ШАГ компьютерная АКАДЕМИЯ

Алгоритмы и структуры данных

Урок 4. Функции ввода/вывода. Системы счисления. Битовые операции

Руденков Александр Сергеевич

План занятия

1. Ввод и вывод
2. Системы счисления
3. Битовые операции

Ввод/вывод

В классическом стандарте ANSI C для ввода и вывода использовались функции библиотеки ***<stdio.h>***.

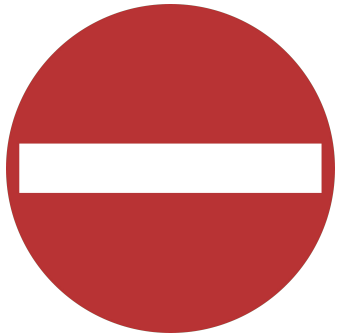
Для вывода используется функция ***printf*** следующего вида:

```
printf("управляющая строка",список аргументов);
```

Управляющая строка обязательно должна присутствовать, а вот списка аргументов может и не быть. В списке аргументов может быть любое количество элементов.

Управляющая строка содержит объекты трех типов: ***обычные символы***, которые просто выводятся на экран консоли, ***Escape-последовательности (управляющие символы)***, и ***спецификации преобразования***.

Каждая спецификация преобразования начинается со знака **%** и заканчивается некоторым символом формата (см. таблицу). Между ними могут встречаться (или не встречаться) знаки, уточняющие формат.



Ввод/вывод

Спецификация преобразования	Описание
%d	десятичное целое число
%o	восьмеричное целое число
%x	шестнадцатеричное целое число
%u	десятичное целое число без знака
%f	вещественное десятичное число в формате с плавающей точкой (например, 82.312)
%e	вещественное десятичное число в экспоненциальном формате (например, 0.82312E2)
%g	вещественное десятичное число, выбирается наиболее короткий формат %f или %e
%c	символ
%s	строка символов
%p	указатель

Ввод/вывод

Значения аргументов, указанных далее в функции в списке аргументов, при выводе подставляются на место соответствующих спецификаций преобразования в порядке их следовании. Английское название спецификации преобразования ***«place holder»***, т.е. ***«держатель места»***, поскольку они как бы «бронируют» место для размещения очередного выводимого элемента в строке.

```
int i=25;
char c='$';
double g=1.72;


printf("\t i=%4d c=%c g=%f\n", i,c,g);
```

```
        i=  25 c=$ g=1.720000

--------------------------------
Process exited after 0.0523 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Ввод/вывод

Чаще всего для вывода ***целого числа*** используется спецификация %d (десятичное целое). Между знаком % и символом d могут находиться:

- знак – , который указывает, что для выводимого параметра должно быть выравнивание влево в своем поле (по умолчанию выравнивание вправо);
- число, задающее минимальный размер поля. Если аргумент занимает меньше позиций, чем указывает это число, то свободные позиции заполняются пробелами (справа или слева в зависимости от выравнивания). Если же аргумент фактически должен занимать больше позиций, чем указано этим числом, то под него отводится столько, сколько нужно.

```
#include <iostream>
using namespace std;
int main(){
int x=24, y=20225;
printf("|%4d|\n|%4d|\n",x,y);
}
```

Ввод/вывод

Для вывода вещественных чисел используется спецификации:

- ***%f*** - вещественное десятичное число в формате с точкой (например, 82.312);
- ***%e*** - вещественное десятичное число в экспоненциальном (научном) формате (например, 0.82312E2);
- ***%g*** - вещественное десятичное число, выбирается наиболее короткий формат %f или %e.

Также можно использовать знак "-" (выравнивание числа влево) и задавать минимальный размер поля между знаком % и символом формата. Кроме того, для вещественных чисел в спецификации могут быть:

- точка, которая отделяет размер поля от количества цифр, которое нужно вывести после десятичной точки;
- число после точки – максимальное количество знаков после десятичной точки. Если фактическое число знаков больше, то число округляется. Если меньше – дополняется нулями.

Если указывается минимальный размер поля, то он рассчитывается ***для всего числа*** (включая знак и десятичную точку).

```
double x=-56.489;
printf("x=%10.6f\n",x);
```

общее число символов

число символов после запятой

```
x=-56.489000
```

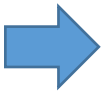
6

10

Ввод/вывод

Для вывода символов и строк используются следующие спецификаторы:
%c - вывод одного символа.
%s - вывод строки.

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main(){
    char abc ='A';
    char str[]="Hello, world!";
    printf("%20s \n %4c", str,  abc);
}
```



```
        Hello, world!
   A
```

Ввод/вывод

В классическом языке С для ввода данных используется функция ***scanf()***.
В Visual Studio 2013 и старше (в зависимости от настроек) компилятор может отказаться работать с этой функцией, и потребовать использовать вместо нее более безопасную функцию ***scanf_s***. Эта функция имеет дополнительный параметр – размер буфера (необязательный), который позволяет проконтролировать ввод и избежать изменения области памяти, которая вводимому объекту не принадлежит. Например, при вводе строки эта функция контролирует, чтобы вводимые данные не вышли за пределы памяти, отведенной под эту строку. Для ввода чисел размер буфера можно не указывать. Тогда формат использования функции ***scanf_s*** будет такой же, как и у классической функции ***scanf()***:

```
scanf("управляющая строка",список указателей на переменные);
scanf_s("управляющая строка",список указателей на переменные);
```

Функция ***scanf*** (или ***scanf_s***) должна считать текст из консоли, преобразовать его в данные нужного типа и разместить их в соответствующие ячейки памяти. Поэтому аргументы этой функции должны быть указателями на соответствующие переменные. Пока можете представлять себе, что указатель – это адрес переменной в памяти. Операция получения указателя на переменную (взятие адреса переменной) обозначается знаком ***&***.

Ввод/вывод

Спецификация преобразования	Описание
%d	десятичное целое число
%o	восьмеричное целое число
%x	шестнадцатеричное целое число
%u	десятичное целое число без знака
%lld	целое число типа long long
%f	вещественное десятичное число в формате с плавающей точкой
%lf	вещественное число типа double
%p	указатель (шестнадцатеричное число)
%c	одиночный символ
%s	строка символов

Ввод/вывод

В управляющую строку могут также включаться символы табуляции и перехода на новую строку (все они игнорируются), а также обычные символы, кроме символа **%** (считается, что они должны совпадать с очередными символами во входном потоке). Как и в случае вывода, устанавливается соответствие между спецификацией преобразования и аргументом в списке указателей в порядке их следования:

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main(){
    setlocale(LC_ALL, "rus");
    int k;
    float x;
    printf("Введите значения k и x через пробел: ");
    scanf("%d%f", &k, &x);
    printf("k=%d x=%f", k, x);
}
```

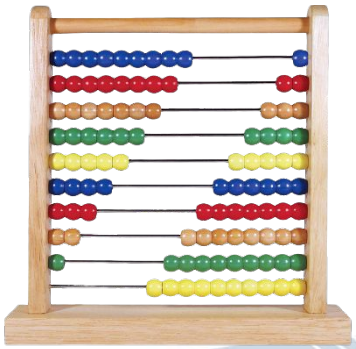
```
Введите значения k и x через пробел: 5 6,12
k=5 x=6,120000
```

Системы счисления

В компьютере данные представляются в двоичной системе, а программисты зачастую используют шестнадцатеричную систему. Но все эти способы записи чисел относятся к одному виду – к позиционным системам счисления. В них значение цифры зависит от ее положения (разряда) в числе.

Количество различных цифр, которые можно использовать для записи числа, называется ее основанием. Если основание системы счисления n, то имеется всего n цифр от ***0*** до ***n-1***. Поэтому в десятичной системе используется ***10*** цифр: ***0,1, 2, … 9***. В двоичной системе две цифры: ***0*** и ***1***. В шестнадцатеричной – 16 цифр: ***0-9, A-F*** (т.е. A соответствует десятичной 10, B – 11, … F – 15).

Позиции цифр в числе можно мысленно пронумеровать справа налево, начиная с 0. Т.е. крайняя правая позиция имеет номер 0, левее ее – номер 1, еще левее – номер 2, и т.д.
Значение числа получается как сумма произведений цифры на основание системы счисления в степени «номер позиции»:



Число=Сумма(цифра*основание$^{\text{позиция}}$)

Системы счисления

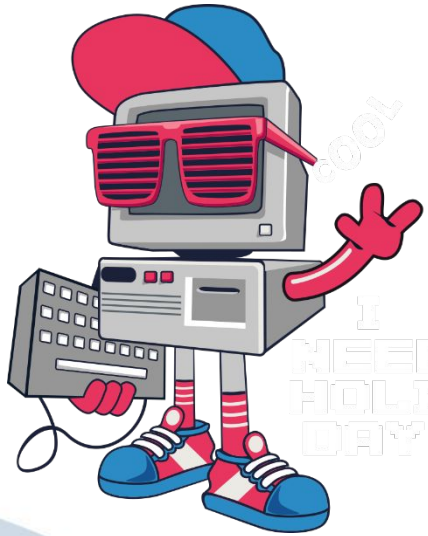
Примеры.

$367_{10}=3*10^2+6*10^1+7*10^0$

$11011_2=1*2^4+1*2^3+0*2^2+1*2^1+1*2^0=16+8+2+1=27_{10}$

$167_8=1*8^2+6*8^1+7*8^0=64+48+7=119_{10}$

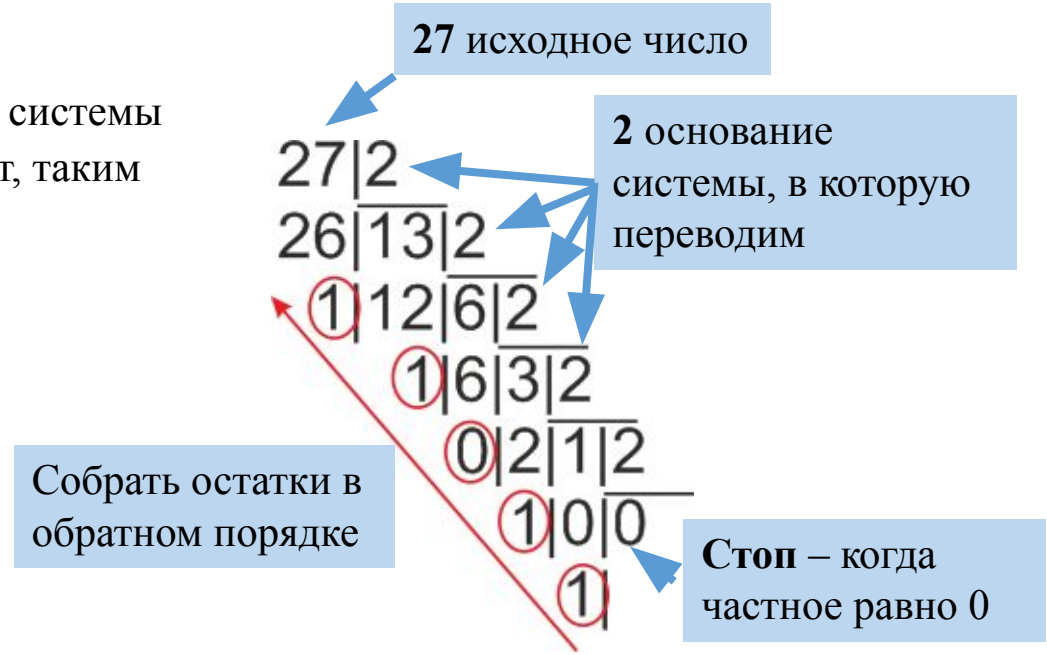
$2A4_{16}=2*16^2+10*16^1+4*16^0=512+160+4=676_{10}$

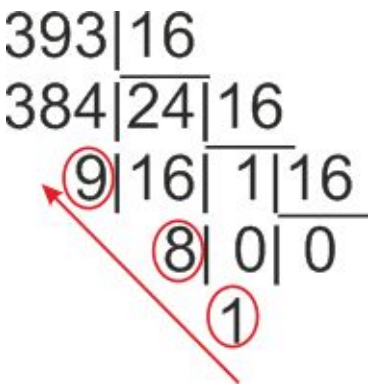


Системы счисления

Для перевода из десятичной системы в любую другую нужно выполнять целочисленное деление исходного числа на основание той системы счисления, в которую нужно перевести число. При этом важен остаток от деления и частное. Частное делится на основание системы счисления до тех пор, пока не будет получен 0. После этого все остатки нужно выписать в обратном порядке - это и будет число в новой системе счисления.

Перевод - числа 27 из десятичной системы счисления в двоичную. Результат, таким образом, следующий: $27_{10}=11011_2$





Битовые операции

Отрицательное число имеет единицу в старшем бите. Но это еще не все. Отрицательное число хранится в дополнительном коде (и эта единица в знаковом разряде появляется автоматически при переводе в дополнительный код). Чтобы представить число в дополнительном коде, нужно выполнить следующее:

- перевести в двоичную систему счисления модуль числа и приписать слева незначащие нули, заполнив нужное число разрядов;
- инвертировать биты числа (т.е. заменить 1 на 0, а 0 на 1);
- прибавить 1 к полученному числу.

Битовые операции

Пример. Для размещения в одном байте представить в дополнительном коде число ***-26***.

1 ***Переводим в двоичную систему*** счисления десятичное число ***26***.
11010
Дополняем слева незначащими нулями до 8 разрядов (т.к. по условию размещать число будем в одном байте):
00011010

2 ***Инвертируем*** биты числа:
11100101

3 ***Прибавляем единицу***. Тут нужно учитывать, что в двоичной системе ***1+1=10***. Поэтому, если в разряде две единицы, то получаем 0, а единица идет в старший разряд...

```
  11100101
+         1
----------
  11100110
```

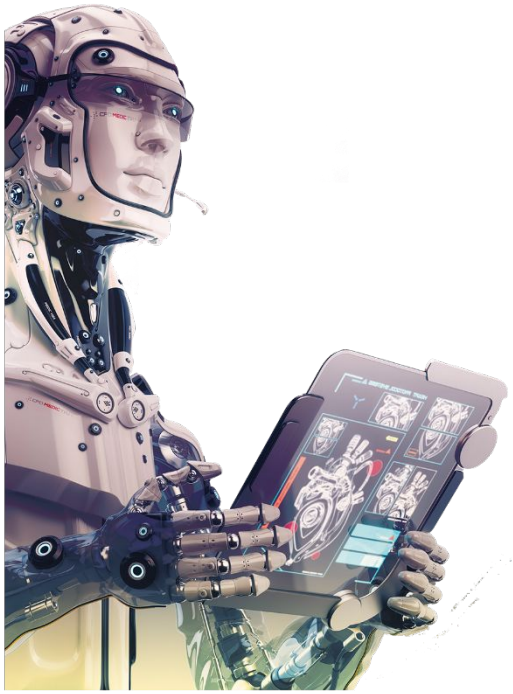
Итог: -26, размещенное в одном байте в виде: ***11100110***

Битовые операции

Обратное преобразование

Чтобы из дополнительного кода получить модуль числа, нужно проделать те же действия: инвертировать и прибавить единицу. Полученное двоичное представление перевести в удобную нам систему счисления.

Например, пусть число в памяти имеет вид: ***11100110***. После инвертирования получаем: ***00011001***. Прибавим единицу и получим: ***00011010***. Переводим в десятичную систему счисления: $2^1+2^3+2^4=2+8+16=26$



Битовые операции

~ (Инверсия)

Все биты аргумента меняются на противоположные (0 – на 1, а 1 – на 0).

$89_{10} = 01011001_2$
$\sim 89_{10} = 10100110_2 = 166_{10}$

```
unsigned char x=~89;
// значение переменной x = 166
cout << x << "\t" << (int)x;
```

& (Битовое И)

Для каждой пары соответствующих битов аргументов выполняется «умножение» по правилам: 1*1=1; 1*0=0; 0*0=0. Таким образом, в результате единица будет только в том разряде, в котором ***у обоих аргументов стоят единицы***.

$89_{10} = 01011001_2$
$101_{10} = 01100101_2$
$x\&y = 01000001_2 = 65_{10}$

```
unsigned char x=89,y=101;
unsigned char k=x&y;
//значение переменной k=65
cout << k << "\t" << (int)k;
```

Битовые операции

| (Битовое ИЛИ)

Для каждой пары битов аргументов выполняется «сложение» по правилам: 1+1=1; 1+0=1; 0+0=0. Таким образом, в результате единица будет в том разряде, где ***хотя бы у одного аргумента в этом разряде имеется единица***.

$89_{10} = 01011001_2$
$101_{10} = 01100101_2$
$x|y = 01111101_2 = 125_{10}$

```
unsigned char x=89,y=101;
unsigned char k=x|y;
//значение переменной k=125
cout << k << "\t" << (int)k;
```

^ (Битовое исключающее ИЛИ)

Для каждой пары битов аргументов выполняется «сложение» по правилам: 1+1=0; 1+0=1; 0+0=0. Таким образом, в результате единица будет в том разряде, где только у одного аргумента в этом разряде имеется единица.

$89_{10} = 01011001_2$
$101_{10} = 01100101_2$
$x^y = 00111100_2 = 60_{10}$

```
unsigned char x=89,y=101;
unsigned char k=x^y;
//значение переменной k=60
cout << k << "\t" << (int)k;
```

Битовые операции

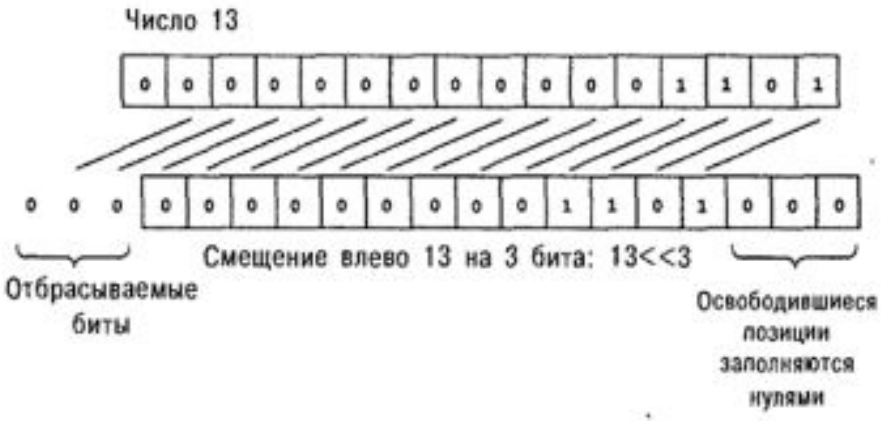
<< (Сдвиг влево)

Формат: ***b<<c*** При этом осуществляется сдвиг значения b влево на c разрядов. В освободившиеся справа разряды b заносятся нули. Биты, «сдвинутые» за пределы разрядной сетки переменной, пропадают. Сдвиг влево эквивалентен ***умножению на степень двойки***, причем выполняется процессором гораздо быстрее. При этом не учитывается переполнение или потеря значимости. Если операнд с отрицательный или больше длины операнда b в битах, то результат операции сдвига не определен.

$25_{10} = 00011001_2$

$25_{10}<<2 = 01100100_2 = 100_{10} = 25 * 2^2$

```
signed char k= 25<<2;
//значение переменной k=100
cout << k << "\t" << (int)k;
```



Битовые операции

>> (Сдвиг вправо)

Формат: b>>c При этом осуществляется сдвиг значения b вправо на c разрядов. Для заполнения позиций слева используется знаковый бит (***ноль*** для ***положительных*** b и ***единица - для отрицательных*** b). Биты, «сдвинутые» за пределы разрядной сетки переменной, пропадают. Сдвиг вправо ***положительных*** чисел эквивалентен ***делению на степень двойки***, причем выполняется процессором гораздо быстрее.

$25_{10} = 00011001_2$

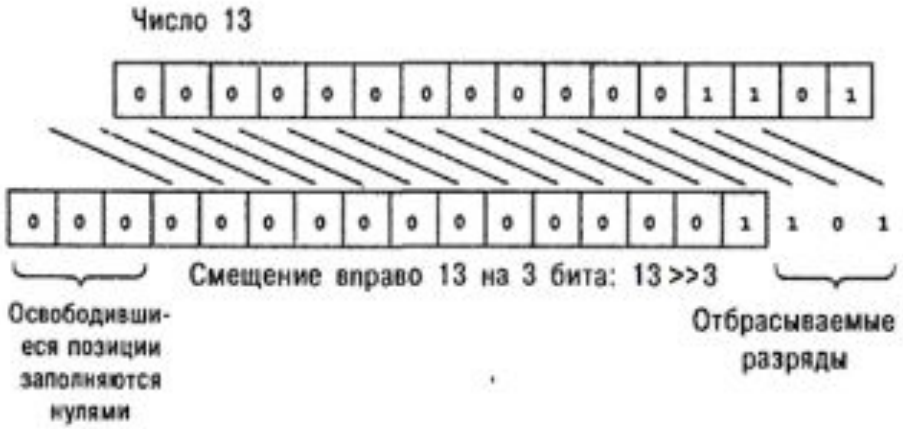
$25_{10}>>3 = 00000011_2 = 3_{10} = 25 / 2^3$

```
signed char k= 25>>3;
//значение переменной k=3
cout << k << "\t" << (int)k;
```

$-14_{10} = 11110010_2$

$-14_{10}>>2 = 11111100_2 = -4_{10} = -14 / 2^2$

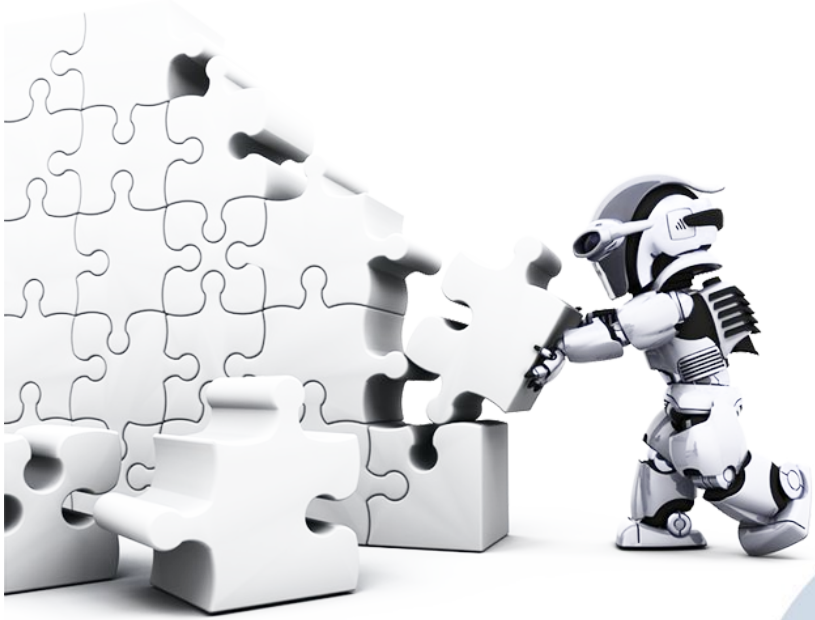
```
signed char k= -14>>2;
//значение переменной k=-4
cout << k << "\t" << (int)k;
```



Битовые операции

Алгоритм проведения битовых операций:

1. Перевести десятичное в двоичное.
2. Дополнить до разряда нулями.
отрицательное число инвертировать и прибавить единицу.
3. Произвести операцию.
4. В случае если в знаковом разряде 1 - результат инвертировать и прибавить единицу.
5. Перевести в десятичное отрицательное число.



Битовые операции

Примеры использования битовых операций

1 В программах управления устройствами или при реализации сетевых протоколов часто бывает необходимо проверить, установлен ли определенный бит числа (т. е. равен ли он единице).

Для реализации этой цели создают "маску" - специальное число, имеющее единицу только в проверяемом разряде, а остальные разряды – нулевые. Исследуемое число подвергают операции «Битовое И» с маской. Если полученный результат равен 0, то бит не был установлен в исходном числе, а если отличен от нуля – то был.

2 Операция ^ (битовое исключающее ИЛИ) обладает одним важным свойством: если ее дважды применить к одному и тому же объекту, то она возвращает его в исходное состояние. Это свойство можно использовать для шифрования информации по ключу. Например, некто желает передать зашифрованное сообщение. Ко всем символам этого сообщения он применяет операцию ^ с фиксированным числом (ключом). В результате получается набор символов, который нельзя прочитать. Он передается получателю.

Получатель также знает ключ. Получив зашифрованную строку, он снова применяет к ней операцию ^ с ключом, и получает исходное послание.

Практика

1 Введите значения двух вещественных чисел и вычислите их произведение. Вывод нужно оформить в виде числа с фиксированной точкой, причем в дробной части выводится 3 десятичных знака, используйте для ввода и вывода функции scanf () и printf().

2 Пользователь вводит с клавиатуры денежную сумму в рублях и копейках (рубли и копейки вводятся в разные переменные). Программа должна откорректировать введенную сумму в правильную форму. Например, если пользователь ввел 11 и 150 , то программа должна вывести 12 р. 50 к. Использовать для вывода функцию printf().

3 Программа должна вычислять скорость, с которой бегун пробежал дистанцию. Пользователь вводит длину дистанции в метрах и, через пробел, время в формате минуты:секунды.
Скорость выводится в км/час и должна быть округлена до двух знаков после десятичной точки.
Подсказка: чтобы пропустить символы во входном потоке, добавьте их в строку форматирования функции scanf().
Например, ввод времени в данной задаче может выглядеть так:
scanf("%d:%d",&min,&sec);.

```
Введите метры и время в формате минуты:секунды 1000 3:35
Скорость = 16,74
--------------------------------
Process exited after 9.209 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Практика

4 Переведите в десятичную систему следующие числа:

10101_2

257_8

$1BA_{16}$

5 Перевидите число (90 + ваш номер по списку группы) в двоичную, восьмеричную и шестнадцатеричную систему

Домашнее задание

Объявите две вещественных переменных и инициализируйте их значениями 2022.242 и 11.9 Выведите эти числа друг под другом, выполнив выравнивание по правому краю. Округлите до двух десятичных знаков в дробной части.

```
2022.24
  11.90
```