Impact Factor: 9.9 ISSN-L: 2544-980X

Recursive Functions and the Basics of their Use

Abdullayeva Mohigul Rakhimjon kizi [1]

Annotation: The article provides information about data structure algorithms, algorithms and their properties, application in software. The article describes a function and a recursive function, its use and advantages over iterative methods, as well as types of recursion and methods for its application.

Keywords: Algorithm, properties, "laws", invariant recurrence relations, Compiler, list element, recursive function.

INTRODUCTION

The algorithms that we have considered so far can be called obvious; their behavior is speculatively observed on a figurative model; the programmer's task is to see its component parts and arrange them in the required sequence. The components of such algorithms are heterogeneous, and the process of their implementation can be tracked "historically". However, there are other approaches to design, besides figurative evidence. Let's talk about them in the most general words: - algorithms that consistently approach a given goal; - algorithms in which the search for a solution is reduced to the formulation of one or several problems of smaller dimension; - algorithms that comply with the relationships established for them (properties, "laws", invariants); Recursion is a definition, description, image of an object or process within this object or process itself, that is, a situation when an object is a part of itself. The term "recursion" is used in various specialized fields of knowledge - from linguistics to logic, but is most widely used in mathematics and computer science. 26 BEST JOURNAL OF INNOVATION IN SCIENCE, www.bjisrd.com RESEARCH AND DEVELOPMENT In programming, recursion is a call to a function (procedure) from itself, directly (simple recursion) or through other functions (complex or indirect recursion), for example, a function calls a function, and a function calls a function. The number of nested calls to a function or procedure is called the depth of recursion. A recursive program allows you to describe a repetitive or even potentially endless calculation, without explicitly repeating parts of the program or using loops. Structurally, a recursive function at the top level always represents a branch command (selecting one of two or more alternatives depending on the condition(s), which in this case is appropriately called the "recursion termination condition"), having two or more alternative branches, of which although at least one is recursive and at least one is terminal. A recursive branch is executed when the recursion termination condition is false and contains at least one recursive call - a direct or indirect call by the function to itself. A terminal branch is executed when the recursion termination condition is true; it returns some value without making a recursive call. A properly written recursive function should guarantee that, after a finite number of recursive calls, the recursion termination condition will be met, causing the chain of successive recursive calls to break and return. In addition to functions making one recursive call on each recursive branch, there are cases of "parallel recursion" where two or more recursive calls are made on the same recursive branch. Parallel recursion is typical when processing complex data structures such as trees. The simplest example of a parallel recursive function is the calculation of the Fibonacci series, where to obtain the value of the nth term it is necessary to calculate the (n-1)th and (n-2)th. The implementation of recursive function calls in practically used languages and programming environments, as a rule, relies on the call stack mechanism - the return address and local variables of the function are written to the stack, due to which each subsequent recursive call of this function uses its own set of local variables and therefore works correctly . The downside of this mechanism, which is

[1] TATU Fergana branch, Uzbekistan

quite simple in structure, is that each recursive call requires a certain amount of computer RAM, and if the recursion depth is too large, a call stack overflow may occur. The question of the desirability of using recursive functions in programming is ambiguous: on the one hand, recursive the form can be structurally simpler and more visual, especially when the implemented algorithm itself is essentially recursive. In addition, some declarative or purely functional languages (such as Prolog or Haskell) simply do not have syntactic means for looping, and recursion is the only available mechanism for organizing repeated calculations. On the other hand, it is generally recommended to avoid recursive programs that lead (or under some conditions may lead) to too great a depth of recursion. Thus, the example of recursive calculation of factorial, widely used in educational literature, is, rather, an example of how not to use recursion, since it leads to a fairly large depth of recursion and has an obvious implementation in the form of a regular cyclic algorithm. Accordingly, the technology for their development is also changing: the main task is to direct the algorithm “in the right direction”, make it move in the right direction, follow the “rules of the game”, and correctly reduce the problem to a similar one. Obviousness and observability here give way to logical consistency and evidence of compliance by the program with the specified properties. Examples of recursion can be found in mathematics. For example, recurrence relations determine (calculate) a certain element of a sequence through several previous ones. There are even more such examples in programming:

- the definition of any specific operator (conditional, loop, block) includes an arbitrary operator as its constituent parts;
- recursive data structure - a data structure element contains one or more pointers to a similar data structure.

For example, a singly linked list can be defined as a list element containing a NULL pointer or a pointer to a similar list; - recursive function - the body of the function contains a direct or indirect (through another function) own call. A recursive function can be infinite, like a loop. To avoid endless execution of a function, there are two principles that must be followed when programming recursive algorithms: The main criterion is that there must be at least one condition that would cause the function to terminate the recursive call. Progressive approach - recursive calls should evolve in such a way that each call gets closer to satisfying the base condition. Many programming languages implement recursion using stacks. Whenever a function calls another function (or itself), the calling function transfers control to the called structure. This transfer process may also involve passing data from the calling function to the called function. It follows that the calling function must suspend its execution, and resume it later when control is returned to it. The calling function must start exactly at the point in execution where it stopped. To continue working, it needs the same parameter values. Recursion can be difficult to understand, especially for those new to programming. Simply put, recursion is a function that calls itself. But let me try to explain with an example. Imagine that you are trying to open the door to your bedroom, but it is closed. Your three-year-old son appears around the corner and says that the only key is hidden in the box. You're late for work and you really need to get into the room and get your shirt. You open the box only to find... more boxes. Boxes within boxes and you don't know which one contains your key. You urgently need a shirt, so you need to come up with a good algorithm and find the key. Computational complexity is defined as the amount of additional memory required to execute a module. In the case of iterations, the compiler requires virtually no additional memory. The compiler continues to update the values of the variables used in iterations. But in the case of a recursive algorithm, the system must save an activation record every time a recursive call is made. It is believed that the computational complexity of a recursive function can be higher than that of a function with iteration.

LITERATURE

1. “Introduction to Algorithms” - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein**.** 2009 by The Massachusetts Institute of Technology.
2. “Data Structures and Algorithm Analysis in C” - Mark Allen Weiss. Algorithm Analysis: PREFACE 2022.