GeekBrains

Типы данных Си

Программирование на языке Си (базовый уровень)

Оглавление

Введение

На предыдущей лекции мы изучили базовое устройство компьютера, принципы фон Неймана. Познакомились с компиляция программы и для чего нужен компилятор. Сделали первые программы.
На этой лекции вы узнаете:

- 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
- Про вычисление выражений
- Узнаем про явное и неявное приведение типов
- Укороченное присваивание и унарные операции
- Форматный ввод и вывод

Термины, используемые в лекции

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

Константа – это неизменяемая ячейка в памяти компьютера, которая имеет имя, адрес в памяти компьютера и хранит некоторое значение. Эти фиксированные величины также называются **литералами**.

Радикс — основание системы счисления.

Оператор – это команда языка программирования.

Оператор присваивания – это команда по изменению переменной.

Приведение типа — преобразование значения одного типа в значение другого типа.

Точка следования (sequence point) – это место в программе, в котором все побочные эффекты предыдущих вычислений закончены, а новые – не начаты.

Целочисленное расширение (integer promotion) – это неявное целочисленное преобразование.

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

💡 Типы данных *_Bool* и *_Complex* – введены в стандарте C99

Можно выделить четыре основных типа данных: *char*, *int*, *float*, *double*. Целочисленными типами являются типы *int* и *char*.

К этим типам применимы модификаторы знаковости **signed** (со знаком положительным или отрицательным) и **unsigned** (большие или равные 0). Если явно модификатор знаковости не указан, то целый тип *int* по умолчанию знаковый - *signed int*.

К типу *int* применимы также модификаторы размера **short** (не менее 16 бит), **long** (не менее 32 бит), **long long** (не менее 64 бит).

По возрастанию размера целочисленные типы можно отсортировать как *char*, *short*, *int*, *long*, *long long*. Размер типа *int* не может быть менее 2 байт, типа *long* – не менее 4 байт, типа *long long* – не менее 8 байт.

Для современной 64-битной архитектуры x86- 64 размер типа *int* составляет 4 байта, long и long long – 8 байт.

Тип	Пояснение
char	Целочисленный, самый маленький из возможных адресуемых типов (как правило, **8 бит**). Обычно *signed*.
signed char	Того же размера что и *char*, но гарантированно будет со знаком, обычно в реализациях [−128, +127]
unsigned char	Того же размера что и *char*, но гарантированно без знака. Как правило, [0, 255]
short short int signed short signed short int	Тип *короткого* целого числа со знаком. Обычно в реализациях [−32768, +32767]. **16 бит** (2 байта).
unsigned short unsigned short int	Такой же, как *short*, но беззнаковый. Диапазон: [0, +65535]
int signed signed int	Основной тип целого числа со знаком. Таким образом, это по крайней мере **16 бит** (2 байта). Как правило, **32 бита** (4 байта) и диапазон [−2 147 483 648, +2 147 483 647], однако на 16- и 8-битных платформах (Embedded и IoT) имеет размер, как правило, 2 байта в диапазоне значений [−32768, +32767]
unsigned unsigned int	Такой же как *int*, но беззнаковый. Диапазон: [0, +4 294 967 295] для **32 бит**.
long long int signed long signed long int	Тип *длинного* целого числа со знаком. Может содержать числа, как минимум, в диапазоне [−2 147 483 647, +2 147 483 647].Таким образом, это по крайней мере **32 бита** (4 байта).
unsigned long unsigned long int	Такой же как *long*, но беззнаковый. Диапазон: [0, +4 294 967 295] **32 бита**.
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 [−9 223 372 036 854 775 808, +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]. **64 бита**

float	Тип вещественного числа с плавающей точкой. **32 бита** Будет разобрана в последующих лекциях. IEEE 754 бинарный формат с плавающей запятой одинарной точности.
double	Тип вещественного числа с плавающей запятой. **64 бита** IEEE 754 бинарный формат с плавающей запятой двойной точности.
long double	В отличие от float и double, может быть **80-битным** «IEEE 754 бинарный формат с плавающей запятой четырехкратной точности».

Значения целочисленных типов хранятся в позиционной двоичной системе счисления. Знаковый и беззнаковый варианты одного и того же целого типа имеют один и тот же размер.

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.

🔥 **Внимание!** Имя переменной не может начинаться с цифры.

Вопрос. Какие имена 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] - присутствую скобки

- _my_string — верное
- a*b - недопустимый знак

Объявление переменных

Чтобы использовать переменную, необходимо задать ее *тип* и *имя*.

При этом в памяти выделяется под нее место определенного размера, соответствующего *типу*. Выделение памяти бывает: автоматическое, статическое и динамическое.

Автоматическое выделение памяти — задавая переменные внутри функций или же вызывая другие функции, компьютер автоматически выделяет в определенном участке памяти, организованной в виде стека, нужное количество байт и так же автоматически освобождает данный участок памяти, когда функции заканчивают свою работу.

Статическое выделение памяти — если вы используете ключевое слово static перед объявлением локальной переменной или любую глобальную переменную (вне какой-либо функции), то такие переменные попадают в отдельную область памяти — в статическую память, которая выделяется еще на этапе компиляции.

Динамическое выделение памяти — если вы не знаете заранее, какое вам нужно количество памяти, и при этом вы не хотите тратиться на "запас", то необходимое количество байт можно выделить (аллоцировать) в ходе выполнения программы динамически.

Опционально можно задать начальное значение переменной — **проинициализировать**.

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

int main()
{
    int a; //объявление целочисленной переменной
    float f; //объявление вещественной 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 и квалификатор const

При объявлении переменных часто используют спецификаторы классов памяти static, extern, auto, register и квалификатор const:

- **auto** - переменная будет создана и доступна только внутри блока. Все локальные переменные по умолчанию имеют такой класс. В Си не используется, но часто применяется в C++.
- **const** – значение переменной не будет изменяться после инициализации. Например, `const int SIZE = 123;`

🔥 При попытке изменить переменную произойдет ошибка на этапе компиляции.

💡 Обратите внимание, имена констант обычно пишут ЗАГЛАВНЫМИ БУКВАМИ

- **static** – место под переменную будет выделено в статической памяти, доступ к переменной возможен во время всего выполнения программы, а ее инициализация выполняется до начала работы программы. Подробно разберем в разделе с функциями и в разделе “Многомодульные программы”.
- **extern** – место под глобальную переменную выделяется при ее объявлении в другом файле, инициализация также выполняется в другом файле, доступ к переменной возможен во время всего выполнения программы. Подробно разберем в разделе “Многомодульные программы”.

- **register** - это намек на “реализацию”, что объявленная переменная будет сильно использоваться (устарело).

```
extern int  Voltage;//место под глобальную переменную выделяется при ее
объявлении в другом файле
int main()
{
    const int SIZE = 1000;//Объявление целочисленной константы.
    auto Power = 0.5;   //warning: type defaults to ‘int’
    static int  Current;//место под переменную будет выделено в статической
памяти
}
```

Константы и литералы

Константы — фиксированные величины, которые не изменяются во время выполнения программы. Эти фиксированные величины также называются **литералами**.

Константы могут относиться к любому из основных типов данных, например:

- целочисленная константа
- константа с плавающей точкой
- символьная константа
- строковый литерал

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

- 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

🔥 **Внимание!** Запись 09 неверна и вызовет ошибку компиляции, это число в восьмеричной системе счисления.

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 может быть прописной или строчной буквой и располагаться в любом порядке. Константы без суффикса имеют знаковый тип *int*.

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

Например, константы 1000u, 34L, 945LLU имеют типы *unsigned int*, *long*, *unsigned long long* соответственно.

💡 При этом если константа без суффикса слишком велика, чтобы быть представлена типом int, она автоматически получит тип long или long long.

Ниже приведены другие примеры различных типов целочисленных литералов:

```
85          /* десятичный */
0213        /* восьмеричный */
0x4b        /* шестнадцатеричный */
30          /* целочисленный */
30u         /* беззнаковый целочисленный */
30l         /* длинный */
30ul        /* беззнаковый длинный */
```

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

При представлении десятичной формы включают десятичную точку, экспоненту либо и то, и другое; а при представлении экспоненциальной формы — целую часть, дробную часть либо обе части. Знаковая экспонента обозначается символом “e” или “E”.

Задача. 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 записываются в одинарных кавычках, например, 'a', '5'.

Строковые литералы (константы) заключаются в двойные кавычки “”. Строка содержит символы, похожие на символьные литералы. Например "hello, dear".

💡 Можно разбить длинную строку на несколько строк, используя строковые литералы и разделяя их пробельными символами. Все три формы являются идентичными строками:

"hello, dear"
"hello, \
dear"
"hello, " "d" "ear"

```c
#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;
}
```

Вычисление выражений

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 = ? и -27%5 = ?,

-27/-5 = ? и -27%-5 = ?,

27/-5 = ? и 27%-5 = ?

Ответ: -27/5 = -5, -27% 5 = -2, -27 = (-5)*5 + (-2)

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

27/-5 = -5, 27%-5 = 2, 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 = ?
}
```

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

Такие присваивания поддерживаются для всех двухместных операций:

“+”, “-”, “/”, “%”.

```
int  a  = 50, b = 7;
     a  = a + b;  // a = 57
     a += b;      // a = 64 можно так
```

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
```

Данные операции также являются операциями с побочным эффектом.

Рассмотрим 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
```

Допустим, нам нужно сделать массив из целых чисел, где вначале будет лежать его длина, а начиная с первого элемента числа с 1 до 9. Эта задача изящно решается при помощи преинкремента.

```
int arr[10] = {0};           //объявление массива
for(int i = 1; i < 10; i++) //цикл по элементам с 1 по 9
    arr[++arr[0]] = i;       //добавление данных в массив
```

💡 `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

Функции ввода-вывода в Си являются частью стандартной библиотеки языка, для их использования нужно подключать заголовочный файл *stdio.h*.

Функции, оперирующие со стандартными потоками ввода (*stdin*) и вывода (*stdout*) обычно связаны с клавиатурой и дисплеем соответственно.

Функции форматного ввода-вывода *scanf* и *printf* могут считывать и записывать данные любых базовых типов и строки согласно заданной форматной строке, которая является первым аргументом функций.

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

```c
#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 с
    printf("%d", c);//Вывести на экран значение в переменной с
    return 0;       //Завершить программу успешно
}
```

В функции *scanf* перед именами переменных надо вводить амперсанд ***&.***

Форматная строка задает количество и тип значений, которые необходимо считать либо записать, а остальные аргументы являются указателями на объекты памяти, куда нужно поместить считанные значения, либо выражениями, вычисляющие значения для записи. Формат одного значения задает **спецификатор** ввода-вывода, начинающийся с символа *%.*

Перед чтением числа пропускаются все пробельные символы (пробелы, табуляции, переводы строк).

Символьная строка показывает какие числа вводятся (выводятся):

%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 (%lld)*
%u – целое десятичное без знака
%lu – целое десятичное число со знаком тип *unsigned long int*
%Lu – целое десятичное число со знаком тип *unsigned long long int (%llu)*
%x – целое число в шестнадцатеричном виде
%f – вещественное число *float*
%lf – вещественное число *double*
%Lf – вещественное число *long double*
%c – 1 символ

- Если первый не пробельный символ не может начинать число, функция *scanf* завершается. Иначе число считывается либо пока не возникнет переполнения, либо пока не встретится символ, который не может быть частью числа.
- Если возникло переполнение, функция scanf завершается с неудачей.
- Если встретился не цифровой символ (кроме разделителя), чтение числа считается успешным, а этот символ не будет считан из потока.
- Если встретился разделитель, то считывается следующее число.

💡 *scanf ("%d;%d", &a, &b);* Красным цветом выделен символ-разделитель. Данные при вводе с консоли должны быть указаны через разделитель. Например: *1;2*

💡 Разделителем может быть любой печатный символ

Задача. Считать целое число и напечатать его в шестнадцатеричном виде.

```
#include <stdio.h>
main()
{
    int a;
    printf("Input number:");
    scanf ("%d", &a); // целое десятичное число со знаком тип int
    printf("%x", a);  // целое число в шестнадцатеричном виде
    return 0;
}
```

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);
```


```
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)
```

Вывод чисел

```
int a = 1, b = 5, c = 123;
printf ("%d", c);
//123
printf ("Result: %d", c);
//Result: 123
printf ("%d+%d=%d", a, b, c );
//1 + 5=123
printf ("%d+%d=%d", a, b, a + b );
//1 + 5=6
```

Структура спецификатора формата

Спецификатор формата имеет вид: **%[флаги][ширина][.точность][размер]тип**

Обязательными составными частями являются символ начала спецификатора формата (%) и тип.

Флаги:

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

```
#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;
}
```

💡 С помощью форматированного вывода удобно делать таблицы

Вывод вещественных чисел

```
float x = 123.4567;
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
```

Задача

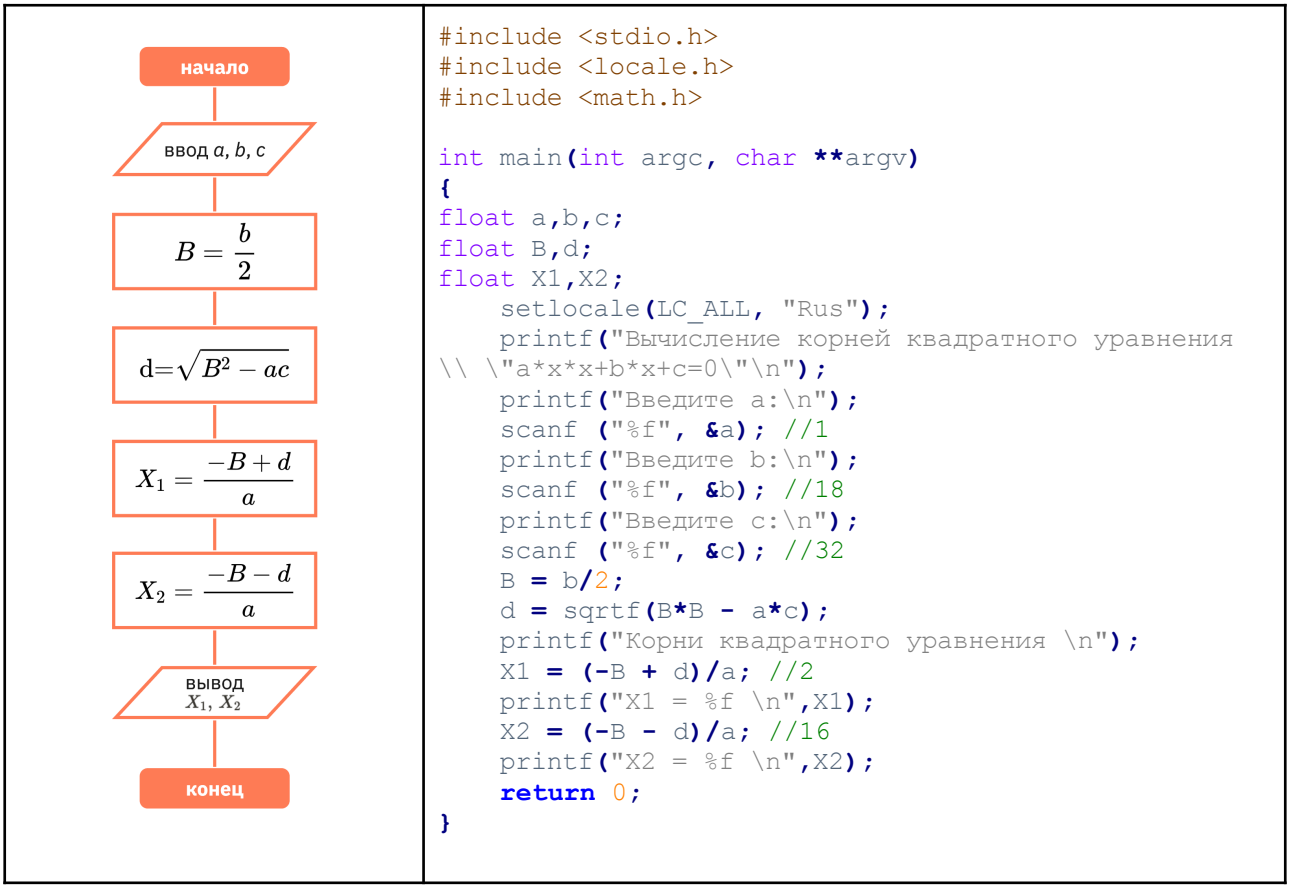
Целой переменной k присвоить значение, равное первой цифре дробной части в записи вещественного положительного числа x.

```
#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
}
```

Вычисление корней квадратного уравнения (продолжение)

По итогам сегодняшней лекции мы уже можем реализовать простой алгоритм вычисления корней квадратного уравнения с вводом и выводом параметров.

Чтобы вычислить квадратные корни мы подключаем стандартную библиотеку math.h и используем функции вычисления квадратного корня *sqrtf()*.




```c
#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
\\ \"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;
}
```

Подведение итогов

Итак, на этой лекции мы рассмотрели типы данных Си, правила объявления и именования переменных, какие бывают спсецификаторы классов памяти и квалификатор const и константы и литералы.

Изучили запись и вычисление выражений.

Разобрали укороченное присваивание и унарные операции.

Познакомились с системой ввода-вывода в языке Си. Как происходит ввод-вывод чисел с помощью спецификатора формата и особенности вывода вещественных чисел.

Написали продолжение программы вычисления корней квадратного уравнения. Теперь программа позволяет вводить и выводить данные, но не обрабатывает исключительные ситуации, такие как деление на 0, отрицательный дискриминант и т.д.

Домашнее задание

Отработать примеры кода, показанные в лекции.

Используемая литература

1. cprogramming.com - учебники по C и C++
2. free-programming-books - ресурс содержащий множество книг и статей по программированию, в том числе по C и C++ (там же можно найти ссылку на распространяемую бесплатно автором html-версию книги Eckel B. «Thinking in C++»)
3. tutorialspoint.com - ещё один ресурс с множеством руководств по изучению различных языков и технологий, в том числе содержит учебники по C
4. Столяров А.В. «Введение в язык Си++» - учебное пособие. Предполагается, что читатель знаком с языком Си
5. Юричев Д. «Заметки о языке программирования Си/Си++» - «для тех, кто хочет освежить свои знания по Си/Си++»
6. Онлайн версия «Си для встраиваемых систем»