

## Алгоритмы и блок-схемы

Итак, к настоящему моменту мы стали писать простые программы. Давайте взглянем на этот процесс с чуть более абстрактной точки зрения. Наши программы

- принимают на вход какую-то информацию
- выполняют анализ и преобразование этой информации
- выводят информацию на экран

Тем самым, наша программа реализует алгоритм – набор инструкций о том, в какой последовательности нужно выполнить действия при переработке исходного материала в требуемый результат. Программа – это запись инструкций на данном языке программирования.

На практике получили известность два способа изображения алгоритмов:





- в виде пошагового словесного описания на естественном языке или языке программирования;
- в виде блок-схем.

Первый из этих способов страдает многословностью и отсутствием наглядности. Второй, напротив, оказался очень удобным средством изображения алгоритмов и получил широкое распространение в научной и учебной литературе.

**Блок-схема** – это граф, представляющий последовательность блоков, предписывающих выполнение определенных операций, и связей между этими блоками. Внутри блоков указывается информация об операциях, подлежащих выполнению. Движение по ребрам графа соответствует процессу обработки и изменения информации.

При соединении блоков следует использовать только вертикальные и горизонтальные линии потоков. Они должны быть обязательно помечены стрелками.

### Наименование    Обозначение    Функция

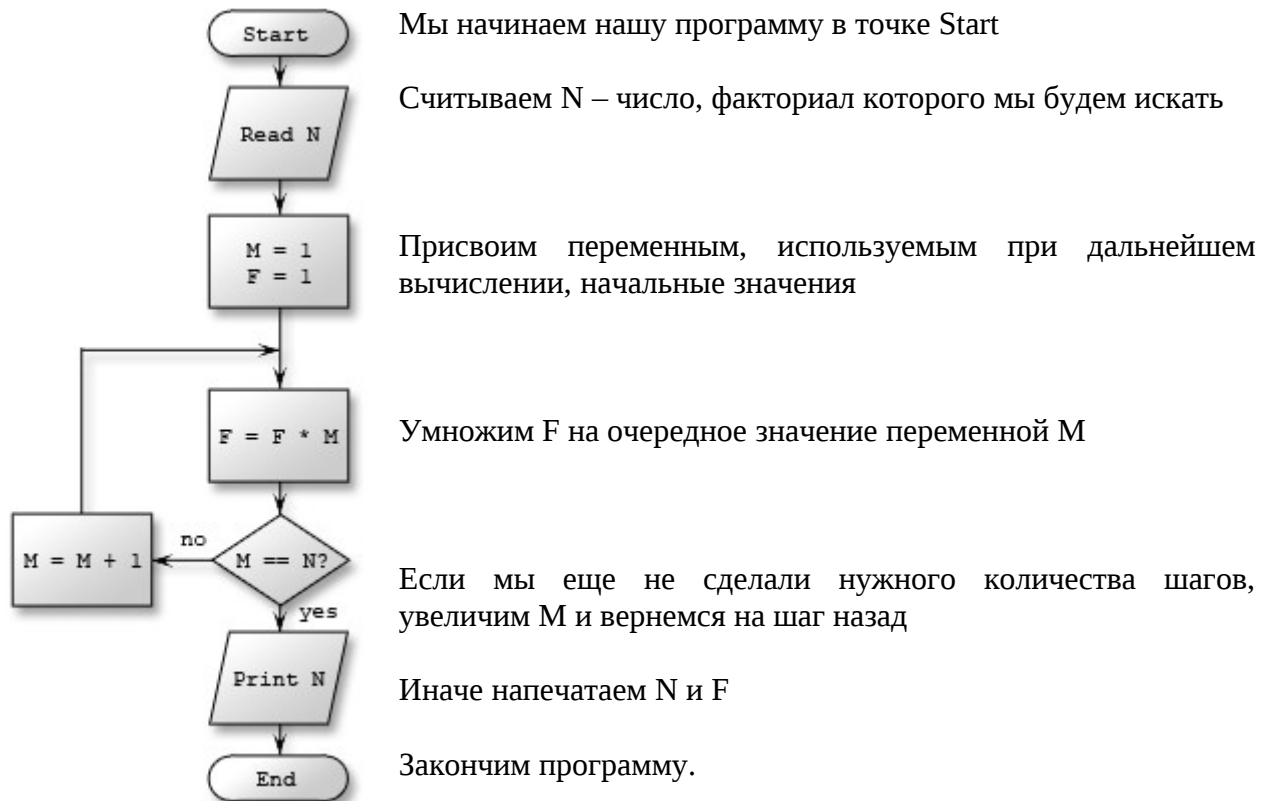
Пуск- остановка		Элемент отображает вход из внешней среды или выход из нее (наиболее частое применение – начало и конец программы). Внутри фигуры записывается соответствующее действие.
Процесс		Выполнение одной или нескольких операций, обработка данных любого вида (изменение значения данных, формы представления, расположения). Внутри фигуры записывают непосредственно сами операции, например, операцию присваивания: $a = 10 * b + c$ .
Решение		Отображает решение с одним входом и двумя или более альтернативными выходами, из которых только один может быть выбран после вычисления условий, определенных внутри этого элемента. Вход в элемент обозначается линией, входящей обычно в верхнюю вершину элемента. Если выходов два или три то обычно каждый выход обозначается линией, выходящей из оставшихся вершин (боковых и нижней). Если выходов больше трех, то их следует показывать одной линией, выходящей из вершины (чаще нижней) элемента, которая затем разветвляется. Примеры решения – условные операторы <i>if</i> (два выхода: <i>true</i> , <i>false</i> ).
Данные (ввод-вывод)		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Данный символ не определяет носителя данных (для указания типа носителя данных используются

специфические символы).

Другие элементы блок-схем существуют и регламентируются соответствующими стандартами, но для наших целей пока достаточно приведенных выше.

Примером использования блок-схемы может служить блок-схема алгоритма вычисления факториала:

Разберем его по шагам:



Язык блок-схем является удобным средством для проектирования и представления ваших замыслов в понятном для окружающих виде.

## Функции

Пусть в процессе написания программы нам захотелось выводить числа не как 2, 3, 4, а как +2, +3, +4. При этом отрицательные числа (-2, -17) так и должны писаться с «-». Можно, конечно, каждый раз писать вместо `cout << number;`

```
if (number >= 0) {  
    cout << "+" << number;  
} else {  
    cout << number;  
}
```

Но это ведь неудобно! Для таких случаев во всех языках программирования используют **функции** (или иногда они называются **процедуры**). Вспомните, мы уже пользовались чужими функциями в предыдущем листке, когда математические операции округления, извлечения корня, возведения в степень были нужны для решения задач. Настало время научиться писать свои функции.

Напишем в данном случае:

```
void printNumber(int number) {
    if (number >= 0) {
        cout << "+" << number;
    } else {
        cout << number;
    }
}
```

А main теперь может выглядеть как-то так:

```
int main() {
    printNumber(3);
    printNumber(4);
    printNumber(-17);
}
```

Как мы видим, функция называется «printNumber». После неё в скобках указаны *аргументы* функции. В данном случае мы сказали, что функции для работы нужно одно целое число `int number`. Если бы мы хотели написать функцию, которая выводит на экран остаток от деления первого аргумента на второй, то заголовок функции выглядел бы как-то так:

```
void printReminder(int a, int b)
```

Пока что я ни слова не сказал про то, что означает `void` в заголовке функции. Об этом чуть позже. После заголовка идёт совершенно обычная «программа», как те, которые вы уже писали. Эта «программа» называется *телом функции*. Таким образом, тело функции `printNumber` – строки

```
{
    if (number >= 0) {
        cout << "+" << number;
    } else {
        cout << number;
    }
}
```

## В. Баяны

Необходимо написать функцию, которая рисует баян вида `[:|:]`, в котором количество палочек «|» в середине равно аргументу функции. С помощью этой функции напишите программу, которая выведет вот такой баян: `[:|||:]`

## С. Язык «Баяны»

Язык программирования «Баяны» умеет рисовать баяны. Программой в нём является последовательность целых неотрицательных чисел из не менее чем одного числа. Пусть  $m$  – первое число. Тогда программа сразу пишет в консоль баян с  $m$  палочками посередине, после чего начинает новую строку. После этого считывается ещё  $m$  чисел, и каждый раз рисуется баян с соответствующим числом палочек. Если же очередное число — 0, то вместо вывода баяна мы начинаем новую строку. Реализуйте язык «баяны».

#### D. Бинарный вывод

Напишите функцию, которая в качестве аргумента получает натуральное число и выводит его в двоичном виде. С помощью этой функции напишите программу, которая переводит в двоичную систему числа 6, 17, 53 и 3535874.

#### Что такое void и bool?

Пойдем с конца. Мы говорили всегда, что внутри компьютера все представляется в виде единичек и ноликов — двоичных битов, символизирующих истину и ложь. Для представления логической информации, которая может быть или истинной, или ложной, нам ничего, кроме одного бита не нужно. Тип данных, который содержит только «истину» или «ложь» (true или false по-английски), называется логическим или булевым типом данных — bool.

Теперь откроем словарь и выясним, что void значит «пустота, вакуум, пропуск, пустое место». void в заголовках наших функций означал, что функция *ничего не возвращает*. А что она могла бы вернуть? Например, напишем функцию, которая бы вычисляла квадрат числа:

```
int sqr(int x) {
    return x * x;
}
```

Тогда в main можно было-бы написать так:

```
cout << sqr(2).
```

Или даже так:

```
int four = sqr(2).
```

Или так:

```
int sixteen = sqr(four). Или так: sixteen = sqr(sqr(2)).
```

Ещё один пример. В одном из предыдущих листков была задача вычисления числа сочетаний из  $n$  элементов по  $k$ . Один из способов решения - вычислить факториалы трех величин:  $n$ ,  $k$  и  $n-k$ , после чего их поделить друг на друга правильным образом. Большинство из вас писали несколько циклов, в результате чего одна и та же процедура вычисления факториала числа была написана трижды! Расточительно, не правда ли?

Гораздо правильнее в таком случае написать отдельную функцию, которая бы вычисляла факториал своего единственного аргумента:

```
int factorial (int n)
{
    int f = 1, i;
    for (i = 2; i <= n; ++i)
    {
        f = f * i;
    }
    return f;
}
```

Теперь посмотрим, во что превратилась программа, которая вычисляет число сочетаний:

```
int factorial (int n)
```

```

{
    int f = 1, i;
    for (i = 2; i <= n; ++i)
    {
        f = f * i;
    }
    return f;
}

int main ()
{
    int n, k;
    cin >> n >> k;
    cout << factorial(n) / (factorial(k) * factorial(n-k)) <<
endl;
    return 0;
}

```

Стало немножко короче и понятнее, неправда ли?

### **Особенности работы return**

Если вдруг кто-то считает, что return может быть написан только в конце тела функции, то вынужден его огорчить — это не так. Например:

```

double max (double a, double b) {
    if (a > b) {
        return a;
    }
    return b;
}

```

Эта функция возвращает максимум из двух своих аргументов. Работает она потому, что как только программа доходит до какого-либо из return, функция завершает свою работу и возвращает результат.

#### **Е. Отрицательная степень**

Дано целое  $a$  и целое число  $n$ .

Вычислите  $a^n$ . Решение оформите в виде функции `double power(int a, int n)`.

Стандартной функцией возведения в степень пользоваться нельзя.

**Ввод**      **Вывод**

2 3          8

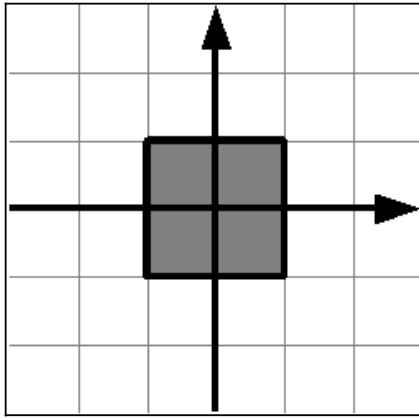
2 -3        0.125

#### **Ф. Длина отрезка**

Даны четыре действительных числа:  $x_1, y_1, x_2, y_2$ . Напишите функцию `double distance(double x1, double y1, double x2, double y2)`, вычисляющую расстояние между точками  $(x_1, y_1)$  и  $(x_2, y_2)$ . Считайте четыре действительных числа и выведите результат работы этой функции.

#### **Г. Принадлежит ли точка квадрату-1**

Даны два действительных числа  $x$  и  $y$ . Проверьте, принадлежит ли точка с координатами  $(x, y)$  заштрихованному квадрату (включая его границу). Если точка принадлежит квадрату, выведите слово YES, иначе выведите слово NO.

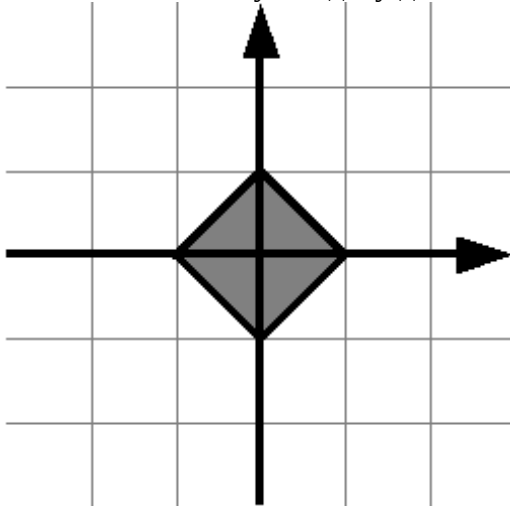


На рисунке сетка проведена с шагом 1. Решение должно содержать функцию `bool IsPointInSquare(double x, double y)`, возвращающую `true`, если точка принадлежит квадрату, и `false`, если не принадлежит.

Функция `main` должна считать координаты точки, вызвать функцию `IsPointInSquare` и в зависимости от возвращенного значения вывести на экран необходимое сообщение.

#### **Н. Принадлежит ли точка квадрату-2**

Решите аналогичную задачу для такого квадрата:



Решение должно соответствовать требованиям к решению задачи G.

#### **И. Принадлежит ли точка кругу**

Даны пять действительных чисел:  $x$ ,  $y$ ,  $x_c$ ,  $y_c$ ,  $r$ . Проверьте, принадлежит ли точка  $(x,y)$  кругу с центром  $(x_c, y_c)$  и радиусом  $r$ .

Решение оформите в виде функции `bool IsPointInCircle(double x, double y, double x_c, double y_c, double r)`.

Решение должно соответствовать требованиям для решения задачи G.

#### **Ж. Минимальный делитель числа**

Дано натуральное число  $n > 1$ . Выведите его наименьший делитель, отличный от 1.

Решение оформите в виде функции `int MinDivisor(int n)`

#### **К. Проверка числа на простоту**

Дано натуральное число  $n > 1$ . Проверьте, является ли оно простым. Программа должна вывести слово YES, если число простое, и NO, если число составное.

Решение оформите в виде функции `bool IsPrime(int n)`, которая возвращает `true` для простых чисел и `false` для составных чисел.

## L. Знак и модуль

Напишите функцию `int sgn(double x)`, которая возвращает 1, если  $x > 0$ , -1, если  $x < 0$ , и 0 иначе.

С помощью этой функции напишите функцию `double abs(double x)`, которая возвращает модуль числа  $x$ . Функция `abs` не может содержать условных операторов.

С помощью этих двух функций напишите программу, которая выводит знак введенного числа и его модуль.

## Рекурсия

```
void ShortStory()
{
    cout << "У попа была собака, он ее любил." << endl;
    cout << "Она съела кусок мяса, он ее убил," << endl;
    cout << "В землю закопал и надпись написал:" <<
endl;
    ShortStory();
}
```

Как мы видели выше, функция может вызывать другую функцию. Но функция также может вызывать и саму себя! Рассмотрим это на примере функции вычисления факториала. Хорошо известно, что  $0! = 1$ ,  $1! = 1$ . А как вычислить величину  $n!$  для большого  $n$ ? Если бы мы могли вычислить величину  $(n-1)!$ , то тогда мы легко вычислим  $n!$ , поскольку  $n! = n * (n-1)!$ . Но как вычислить  $(n-1)!$ ? Если бы мы вычислили  $(n-2)!$ , то мы сможем вычислить и  $(n-1)! = (n-1) * (n-2)!$ . А как вычислить  $(n-2)!$ ? Если бы... В конце концов, мы дойдем до величины  $0!$ , которая равна 1. Таким образом, для вычисления факториала мы можем использовать значение факториала для меньшего числа. Это можно сделать и в программе на C++:

```
int factorial (int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * factorial(n - 1);
    }
}
```

Подобный прием (вызов функцией самой себя) называется рекурсией, а сама функция называется рекурсивной.

Рекурсивные функции являются мощным механизмом в программировании. К сожалению, они не всегда эффективны (об этом речь пойдет позже). Также часто использование рекурсии приводит к ошибкам, наиболее распространенная из таких ошибок – бесконечная рекурсия, когда цепочка вызовов функций никогда не завершается и продолжается, пока не кончится свободная память в компьютере. Пример бесконечной рекурсии приведен в эпиграфе к этому разделу. Две наиболее распространенные причины для бесконечной рекурсии:

1. Неправильное оформление выхода из рекурсии. Например, если мы в программе вычисления факториала забудем поставить проверку `if (n == 0)`, то `factorial(0)` вызовет `factorial(-1)`, тот вызовет `factorial(-2)` и т.д.

2. Рекурсивный вызов с неправильными параметрами. Например, если функция `factorial(n)` будет вызывать `factorial(n)`, то также получится бесконечная цепочка.

Поэтому при разработке рекурсивной функции необходимо прежде всего оформлять условия завершения рекурсии и думать, почему рекурсия когда-либо завершит работу.

**Общее требование к задачам с рекурсией:** когда вы сдаёте задачу, использующую рекурсию, вы устно обосновываете, почему рекурсия всегда когда-нибудь закончится.

## Упражнения на рекурсию

### М. Возведение в степень

Дано действительное положительное число  $a$  и целое неотрицательное число  $n$ . Вычислите  $a^n$  не используя циклы и стандартную функцию `pow`, а используя рекуррентное соотношение  $a^n = a * a^{n-1}$ .

Решение оформите в виде функции `double power(double a, int n)`.

Ввод	Вывод
2 3	8

### Н. Сложение без сложения

Напишите рекурсивную функцию `int sum(int a, int b)`, возвращающую сумму двух целых неотрицательных чисел. Из всех арифметических операций допускаются только `+1` и `-1`. Также нельзя использовать циклы.

Ввод	Вывод
2 2	4

### О. Числа Фибоначчи

Последовательность Фибоначчи определена следующим образом:  $F_0=1, F_1=1, F_n=F_{n-1}+F_{n-2}$ , при  $n>1$ .

Начало ряда Фибоначчи выглядит следующим образом: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Напишите функцию `int Fib(int n)`, которая по данному целому неотрицательному  $n$  возвращает  $F_n$ .

Ввод	Вывод
6	13

### Р. Возведение в степень

Возводить в степень можно гораздо быстрее, чем за  $n$  умножений! Для этого нужно воспользоваться следующими рекуррентными соотношениями:

$$a^n = (a^2)^{n/2} \quad \text{при четном } n,$$

$$a^n = a * a^{n-1} \quad \text{при нечетном } n.$$



Реализуйте алгоритм быстрого возведения в степень.

Ввод	Вывод
1.0000000001 1000000000	2.71828

#### Q. Алгоритм Евклида

Для быстрого вычисления наибольшего общего делителя двух чисел используют алгоритм Евклида. Он построен на следующем соотношении:  $\text{НОД}(a,b)=\text{НОД}(a \% b,b)$ .

Реализуйте рекурсивный алгоритм Евклида в виде функции `int gcd(int a, int b)`.

Ввод	Вывод
12 16	4

#### R. Сумма делителей

Для данного натурального числа  $n$  вычислите сумму всех его натуральных делителей, включая 1 и само число.

Решение оформите в виде функции `int SumOfTheDivisors (int n)`

Ввод	Вывод
6	12