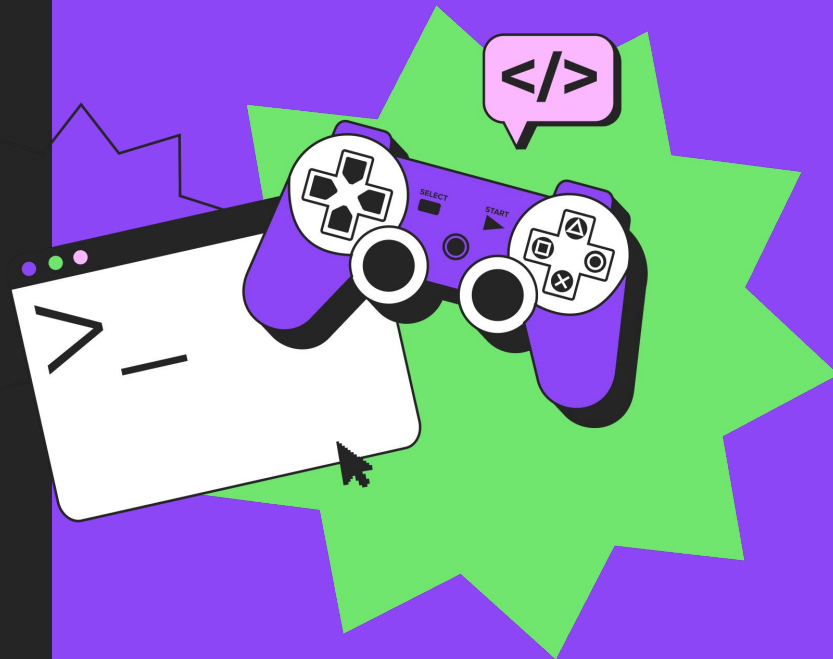


Типы данных. Операторы и выражения

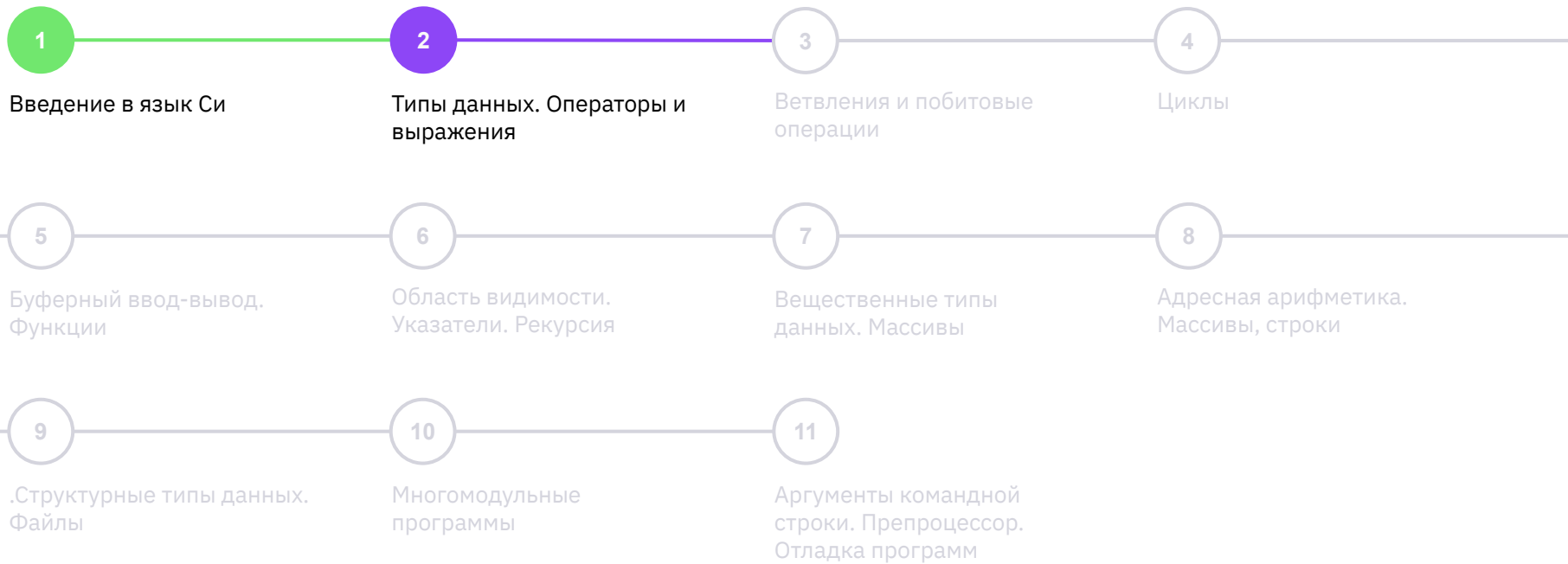
Урок 2

Программирование на языке Си (базовый уровень)













План курса





Содержание урока

На этой лекции вы узнаете про:

-  Какие типы данных бывают в языке Си
-  Спецификаторы типов (signed, unsigned, short и long)
-  Про объявление переменных и их имена
-  Спецификаторы классов памяти и квалификатор const
-  Что такое константы и литералы
-  Вычисление выражений
-  Укороченное присваивание и унарные операции
-  Форматный ввод и вывод



Типы данных



Типы данных

- `_Bool` — целый тип для хранения 0 и 1
- `char` — целый тип для хранения кода символа
- `int` — целый тип
- `float` — с плавающей точкой
- `double` — двойной точности
- `void` — без значения
- `_Complex` — модификатор комплексного типа





Опциональные спецификаторы типов:
знаковости signed, unsigned и
размера short, long



Опциональные спецификаторы типов (**signed, unsigned**)

Символьный тип **char**

- **char** Самый маленький, 8 бит, обычно *signed*
- **signed char** Гарантированно будет со знаком, $[-128, +127]$
- **unsigned char** Гарантированно без знака, $[0, 255]$



Опциональные спецификаторы **signed**, **unsigned** **long long**

Целочисленный тип **int**

- **short**, **short int**, **signed short**, **signed short int** Тип *короткого* целого числа со знаком, 16 бит, $[-32768, +32767]$
- **unsigned short**, **unsigned short int** такой же, как **short**, но беззнаковый, $[0, +65535]$
- **int**, **signed**, **signed int** Основной тип целого числа со знаком, как минимум из диапазона $[-32767, +32767]$ Как правило, в современных компиляторах 32 бит, $[-2\,147\,483\,647, +2\,147\,483\,647]$
- **unsigned**, **unsigned int** Такой же, как у **int**, но беззнаковый. Диапазон: $[0, +65\,535]$
- **long**, **long int**, **signed long**, **signed long int** Тип *длинного* целого числа со знаком, по крайней мере 32 бита, $[-2\,147\,483\,647, +2\,147\,483\,647]$
- **unsigned long**, **unsigned long int** Такой же, как у **long**, но беззнаковый. Диапазон: $[0, +4\,294\,967\,295]$
- **long long**, **long long int**, **signed long long**, **signed long long int** Тип *длинного длинного* (двойного длинного) целого числа со знаком. Может содержать числа как минимум в диапазоне $[-9\,223\,372\,036\,854\,775\,807, +9\,223\,372\,036\,854\,775\,807]$. Таким образом, это по крайней мере 64 бита. Утверждён в стандарте C99.
- **unsigned long long**, **unsigned long long int**. Похож на **long long**, но беззнаковый. Диапазон: $[0, 18\,446\,744\,073\,709\,551\,615]$



Опциональные спецификаторы типов

Вещественные типы **float** и **double**

- **float** Тип вещественного числа с плавающей точкой **32** бита
IEEE 754 бинарный формат с плавающей запятой одинарной точности
- **double** Тип вещественного числа с плавающей запятой двойной точности **64** бита
IEEE 754 бинарный формат с плавающей запятой двойной точности
- **long double** четырехкратной точности **80**-бит
IEEE 754 бинарный формат с плавающей запятой четырехкратной точности



Переменные



Переменная – это ячейка в памяти компьютера, которая имеет имя, адрес в памяти компьютера и хранит некоторое значение.

- Значение переменной может меняться во время выполнения программы
- При записи в ячейку нового значения старое стирается



Имена переменных

Могут включать

- латинские буквы (A-Z, a-z)
- знак подчеркивания _
- цифры 0-9

НЕ могут включать

- русские буквы
- пробелы
- скобки, знаки +, =, !, ? и прочее
- Имя переменной не может начинаться с цифры



Какие имена переменных верные?

- **ABCdf**
- **h&m**
- **4you**
- **Иван**
- **“GeekBrains”**
- **super173**
- **[goodname]**
- **_my_string**
- **a*b**



Ответ

- **ABCdf** — верное
- **h&m** — недопустимый знак &
- **4you** — начинается с цифры
- **Иван** — русские буквы
- **“GeekBrains”** — присутствуют кавычки
- **super173** — верное
- **[goodname]** — присутствуют скобки
- **_my_string** — верное
- **a*b** — недопустимый знак





Объявление переменных

```
int value; //объявление целочисленной глобальной переменной
double big_pi; //объявление вещественной глобальной переменной двойной точности

int main()
{
    int a; //объявление целочисленной локальной переменной
    float f; //объявление вещественной локальной переменной
    int su7, prime = 7, five = 5; //объявление переменных с инициализацией
    float pi = 3.14; //объявление переменных с инициализацией
    char c, c2 = 'A', m = 10; //объявление символьных переменных с инициализацией
}
```



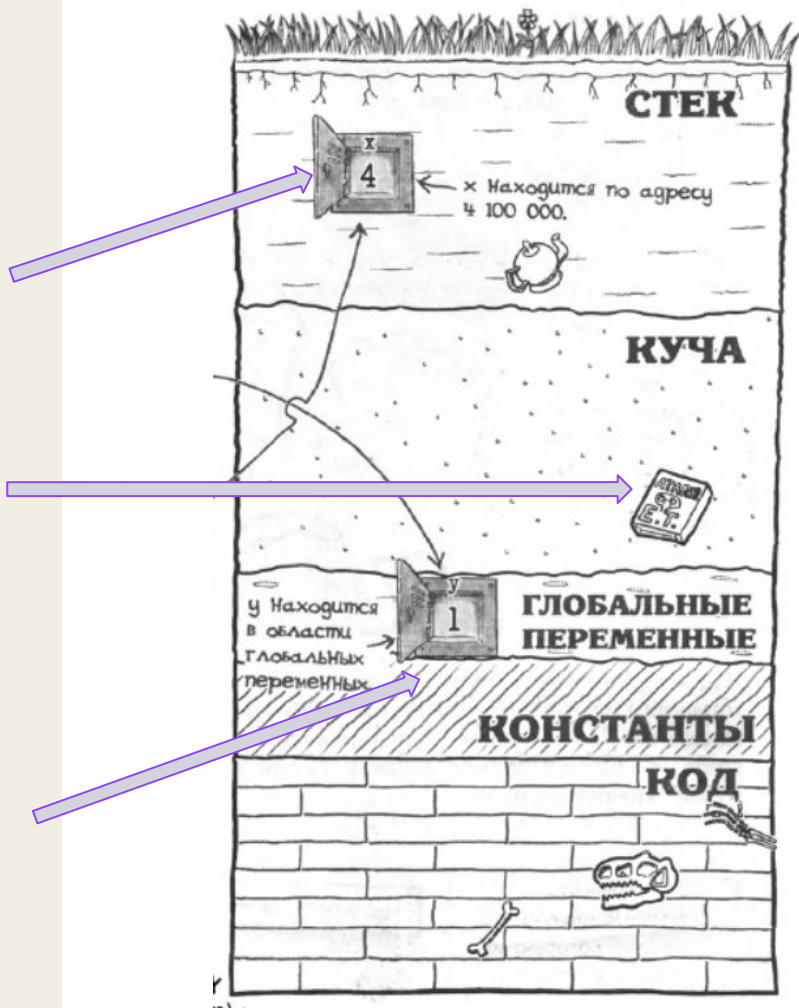
Внимание! Если начальное значение переменной не задано, то в переменной лежит “мусор”. Это **наиболее** распространенная причина нестабильности работы программы



Спецификаторы классов памяти

Выделение памяти

- **Автоматическое** — для локальных переменных внутри функций компьютер автоматически выделяет память в стеке.
- **Динамическое** — память выделяется в ходе выполнения программы динамически.
- **Статическое** — переменные попадают в отдельную область памяти, которая выделяется еще на этапе компиляции.





Выделение памяти

- **Автоматическое** — для локальных переменных внутри функций компьютер автоматически выделяет память в стеке.
- **Динамическое** — память выделяется в ходе выполнения программы динамически.
- **Статическое** — переменные попадают в отдельную область памяти, которая выделяется еще на этапе компиляции.

Также есть память, которая хранит код программы





Спецификаторы классов памяти и квалификатор **const**

Спецификаторы:

- **auto** – переменная будет создана и доступна только внутри блока
- **static** – место под переменную будет выделено в статической памяти
- **extern** – место под глобальную переменную выделяется при ее объявлении в другом файле

Квалификатор:

- **const** – значение переменной не будет изменяться после инициализации.



Объявление переменных

```
extern int Voltage;    //место под глобальную переменную выделяется
                        при ее объявлении в другом файле
int main()
{
    const int SIZE = 1000; //Объявление целочисленной константы.
    static int Current;    //место под переменную будет выделено в
    статической памяти
}
```



Обратите внимание, имена констант обычно пишут ЗАГЛАВНЫМИ БУКВАМИ



Константы и литералы



Константы и литералы

Константы — фиксированные величины, которые не изменяются во время выполнения программы. Значение константы называется **литералом**.

- целочисленная константа (имеют *префикс* и *суффикс*)
- константа с плавающей точкой (*суффикс*)
- символьная константа
- строковый литерал



Целочисленные константы (префикс)

Целочисленные константы записываются в виде чисел в десятичной, восьмеричной или шестнадцатеричной системе; **префикс** определяет основание (радикс) системы счисления:

- десятичные числа – без префикса (100)
- восьмеричные – с префиксом 0 (077 = 63)
- шестнадцатеричные – с префиксом 0x или 0X (0x1F = 31)
- двоичные - с префиксом 0b00010000 были введены в стандарте C18 и GCC



Целочисленные константы (суффикс)

Тип целочисленной константы определяется буквенным **суффиксом**, приписываемым к цифрам числа. Он представляет собой комбинацию букв U и L, означающих целое беззнакового и длинного типа (соответственно).

- суффикс L соответствует типу long (34L)
- суффикс LL – long long (123LL)
- буква U (или u) – типу unsigned (1000u)



Литералы с плавающей точкой

Литералы с **плавающей** точкой состоят из целой части, десятичной точки, дробной части и экспоненты. Их можно представлять форме:

- Десятичной, например, 100.50 или 0.000127.
- Экспоненциальной, например, -0.77E-5. Здесь “E-5” означает 10^{-5} .



Всем литералам с плавающей точкой по умолчанию присваивается тип `double`. Чтобы создать литерал типа `float`, нужно после литерала указать суффикс `'f'` или `'F'`.

Какие из литералов верные?

- **212**
- **215u**
- **0xFeeL**
- **078**
- **032UU**
- **3.14159**
- **314159E-5L**
- **510E**
- **210f**
- **.e55**



Ответ

- **212** — верно
- **215u** — верно
- **0xFeeL** — верно
- **078** — неверно: 8 - не восьмеричное число
- **032UU** — неверно: нельзя повторять суффикс
- **3.14159** — верно
- **314159E-5L** — верно
- **510E** — неверно: неполная экспонента
- **210f** — неверно: отсутствует десятичная точка или экспонента
- **.e55** — неверно: отсутствует целая часть или дробь





Символьные константы и строковые литералы

Символьные константы записываются в одинарных кавычках, например, 'a', '5'.

Строковые литералы заключаются в двойные кавычки “”



Можно разбить длинную строку на несколько строк разделяя их пробельными символами.

```
"hello, dear"
```

```
"hello, \
```

```
dear"
```

```
"hello, " "d" "ear"
```



Пример программы с константами и литералами

```
#include <stdio.h>

int main ()
{
    const float LENGTH = 10.f; //Литерал с плавающей точкой
    const int    WIDTH  = 0xFFU; //Ширина задана в HEX-формате без знака = 255
    const char   NEWLINE = '\n';
    const char   STR[] = "value of area : " \
                        "%f";

    float area;

    area = LENGTH * WIDTH;
    printf (STR, area);
    printf ("%c", NEWLINE);

    return 0;
}
```



Арифметические операции



Арифметические операции над целочисленными значениями:

- сложение "+"
- вычитание "-"
- умножение "*"
- деление нацело "/"
- остаток от деления нацело "%"



При делении нацело результат всегда округляется в сторону нуля и выполняется равенство $(a / b) * b + a \% b = a$



Знак остатка совпадает со знаком делимого



Задание:

$$-27/5 = ? \quad \text{и} \quad -27\%5 = ?,$$

$$-27/-5 = ? \quad \text{и} \quad -27\%-5 = ?,$$

$$27/-5 = ? \quad \text{и} \quad 27\%-5 = ?$$





Ответ:

$$-27/5 = -5, \quad -27\% \ 5 = -2, \quad -27 = (-5)*5 + (-2)$$

$$-27/-5 = 5, \quad -27\%-5 = -2, \quad -27 = 5*(-5) + (-2)$$

$$27/-5 = -5, \quad 27\%-5 = 2, \quad 27 = (-5)*(-5) + 2$$

Задание

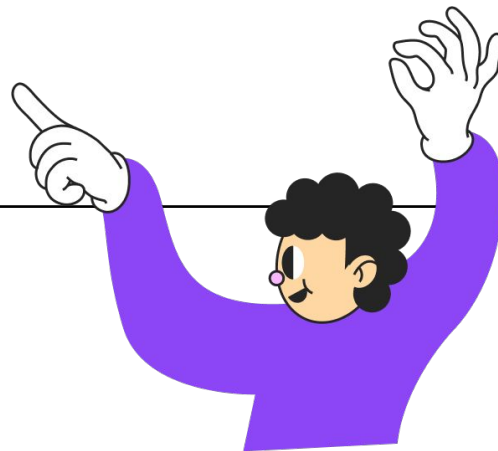
```
int main()
{
    int a, x, y, z;
    a = 27;           //положить целое число 27 в переменную a
    x = a / 5;        // x = ?
    y = 11 % 3;       // y = ?
    z = (x + 5) * y;  // z = ?
}
```



**Поставьте видео
на паузу и
решите задачу**

Ответ

```
int main()
{
    int a, x, y, z;
    a = 27; //положить целое число 27 в переменную a
    x = a / 5; // x = 5
    y = 11 % 3; // y = 2
    z = (x + 5) * y; // z = 20
}
```





Укороченное присваивание

вместо записи: $v = v \# e$; можно использовать запись $v \# = e$;

где $\#$ — любая арифметическая операция, v — переменная

Такие присваивания поддерживаются для всех двухместных операций:

“+”, “-”, “/”, “%”.

```
int a = 50, b = 7;  
a = a + b; // a = 57
```

```
int a = 50, b = 7;  
a += b; // a = 57
```



Унарные операции

- **Инкремент** “++” - увеличение операнда на единицу
- **Декремент** “--” - уменьшение операнда на единицу



Унарные операции

- **Инкремент** “++” - увеличение операнда на единицу
- **Декремент** “--” - уменьшение операнда на единицу

// Постфиксная форма

```
int a, b = 7;
```

```
    a = b++; // a = 7 b = 8
```

// Префиксная форма

```
int a, b = 7;
```

```
    a = ++b; // a = 8 b = 8
```

Эквивалентный код для *постинкремента* (суффиксный) и *преинкремента* (префиксный)

// Постфиксная форма

```
int a, b = 7;
```

```
    a = b;
```

```
    b = b + 1 // a = 7 b = 8
```

// Префиксная форма

```
int a, b = 7;
```

```
    b = b + 1
```

```
    a = b; // a = 8 b = 8
```



Унарные операции

- **Инкремент** “++” - увеличение операнда на единицу
- **Декремент** “--” - уменьшение операнда на единицу

// Постфиксная форма

```
int a, b = 7;  
a = b++; // a = 7 b = 8
```

// Префиксная форма

```
int a, b = 7;  
a = ++b; // a = 8 b = 8
```

Эквивалентный код для *постинкремента* (суффиксный) и *преинкремента* (префиксный)

// Постфиксная форма

```
int a, b = 7;  
a = b;  
b = b + 1 // a = 7 b = 8
```

// Префиксная форма

```
int a, b = 7;  
b = b + 1  
a = b; // a = 8 b = 8
```



`arr[++arr[0]] = value;` индекс в 0-м элементе сначала увеличивается, потом данные `value` будут внесены в массив



`arr[arr[0]++] = value;` сначала данные `value` будут внесены в массив, затем индекс в 0-м элементе увеличивается



Ввод-вывод



Ввод-вывод

Пример. Ввести два целых числа и вывести на экран их сумму

```
#include <stdio.h> //Объявить библиотеки ввода-вывода
main()
{
    int a, b, c; //Объявить переменные
    printf("Input number:\n"); //Вывести на экран подсказку
    scanf ("%d%d", &a, &b);      //Считать два целых числа и записать их по
адресу a, b
    c = a + b;          //Сложить два числа и поместить сумму в c
    printf("%d", c); //Вывести на экран значение в переменной c
    return 0;          //Завершить программу успешно
}
```



Спецификатор типа ввода-вывода

- **%d** – целое десятичное число со знаком тип **int**.
- **%ld** – целое десятичное число со знаком тип **long int**
- **%Ld** – целое десятичное число со знаком тип **long long int**
- **%u** – целое десятичное без знака
- **%x** – целое число в шестнадцатеричном виде
- **%f** – вещественное число **float**
- **%lf** – вещественное число **double**
- **%Lf** – вещественное число **long double**
- **%c** – **1 символ**



Где ошибка?

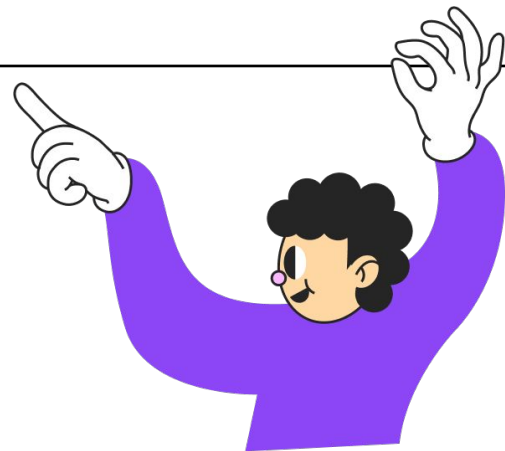
```
int a, b;  
scanf ("%d", a);  
scanf ("%d", &a, &b);  
scanf ("%d%d", &a);  
scanf ("%d %d", &a, &b);  
scanf ("%f%f", &a, &b);
```





Ответ

```
int a, b;  
scanf ("%d", a);      // &a не хватает знака & амперсанд  
scanf ("%d", &a, &b); // %d%d две переменные, а спецификатор один  
scanf ("%d%d", &a);   //&a, &b два спецификатора, одна переменная  
scanf ("%d %d", &a, &b); // пробел будет считаться разделителем, если  
второй аргумент введен не через пробел, он не считается  
scanf ("%f%f", &a, &b); // %d%d тип переменной (int) не соответствует  
типу спецификатора (float)
```





Структура спецификатора формата

%[флаги][ширина][.точность][размер]тип

Знак	Название знака	Значение	В отсутствии этого знака
-	минус	выводимое значение выравнивается по левому краю в пределах минимальной ширины поля	по правому
+	плюс	всегда указывается знак (плюс или минус) для выводимого десятичного числового значения	только для отрицательных чисел
	пробел	помещать перед результатом пробел, если правый символ значения не знак	вывод может начинаться с цифры
#	октоторп	“альтернативная форма” вывода значения	
0	ноль	дополнять поле до ширины, указанной в поле ширина управляющей последовательности, символом 0	дополнить пробелами



Структура спецификатора формата

```
#include <stdio.h>
int main()
{
    int x = 1234;
    printf ("%d\n", x); //минимальное число позиций под вывод числа
//1234
    printf ("%9d\n", x); //под вывод числа выделено 9 позиций
//    1234
    printf ("%09d\n", x); //под вывод числа выделено 9 позиций дополненный 0
//000001234
    printf ("%+09d\n", x); //под вывод числа со знаком +
//+00001234
    printf ("%#09x\n", x); //под вывод типа системы счисления
//0x00004d2
    printf ("%04d %04d %04d\n", x,x,x); //таблица
    printf ("%04d %04d %04d\n", x,x,x); //таблица
//1234 1234 1234
    return 0;
}
```



Структура спецификатора формата 2

```
#include <stdio.h>
int main()
{
    // float x = 123.4567;
    float x = 123.625;
    printf ("%f\n", x);
    //123.456700
    printf ("%9.3f\n", x);
    //123.456
    printf ("%e\n", x); // стандартно 1.234567 * 102
    //1.234560e+02
    printf ("%10.2e\n", x); // всего 10 знаков, 2 цифры под мантиссу
    //1.23e+02
    return 0;
}
```



Задача:

Целой переменной k присвоить значение, равное первой цифре дробной части в записи вещественного положительного числа x .



Ответ

Целой переменной *k* присвоить значение, равное первой цифре дробной части в записи вещественного положительного числа *x*. Можно добавить *scanf* и *printf*

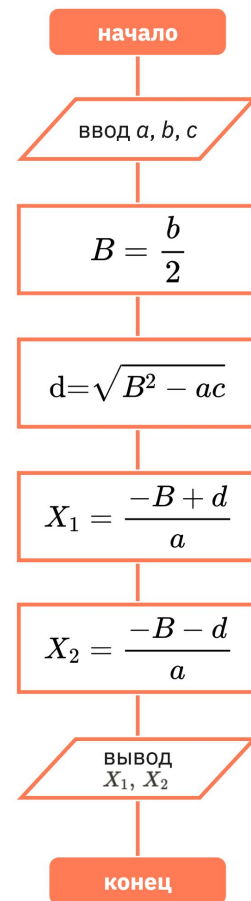
```
#include <stdio.h>
int main(void)
{
    float f=123.567;
    int k, fint;
    fint = f; //fint = 123
    fint *= 10; //fint = 1230
    f = f*10; //f = 1235.67
    k = f - fint; //k=5
    printf("%d",k);
}
```






```
#include <stdio.h>
#include <locale.h>
#include <math.h>

int main(int argc, char **argv)
{
    float a,b,c;
    float B,d;
    float X1,X2;
    setlocale(LC_ALL, "Rus");
    printf("Вычисление корней квадратного уравнения\n");
    printf("a*x*x+b*x+c=0\n");
    printf("Введите a:\n");
    scanf ("%f", &a); //1
    printf("Введите b:\n");
    scanf ("%f", &b); //18
    printf("Введите c:\n");
    scanf ("%f", &c); //32
    B = b/2;
    d = sqrtf(B*B - a*c);
    printf("Корни квадратного уравнения \n");
    X1 = (-B + d)/a; //2
    printf("X1 = %f \n",X1);
    X2 = (-B - d)/a; //16
    printf("X2 = %f \n",X2);
    return 0;
}
```



На этой лекции вы узнаете про:

- Какие типы данных бывают в языке Си
- Спецификаторы типов (signed, unsigned, short и long)
- Объявление переменных и их имена
- Спецификаторы классов памяти и квалификатор const
- Что такое константы и литералы
- Вычисление выражений
- Укороченное присваивание и унарные операции
- Форматный ввод и вывод

Спасибо 
за внимание

Занимайтесь любимым делом!

