Лекция 13

Вспомогательные алгоритмы. Функции

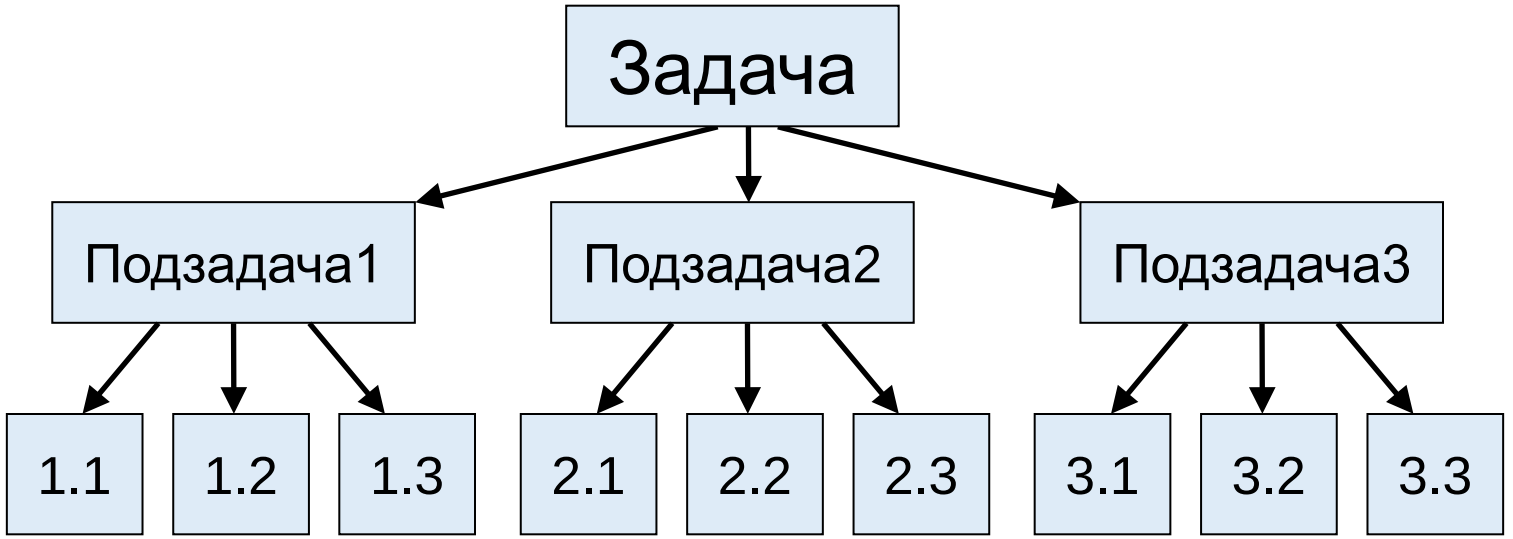
Преподаватель Палехова Ольга Александровна,
кафедра О7 БГТУ «Военмех»

Вспомогательные алгоритмы

Вспомогательный алгоритм – это алгоритм, предназначенный для использования в других алгоритмах.

Применение:

- выполнение одинаковых действий в разных местах программы
- разбивка задачи на подзадачи для лучшего восприятия

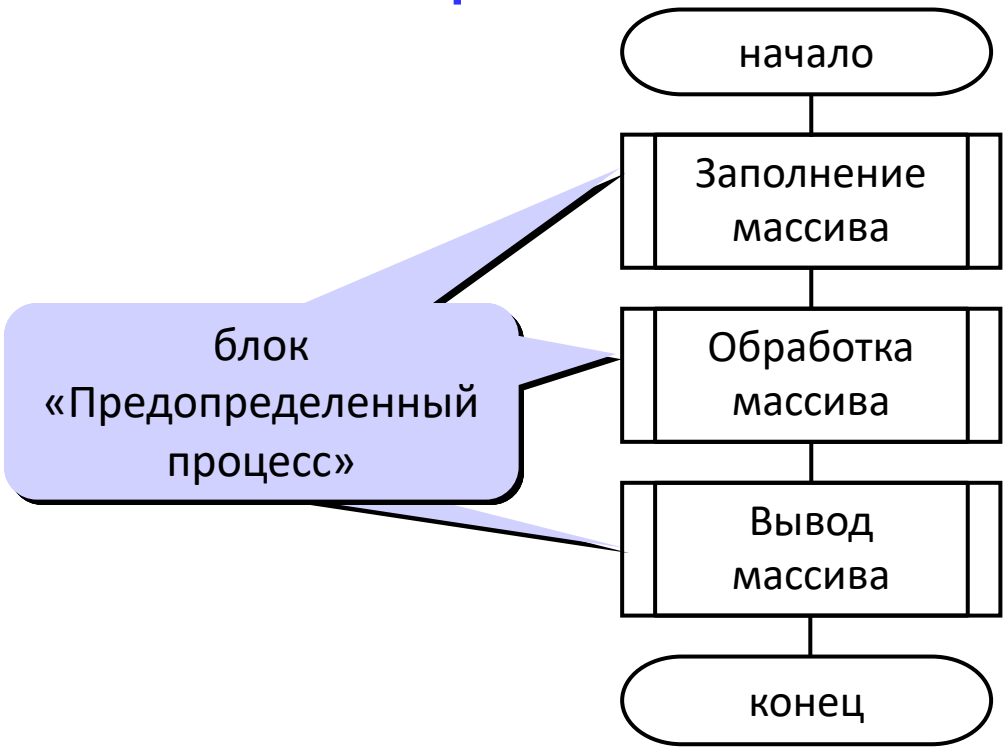


Вспомогательные алгоритмы

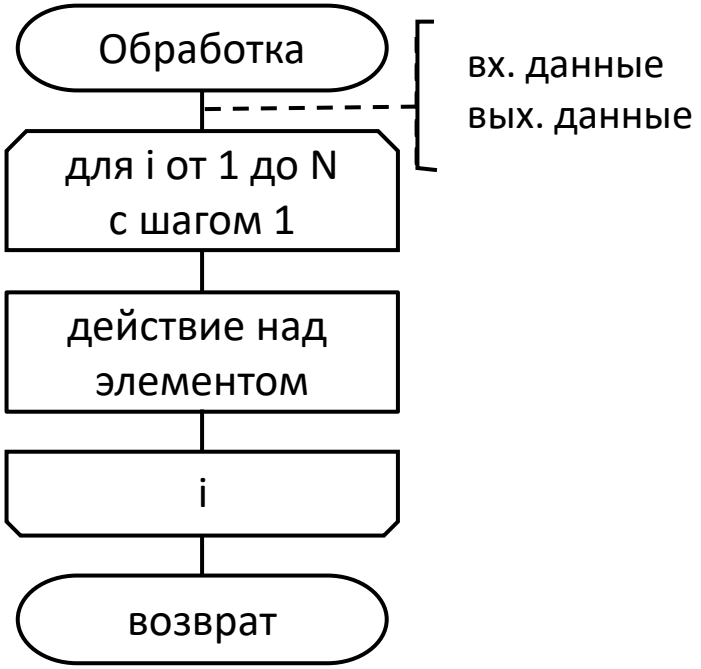
Алгоритм, из которого происходит обращение к вспомогательным алгоритмам, называют **основным алгоритмом** или **вызывающим алгоритмом.**

Если в самом вспомогательном алгоритме используются другие вспомогательные алгоритмы, то по отношению к ним данный вспомогательный алгоритм является основным.

Основной алгоритм:



Вспомогательный алгоритм:



Проектирование ПО

Два подхода к проектированию:

- **нисходящее проектирование** (проектирование **«сверху вниз»**) – подход, при котором разработка начинается с определения целей решения проблемы, с последующей детализацией (от общего к частному);
- **восходящее проектирование** (проектирование **«снизу вверх»**) – подход, при котором разработка идет от частного к общему

В большинстве случаев применяют оба подхода:

- сначала выполняют проектирование сверху-вниз, чтобы понять, что от каждой компоненты требуется и как она будет использоваться,
- затем для разработки общих хорошо отлаженных блоков выполняют проектирование снизу вверх,
- после чего возвращаются нисходящему подходу для объединения готовых блоков в единое целое.

Нисходящее проектирование

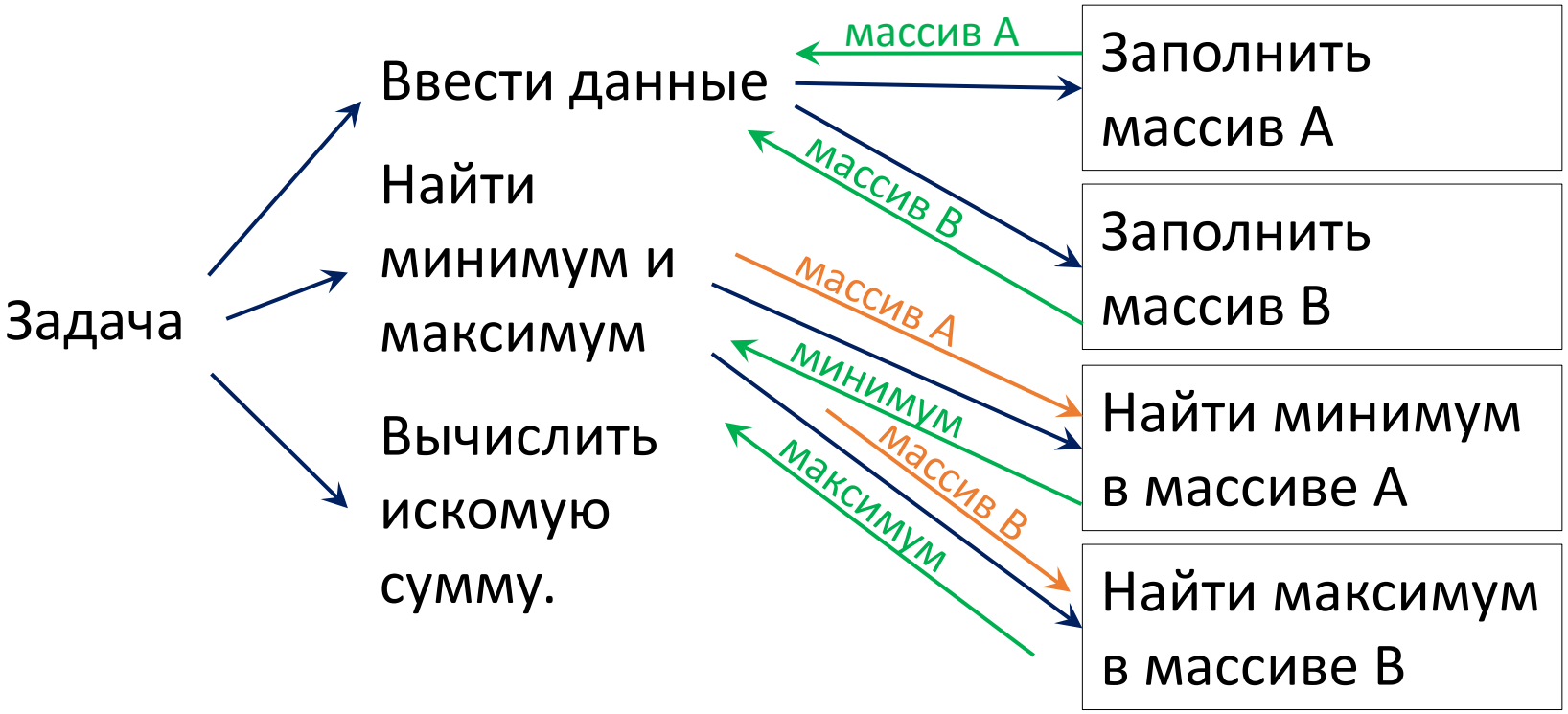
Порядок разработки:

1. выделить главную задачу, поделить ее на независимые подзадачи (блоки), определить функциональное **назначение** каждого блока;
2. определить связи между блоками, продумать средства взаимодействия (**интерфейс**) каждого блока с остальными, представить их как **входные** и **выходные данные;**
3. разработать структуру каждого блока, возможно, повторив пп.1-3
4. записать все блоки в форме вспомогательных алгоритмов.

Пример нисходящего проектирования

Задача: Найти сумму минимального элемента массива A(NA) и максимального элемента массива B(NB).

1. Разбиваем задачу на более мелкие:



2. Определяем назначение каждого блока
3. Определяем входные и выходные данные
4. Записываем каждый блок отдельным алгоритмом

Пример нисходящего проектирования

Вспомогательный алгоритм ввода массива ***a*** размера *NA*

Вход – нет
Выход – заполненный массив ***a***

```
цикл для i от 1 до NA с шагом 1
    ввести a_i
конец цикла
```

Вспомогательный алгоритм ввода массива ***b*** размера *NB*

Вход – нет
Выход – заполненная матрица ***b***

```
цикл для i от 1 до NB с шагом 1
    ввести b_i
конец цикла
```

Восходящее проектирование алгоритмов

Порядок разработки:

1. выделить в анализируемых алгоритмах одинаковое или похожее;
2. найти в них **общее** и **отличия;**
3. дать ***отличиям*** обозначения и использовать в качестве **входных** и **выходных данных** для проектируемого вспомогательного алгоритма;
4. общее записать в виде вспомогательного алгоритма.

Примеры восходящего проектирования

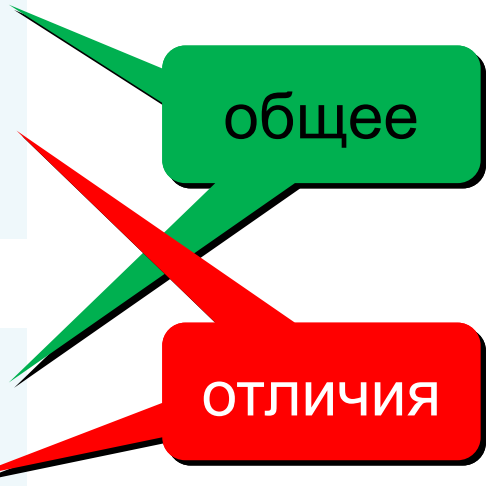
Пример 1. Ввод массива

Алгоритм ввода массива ***a*** размера *NA*

```
Цикл для i от 1 до NA с шагом 1
    ввести a_i
конец цикла
```

Алгоритм ввода массива ***b*** размера *NB*

```
Цикл для i от 1 до NB с шагом 1
    ввести b_i
конец цикла
```



Вспомогательный алгоритм Ввод_Массива

Вход – размер массива ***N***
Выход – заполненный массив ***array***

```
Цикл для i от 1 до N с шагом 1
    ввести array_i
конец цикла
```

Примеры восходящего проектирования

Пример 2. Поиск минимального и максимального элемента массива

Алгоритм поиска максимума в массиве ***a*** размера *NA*

```
эталон = a1
Цикл для i от 2 до NA с шагом 1
    если эталон < ai , то эталон = ai
конец цикла
```

Алгоритм поиска минимума в массиве ***b*** размера *NB*

```
эталон = b1
Цикл для i от 2 до NB с шагом 1
    если эталон > bi , то эталон = bi
конец цикла
```

общее

отличия

Примеры восходящего проектирования

Пример 2. Поиск минимального и максимального элемента массива

Вспомогательный алгоритм Min_Max_Array

Вход – размер массива ***N,*** массив ***array,*** вспомогательный алгоритм ***Надо_менять***
Выход – значение минимального или максимального элемента ***эталон***

```
эталон = array_1
Цикл для i от 2 до N с шагом 1
    если Надо_менять (эталон, array_i),
        то эталон = array_i
конец цикла
```

Функции языка Си

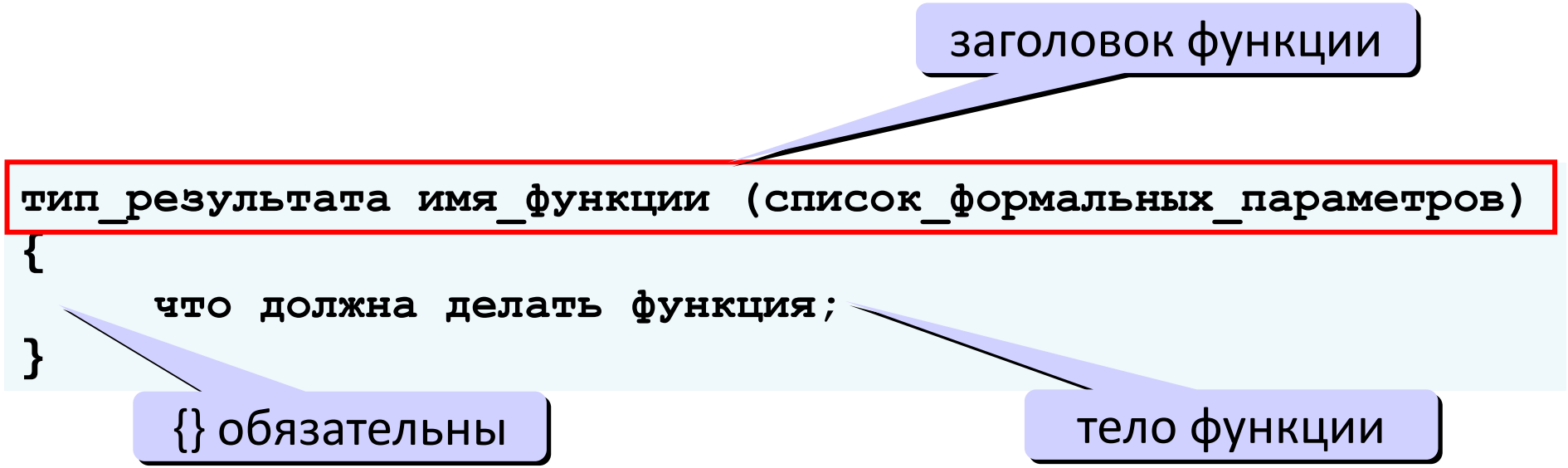
На языке Си вспомогательные алгоритмы программируются с помощью **функций**.

Функция — это самостоятельная именованная единица программы, реализующая некоторый законченный вспомогательный алгоритм.

Функция может вызываться из любой части программы столько раз, сколько нужно.

Главная функция программы – **main** – не может быть повторно вызвана в этой же программе.

Определение функции

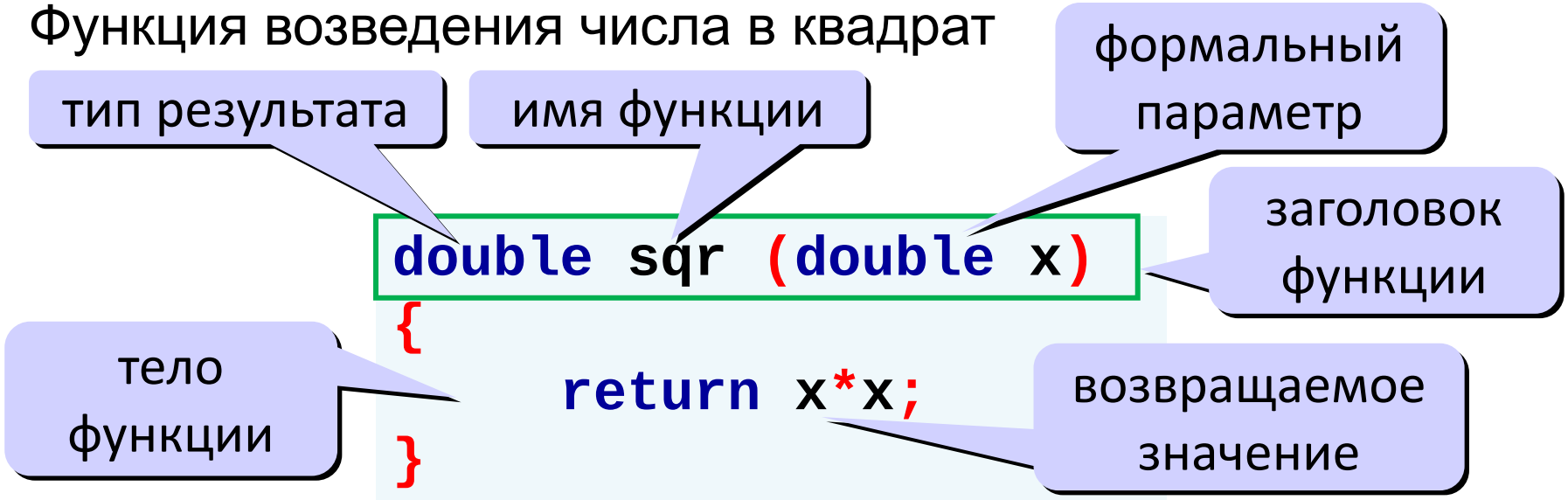


Если тип результата отличен от *void*, то в теле функции обязательно должна присутствовать инструкция ***return***, возвращающая результат в точку вызова функции.

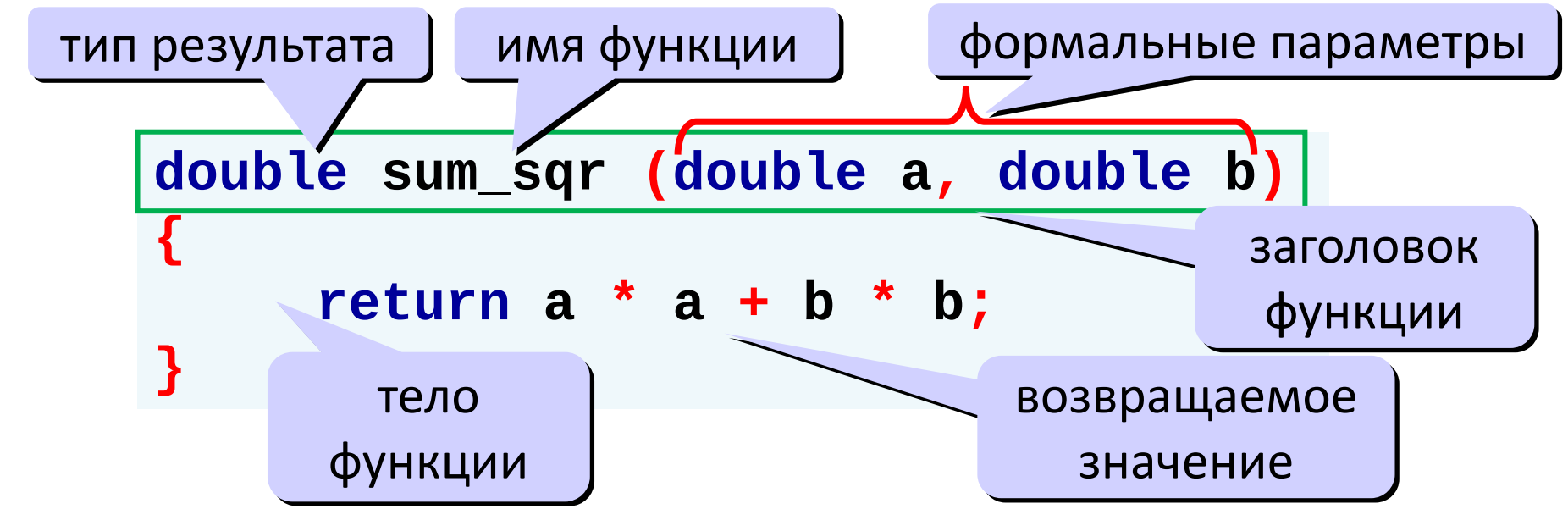
! Функция может быть определена в любом месте программы, но не в теле другой функции.

Определение функции

Функция возведения числа в квадрат



Функция вычисления суммы квадратов аргументов



Вызов функции

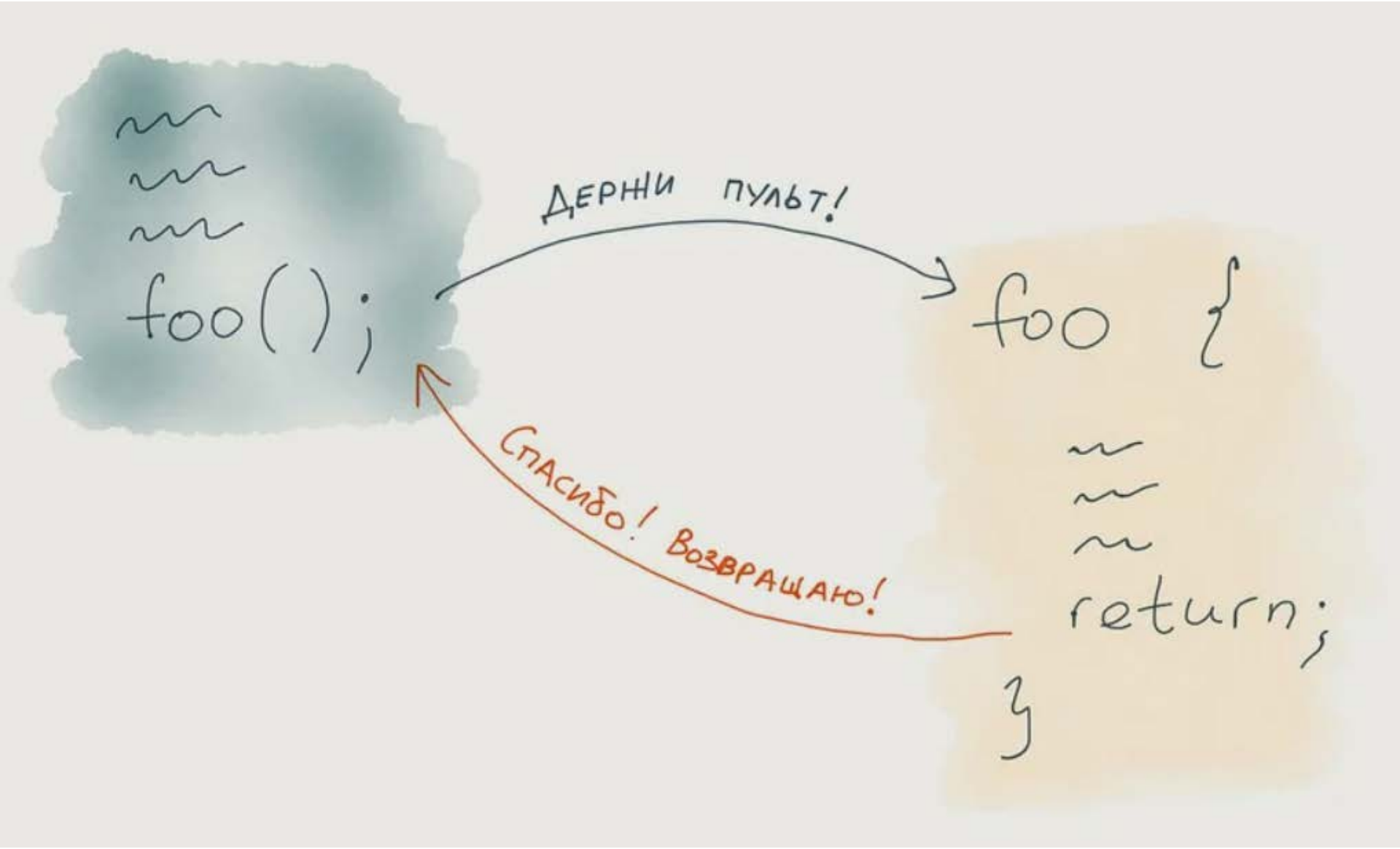
```
имя_функции (список_фактических_параметров)
```

входные данные

Особенности:

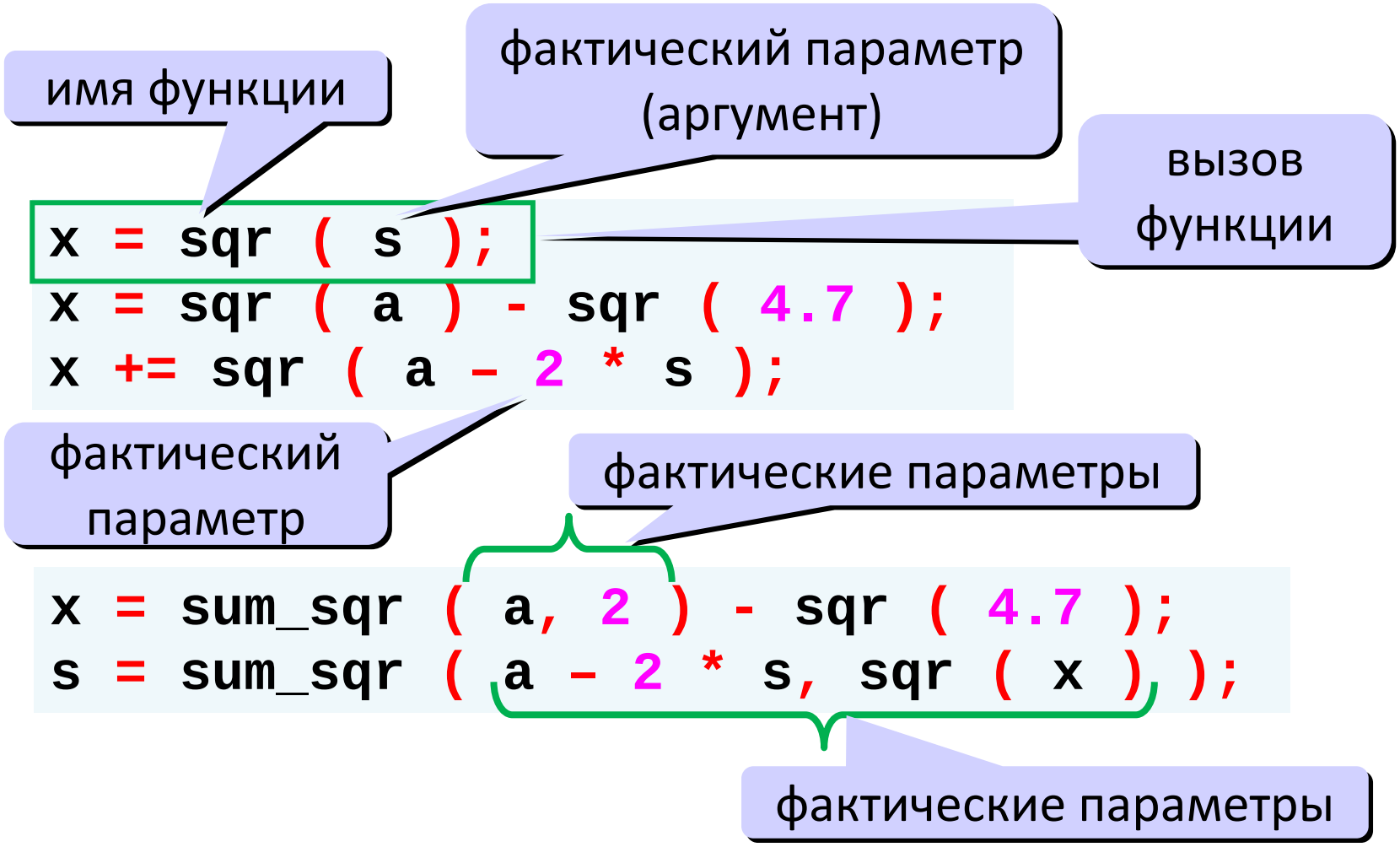
- вызов функции – это **выражение**, которое может являться операндом в более сложном выражении;
- независимо от того, возвращает ли функция значение, вызов функции может быть самостоятельной **инструкцией-выражением**;
- при вызове функции **управление передается** из вызывающей функции на первую инструкцию вызываемой функции;
- по команде ***return*** или при достижении конца тела функции **управление передается обратно** в точку вызова.

Вызов функции



Вызов функции

```
int a = 5;
double x, s = 1.13;
```



Передача параметров в функцию

В языках программирования существует **два способа** передачи параметров:

- **по значению** – функция работает с собственным экземпляром значений аргументов, любые изменения значений параметров внутри функции никак не отразятся на соответствующих значениях аргументов;
- **по ссылке** – и вызывающая, и вызываемая функции работают с одним и тем же блоком памяти.

В языке Си передача параметров в функцию осуществляется только ПО ЗНАЧЕНИЮ.

Передача параметров по значению

Особенности:

- формальные параметры – это локальные **переменные** функции, которые создаются при ее вызове и **инициализируются значениями фактических** параметров;
- все значения, передаваемые в функцию, являются для нее **входными**;
- аргументами функции могут являться любые **выражения**, тип значений которых совместим по присваиванию с типом соответствующего формального параметра.

Входные данные для вспомогательного алгоритма и входные данные для функции, реализующей этот алгоритм, не тождественны.

Критерии соответствия параметров

Фактические параметры должны соответствовать формальным по **трем критериям:**

- **по количеству** – количество аргументов, переданных функции при вызове, должно **совпадать** с количеством формальных параметров в заголовке функции;

проверяет компилятор

- **по типу** – тип фактического параметра должен быть **совместимым по присваиванию** с типом формального параметра (или совпадает, или возможно неявное приведение типа значения фактического параметра к типу формального);

проверяет компилятор

- **по смыслу**.

проверяет только **сам программист**

Соответствие формальных и фактических параметров

Функция вычисления разности аргументов

```
double difference (double a, double b)
{   return a - b; }
```

уменьшаемое

вычитаемое

формальные параметры

```
int a = 5;
double s = 1.13, *p = &s;
```

Вызов функции

```
x = difference ( s, 2 ); /* s – 2 */
x = difference ( s, a, 2 );
x = difference ( *p, p );
x = difference ( s );
```

корректный вызов, a=s, b=2

слишком много аргументов

несоответствие типа аргумента

слишком мало аргументов

Объявление функции

Любая функция должна быть или определена, или объявлена до ее вызова.

В программах на языке Си ***первой*** принято определять функцию ***main()***.

Объявление функции называется **прототипом** функции.

```
тип_результата имя_функции (спецификация_параметров);
```

сигнатура

; обязательна

Спецификация параметров – это список типов формальных параметров в порядке их объявления в заголовке функции.

Часть объявления функции, позволяющая идентифицировать функцию среди других, составляет **сигнатуру** функции.

Прототип функции может быть записан перед вызывающей функцией или в теле вызывающей функции.

Прототип функции

```
тип_результата имя_функции (спецификация_параметров);
```

прототип библиотечной функции

тип функции

спецификация параметров

```
double sin (double);
double sqr (double);
double sum_sqr (double, double);
double difference (double, double);
```

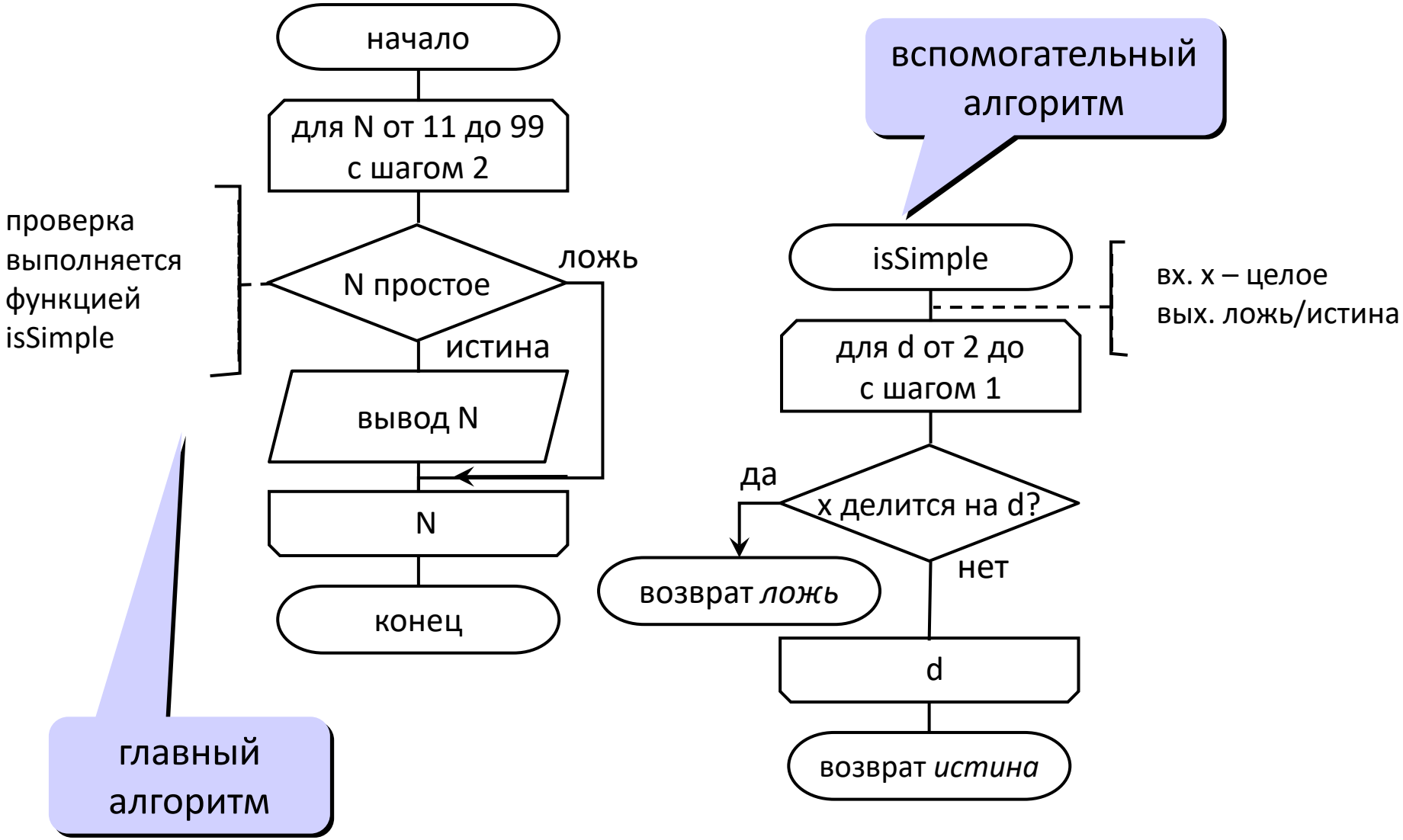
однотипные функции

однотипные функции

Функции, объявления которых отличаются только идентификаторами, являются **однотипными**

Пример программы с функцией

Задача. Вывести на экран все простые двузначные числа.



Пример программы с функцией

Задача. Вывести на экран все простые двузначные числа.

```
#include <stdio.h>
int isSimple (int); /* 1-простое, 0-составное */
int main()
{
    int num;
    for ( num = 11; num < 99; num += 2 )
        if ( isSimple (num) )
            printf ("%d  ", num);
    return 0;
}
int isSimple (int x)
{
    int d;
    for ( d = 2 ; d * d <= x ; d++ )
        if ( x % d == 0) return 0;
    return 1;
}
```

прототип функции

вызов функции

локальная переменная

определение функции

прерывание функции и возврат в точку вызова