# Chapter 14. Recursive functions. Types of recursions

## 14.1 The concept of recursion

Recursion is a programming technique in which an object can define itself or contain elements that reference it. An algorithm is considered recursive if it contains conditions for exiting the recursive function, known as a recursion basis, as well as a recursion step that should ultimately lead to the completion of the algorithm.

Recursive algorithms are not used if:

1. Limited memory: The function stores local variables and parameter values. With deep recursion, the memory footprint increases and an error occurs when the recursion stack overflows.
2. Infinite recursion: exit condition failed, unintentional indirect recursion (overloaded functions with the same functionality).
3. Recursion too deep.
4. Variables are too large.

## 14.2 Types of recursion

There are several types of recursion, which differ in the way they are used and their structure:

1. Simple (direct) recursion. The function calls itself directly. This type of recursion is the most common.
2. Complex (indirect) recursion. In this case, function A calls function B, and function B in turn calls function A. This can be useful in more complex structures where functions work on different aspects of the same task.
3. Tail recursion. Recursion in which the last step of a function is to call itself. This type of recursion can be optimized by the compiler to use fewer resources, thereby avoiding stack overflow.

For understanding, examples illustrating the use of recursion are considered.

Statement of problem 1: two natural numbers m and n are given. Implement a program that will find the greatest common divisor of the numbers m and n.

Algorithm for solving the problem:

1. Data input: The user enters the values ​​of n and m.
2. Checking numbers for naturalness: whether the numbers m and n are natural.
3. If both numbers are 0, an error message is printed to the console.
4. If one of the numbers m or n is 0, a non-zero number is printed.
5. Execution of the Euclidean algorithm, condition for exiting the loop m== n.
6. Any of the numbers m and n is output.

Figure 14.1 is a block diagram of a C++ program that demonstrates the process of finding the greatest common factor (GCD) between two natural numbers [2].

|  |
| --- |
|  |
| Figure 14.1 – Flowchart for solving the problem of finding the greatest common divisor |

Problem statement 2: Calculate approximately the sum of n terms of the series. The function arguments are n and x, where n is the number of terms of the series, x is a variable. The series is shown in figure 14.2.

|  |
| --- |
|  |
| Figure 14.2 – Example of a series for task 2 |

Solution algorithm

1. Data Entry: The user enters the values ​​of n and x. You need to make sure that n is a natural number, since the number of terms in a series cannot be negative or equal to zero.
2. Addition Loop: Uses a loop that will go from 1 to n. Inside the loop, the corresponding member of the series is calculated depending on the value of i and the variable x. This term is added to the total.
3. After the loop completes, the program prints the final sum of the n terms of the series.

The solution in the form of code in C++ is presented in figure 14.3.

|  |
| --- |
|  |
| Figure 14.3 – Flow diagram for solving the problem of approximate calculation of the sum of n terms of a series |

There are two approaches to writing recursive functions: prefix and postfix forms.

In the prefix form of function notation, the recursive call is made before the main actions of the function are performed. This means that the function first calls itself and then performs the necessary operations after that call. This approach can be used, for example, for preliminary calculation analysis before performing other operations.

In postfix form of function notation, the necessary actions are performed first, and then the recursive call is made. This allows the result of intermediate calculations to be processed before the function calls itself again.

Let's look at an example of a function that calculates the factorial of a number using tail recursion.

Statement of problem 3: given a natural number N. Implement a program that will print the numbers from N to 0.

Recursive function func(int num): If the argument num is greater than zero, the function prints the value of that argument and calls itself with the value decremented by one. This causes a sequence of numbers from N to one to be printed to the console. The solution in the form of C++ code is presented in Figure 14.4 [3].

|  |
| --- |
|  |
| Figure 14.4 – C++ code for task 3 |

## 14.3 Problem of finding the n-th Fibonacci number

Fibonacci numbers are a series of integers. Their peculiarity is that each element is the sum of the two previous numbers (except for the first and second number).

The Fibonacci sequence starts with 0 and 1: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Recursive function algorithm:

1. If n = 0, return 0.
2. If n = 1, return 1.
3. Otherwise, return the sum of the two previous numbers F(n-1) + F(n-2).

The implementation of finding the nth Fibonacci number in C++ is presented in the form of a block diagram in figure 14.5.

|  |
| --- |
|  |
| Figure 14.3.2 – Flowchart for finding Fibonacci numbers |

## 14.4 Complex recursion

Complex recursion is a process in which one function initiates a call to another function, which in turn calls the first function again. This mutual exchange of calls can continue until the recursion termination condition is reached.

A function signature, in turn, is part of the overall function declaration that helps the compiler or interpreter understand that the function exists and can be used in further calculations or manipulations. A function signature contains several key components:

1. Function name.
2. Number, type and order of parameters - this information indicates how many and what parameters the function expects, as well as in what order they should be passed.
3. Return value type - indicates what type of data the function will return after execution. This can be any type, including primitive types (eg integers, strings) or complex data structures (eg lists, objects).