255
```

- %d означает формат целого 4-х байтового значения int,
- модификатор h приводит целое к 2-х байтовому short int,
- модификатор hh – к однобайтовому целому

Максимальные значения целых констант

```
#include <limits.h>
#include <math.h>
 printf("%d \t %d\n", INT_MIN, INT_MAX);
```

```
-2147483648     2147483647
```

Диапазон стандартных целых чисел в C/C++
(int) от -2^{31} до $2^{31}-1$

```
printf("%d\t%d\n\n", (int)pow(-2,31), (int)pow(2,31));
```

```
-2147483648     -2147483648
```

Ошибка!

```
printf("%d\t%d\n\n", pow(-2,31), pow(2,31)-1);
```

```
-2147483648     2147483647
```

Верно!

```
printf("%f\t%f\n\n", pow(-2,31), pow(2,31));
```

```
-2147483648,000000     2147483648,000000
```

Верно! Float – большой диапозон

```
printf("%2.0f\t%2.0f\n\n", pow(-2,31), pow(2,31));
```

```
-2147483648     2147483648
```

Максимальные значения целых констант

```
#include <limits.h>
```

```
-2147483648     2147483647
```

```
printf("%d \t %d\n", INT_MIN, INT_MAX);
```

Диапазон стандартных целых чисел в C/C++ (тип int) от -2^{31} до $2^{31}-1$

```
int Sum()
{    int c = 0;
     for(int i=1; i<1000000; i++)
     c = c + 100000;
     return c;
}

int main()
{
  printf("\n%d\n", Sum());
  return 0;
}
```

При сложении целых может происходить переполнение, при этом не будет предупреждения о переполнении: программа продолжит работать с неверными данными.

```
#include <limits.h>
```

Содержит определения макросов

```
#define CHAR_BIT
#define SCHAR_MIN
#define SCHAR_MAX
#define UCHAR_MAX
#define CHAR_MIN
#define CHAR_MAX
#define MB_LEN_MAX
#define SHRT_MIN
#define SHRT_MAX
#define USHRT_MAX
#define INT_MIN (-2147483647-1)
#define INT_MAX
#define UINT_MAX
#define LONG_MIN
#define LONG_MAX
#define ULONG_MAX
#define LLONG_MIN
#define LLONG_MAX
#define ULLONG_MAX
```

ПРАКТИКА: Как посмотреть определение **INT_MIN**, типов, код реализации функций?

1. Подводим курсор– всплывает подсказка

```
printf("%d \t %d\n", INT_MIN, INT_MAX);

unsigned char num, mask = 1
printf("\nNumber: \n");
scanf("%hhc", &num);
```

#define INT_MIN (-2147483647 - 1)

Расширяется в:(-2147483647 - 1)

Поиск в Интернете

2. Выделить и правый клик – всплывает подсказка – выбрать показать определение (Alt + F12)

```
int main()
{
    printf("%d \t %d\n", INT_MIN, INT_MAX);
```

```
34      #define USHRT_MAX      0xffff
35      #define INT_MIN        (-2147483647 - 1)
36      #define INT_MAX          2147483647
37      #define UINT_MAX         0xffffffff
38      #define LONG_MIN       (-2147483647L - 1)
39      #define LONG_MAX         2147483647L
40      #define ULONG_MAX        0xffffffffUL
41      #define LLONG_MAX        9223372036854775807i64
42      #define LLONG_MIN      (-9223372036854775807i64 - 1)
43      #define ULLONG_MAX       0xffffffffffffffffui64
44
45      #define _I8_MIN        (-127i8 - 1)
46      #define _I8_MAX          127i8
47      #define _UI8_MAX         0xffui8
48
49      #define _I16_MIN       (-32767i16 - 1)
50      #define _I16_MAX         32767i16
51      #define _UI16_MAX        0xffffui16
52
53      #define  I32_MIN       (-2147483647i32 - 1)
```

3. Чтобы просмотреть содержимое библиотек – выделить и правый клик – всплывает подсказка – перейти к документу (определению) (F12)

```
1   //
2   // limits.h
3   //
4   //      Copyright (c) Microsoft Corporation. All rights reserved.
5   //
6   // The C Standard Library <limits.h> header.
7   //
8   #pragma once
9   #define _INC_LIMITS
10
11  #include <vcruntime.h>
12
13  #pragma warning(push)
14  #pragma warning(disable: _VCRUNTIME_DISABLED_WARNINGS)
15
16  _CRT_BEGIN_C_HEADER
17
18  #define CHAR_BIT      8
19  #define SCHAR_MIN   (-128)
20  #define SCHAR_MAX     127
21  #define UCHAR_MAX     0xff
22
23  #ifndef _CHAR_UNSIGNED
24      #define CHAR_MIN    SCHAR_MIN
25      #define CHAR_MAX    SCHAR_MAX
26  #else
27      #define CHAR_MIN    0
28      #define CHAR_MAX    UCHAR_MAX
29  #endif
30
31  #define MB_LEN_MAX    5
32  #define SHRT_MIN    (-32768)
33  #define SHRT_MAX      32767
34  #define USHRT_MAX     0xffff
35  #define INT_MIN     (-2147483647 - 1)
```

Каков результат работы:

```
int maxInt = 0x7fffffff;
int oct = 012; // octal value
int hex = 0x80;// hexadecimal value
printf("%d,  %d,   %d\n\n", maxInt,  oct, hex);
```

А в какой системе счисления выведен результат? Не понятно!

```
2147483647,  10,   128
```

Знак # - вывод лидирующих символов

```
printf("\n%d, %#o, %#x\n\n",INT_MAX,INT_MAX,INT_MAX);
```

```
2147483647, 017777777777, 0x7fffffff
```

Обратите внимание – тип знаковый, поэтому в старшем разряде не f, а 7 !!!

2. Вещественные константы

С фиксированной точкой:

целая часть – точка – дробная часть **1.5 3.14**

С плавающей точкой или в экспоненциальном виде:

целая часть – символ e – порядок **1.23e+5, 1.23e5, 1.23e-3**

```
float x, y, z;
double a, b, c;
```

Пример (с фиксированной точкой)

```
float f1 = 0.0075;
float f2 = 75e-4;
float f3 = .75;
float f4 = 75.;
printf("%f,  %f,  %f, %f\n\n", f1, f2, f3, f4);
```

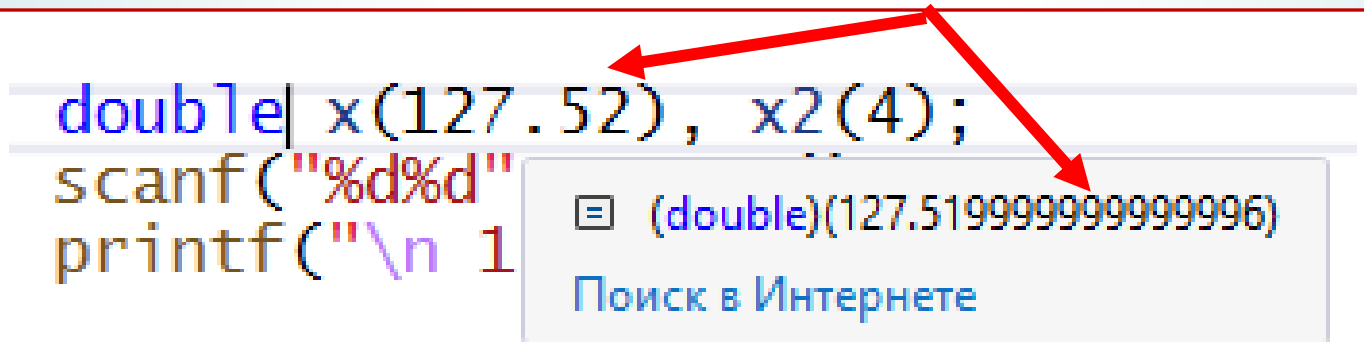
```
0.007500,  0.007500,  0.750000, 75.000000
```

Пример (%e - в экспоненциальном виде, с плавающей точкой)

```
float f1 = 0.0075;
float f2 = 75e-4;
float f3 = .75;
float f4 = 75.;
printf("%e, %e, %e, %e\n\n", f1, f2, f3, f4);
```

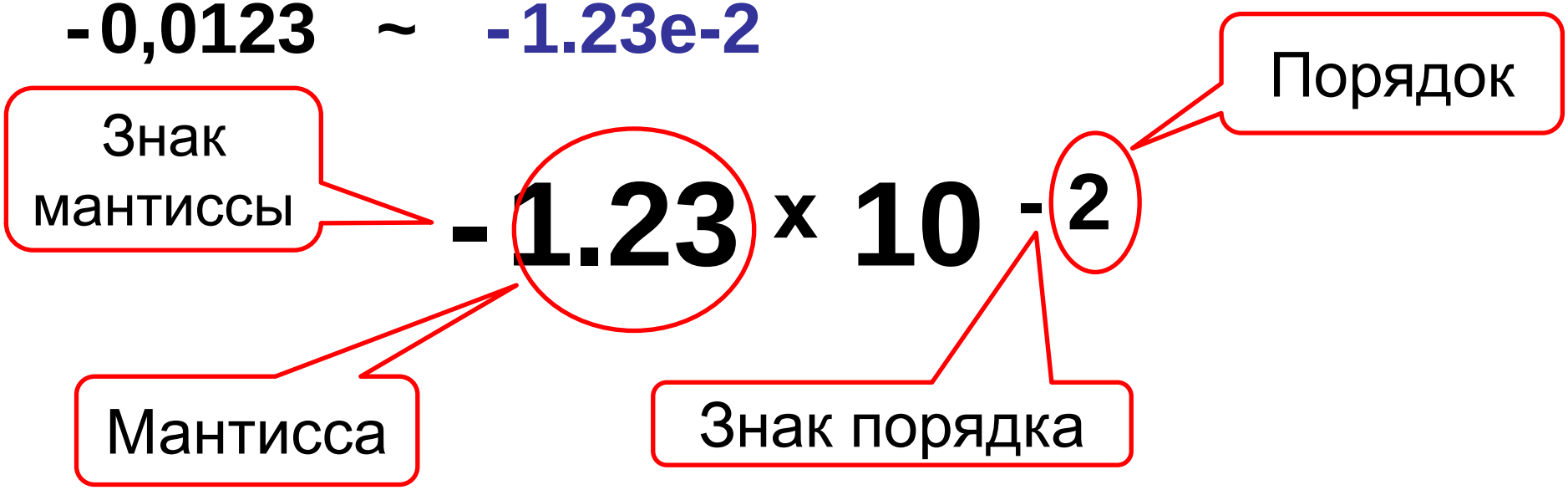
```
7.500000e-03, 7.500000e-03, 7.500000e-01, 75.000000
```

Вещественные числа хранятся в приближенном виде, например:



Как хранятся в памяти вещественные числа?

-0,0123 ~ -1.23e-2



1. Числа с одинарной точностью `float` – 24 разряда мантисса (1 разряд под знак), 8 разрядов порядок (1 разряд под знак), всего 32 разряда, точность 7 десятичных разрядов, максимум около $3.4*10^{38}$.
2. Числа с двойной точностью (double) – 53 разряда мантисса, 11 разрядов порядок, всего 64 разряда, точность 15 десятичных разрядов, максимум около $1.7*10^{308}$

Вывод на экран побитовых представлений значений типа `int` и типа `float`

```
for (int i = 0; i < 32; i++)
        printf("%3d", i); printf("\n");

{// побитовый вывод значения unt
unsigned int n = 1;    //меняем значение n, посмотреть результат
printf("\nCod n=%d\n ", n);
for (unsigned int i = 0x80000000; i; i>>=1)
   {   if (n & i)
            printf("%c  ", '1');
       else printf("%c  ", '0');
   }
}

{// побитовый вывод значения float
union {
    float f;
    unsigned char c[sizeof(float)*8];
} cod;
cod.f = 1.0;    //меняем значение f, посмотреть результат
printf("\nCod f=%f\n ", cod.f);
int i = sizeof(float) * 8-1;
{
    for (unsigned int j = 0x80000000; j; j >>= 1)
        {
            if (cod.c[i] & j)
                printf("%c  ", '1');
            else printf("%c  ", '0');
            i-- ;
        }
}
}
```

```
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

od n=1
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
od f=1.000000
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  0  0  0  0
:\Users\Admin\source\repos\Project8\x64\Debug\Project8.exe (процесс 5052) завершил работу с кодом
```

3. Символы константы

1. Символы в апострофах '1', 'A', '+'

2. Использование кода ASCII:

'\061', '\101', '\053' (восьмеричная нотация)

'\x31', '\x41', '\x2b' (шестнадцатеричная нотация)

Символьный тип char – конечный, упорядоченный целый тип

- беззнаковый 0 . . 255
- знаковый -128 . . 127

ASCII код	32	40-41	48-57	65-90	99 - 122	128 - 159	180
Сим-волы	про-бел	( )	0 - 9	A - Z	a - z	А - Я	┤

ASCII - American Standard Code for Information Interchange

Д/З

Таблица ASCII кодов (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)

Вывод символов

```
printf("\n%c %c %c %c %s\n", 'A', '+', 'B', '=', "AB");
```

```
A + B = AB
```

```
char ch1 = 'A', ch2 = 'B';
printf("\n A + B = %d \n", ch1+ch2) ;
```

```
A + B = 131
```

```
char ch1 = 'A', ch2 = 'B';
printf("\n A + B = %c \n", ch1+ch2) ;
```

```
A + B = Г
```

Получаем число, представляющее ASCII код русского символа Г

Вывод символов

```
printf("\n %d %d %d\n", 'A', '+', 'B');
```

```
65 43 66
```

```
printf("\n %c %c %c\n", '\101', 43, '\102');
```

```
A + B
```

```
printf("%o %c %o\n", '\101', '\053', '\102');
```

```
101 + 102
```

```
printf("%o %c %o\n", 'A', '\053', 'B');
```

4 «перевертыша»

```
101 + 102
```

Вроде бы работает! Но! Тип char занимает 1 байт, %d вводит и выводит 4 байта в 32 битном компиляторе!

1
```
printf("\n %d %d %d\n", 'A', '+', 'B');
printf("%o %c %o\n", '\101', '\053', '\102');
```

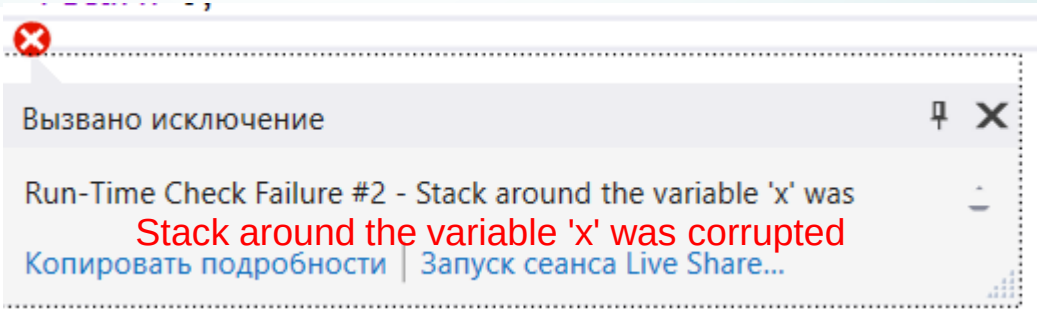
```
 65 43 66
101 + 102
```

2
```
printf("\n %d %d %d\n", (int)'A', (int)'+', (int)'B');
printf("%o %c %o\n", (int)'\101', (int)'\053', (int)'\102');
```

1 и 2 – данные не «искажаются», автоматически добавленные 3 нулевых байта слева при выводе не влияют на результат!
В случае 2 – явно результат приводим к типу `int` , т.е. к 4 байтам
3 – не верно! Так как под переменную x выделен 1 байт, а мы значение вводим в формате 4 байта! **4 - верно, приводим к формату %c - char**

3
```
char x;
scanf("%d", &x);
```

4
```
char x;
scanf("%c", &x);
```



В низкоуровневом программировании и форматном вводе/выводе аккуратно работаем с типом и форматированием. **Компилятор интерпретирует байты в соответствии формату ввода/вывода!**

Многосимвольные литералы (экзотика)

Особенность – в языке C/C++ литералы, имеющие более одного символа (до 4-х), имеют тип `int`.

Литералы заключаются в апострофы, а не кавычки!!!

Таким образом, 'ab' имеет тип `int`

'a' имеет тип `char`

```
int i1 = 'ABA';  //i1 содержит 4 символа 0|'A'|'B'|'A'
int i2 = i1 + 100;
printf("\n %d %d\n", i1,i2);
```

```
4276801 4276901
```

! ***Значение 'ABA' не путать со строковым значением "ABA"***

```
int i1 = 'ABA';                    //
int i2 = i1
int w = i1;
printf("\n %                       2);
```

Если подвести курсор, то будет подсказка по значению i1 – (int)**4276801**

Многосимвольные литералы

Максимальное количество символов в многосимвольной константе равно 4. Это размер типа `int`.

```
int i1 = 'ABCD';  // i1 содержит 'A'|'B'|'C'|'D'
int i2 = i1 + 100;
printf("\n %d %d\n", i1,i2);
```

```
1094861636 1094861736
```

Подведем курсор под переменную i1 в функции printf

```
int i1 = 'ABCD';  // i1 содержит 'A'|'B'|'C'|'D'
int i2 = i1 + 100;
int w = i1;
printf("\n %d %d\n", i1,i2);
```

(локальная переменная) int i1

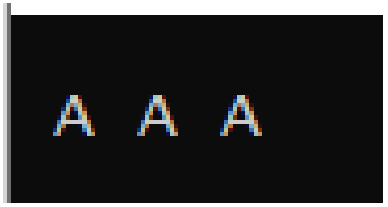
i1 содержит 'A'|'B'|'C'|'D'

Поиск в Интернете

Объявления и инициализация символов
(используем разные представления значений символов)

Инициализировать значения типа `char` можно несколькими способами:

```
char ch1 = 'A';     // значение символа
char ch2 = '\101'; // 8-ричный код символа
char ch3 = 0x41;    // 16-ричное код-целое
printf("\n %c %c %c \n", ch1, ch2, ch3);
```



Специальные символы или Escape-последовательности

	символ	8-рич. код	10-рич. код
Нулевой символ	'\0'	0	0
Звонок	'\a'	7	7
Возврат на шаг	'\b'	10	8
Горизонтальная табуляция	'\t'	11	9
Перевод строки	'\n'	12	10
Вертикальная табуляция	'\v'	13	11
Перевод формата	'\f'	14	12
Возврат каретки	'\r'	15	13
Кавычки	'\"'	42	34
Апостроф	'\''	47	39
Обрат. косая черта	'\\'	134	92

4. Строковые константы – массивы символов

```
"Stroka"
"Number%5d, %5d\n"
```

Строковые значения заключаются в кавычки

```
char s1[10] = "String";
char s2[]   = "String";
```

Выделено 10 байт

7 байт

s1

S	t	r	i	n	g	\0	\0	\0	\0

s2

S	t	r	i	n	g	\0

В конце каждой строки компилятор помещает нулевые символы ‘\0’, первый из них служит признаком конца строки.

2-й способ строк – объявить их с типом `string` и подключить библиотеку `<string>`, содержащую функции обработки строк. Будет в следующих лекциях.

Примеры вывода значений строковых переменных и размера памяти для их размещения

Инициализируем строки 3-мя способами

```
char s1[10] = "String";
char s2[]   = "String";
char s3[10] = { 's','t','r','i','n','g','\0' };
printf("\ns1=%s, sizeof=%d\n", s1, sizeof(s1));
printf("\ns2=%s, sizeof=%d\n", s2, sizeof(s2));
printf("\ns3=%s, sizeof=%d\n", s3, sizeof(s3));
```

```
s1=String, sizeof=10
s2=String, sizeof=7
s3=string, sizeof=10
```

В книгах и интернет ресурсах для версий Visual Studio C/C++ 2011 и ниже встречается следующая инструкция:

```
char* str = "string";
```
работало в старых версиях!!!

Здесь `str` – имя переменной, а `char*`– тип указателя на `char`, и при таком объявлении типа значение строки `str` можно менять. Но `"string"`– константное значение строки, которую компилятор размещает где-то в памяти, и ее значение константно! Поэтому ее нельзя объявлять с типом `char*`.

При таком присваивании получим сообщение об ошибке:
Ошибка C2440 инициализация: невозможно преобразовать "const char [8]" в "char *"

Значение типа `const char*` нельзя присвоить сущности типа `char*`

Однако, если не намерены изменять значение строки, правильное объявление: `const char* str = "string";`

5. Логический тип bool

Переменные типа `bool` могут содержать всего два значения – `true` (истина) и `false` (ложь).

`true` и `false` являются логическими литералами. В C/C++ `true` интерпретируется, как 1, а `false` как 0, т.е. логический тип относится к целым типам.

```
bool b1 = true, b2 = false;
printf("\n%s\n", (b1) ? ("TRUE") : ("FALSE"));
printf("\n%s\n", (b2) ? ("TRUE") : ("FALSE"));
```

```
TRUE

FALSE
```

В C++ :

```
#include <iostream>
.  .  .
bool x = true;
std::cout << std::boolalpha << x;
```

BCE