Computer Science

E-ISSN No : 2581-6195 | Volume : 1 | Issue : 5 | December 2018

RECURSIONS, A PROGRAMMING CONCEPTS THAT ALLOWS FUNCTIONS AND MODULES TO CALL ITSELF: DESIGN, IMPLEMENTATION AND ANALYSIS.

Amanze, B.C.[1], Onukwugha.C. G.[2]

ABSTRACT

The objective of this work is to look at the features of these languages that allow itself to be called. One want to look at a function ∞ on how it either calls itself directly or calls a function that in turn calls the origin function . This function is called recursive function or module. This work would look at its properties, implementation, analysis and complexity. The result of all these features would be based on chosen examples of recursion.

KEYWORDS: Function, Recursion, Modules, Analysis

1. INTRODUCTION

An object is said to be recursive if it particularly consists of or is defined in terms of itself. It also refers to several related concepts in computer science and mathematics, where one or more functions of an integer's variable are defined by giving initial values and by giving the value for larger integers in terms of smaller ones. In recursion, a function ∞ either calls itself directly or calls a function that in turn calls the original function ∞ , then the function ∞ is called a recursive function. The power of recursion lies evidently in the possibility of defining an infinite set of objects by a finite statement. In the same manner, an infinite number of computations can be described by a finite recursive program, even if this program contains no explicit repetitions. Recursive algorithms however are primarily appropriate when the problem to be solved or the functions to be computed, or data structure to be processed are already defined in recursive terms.

Recursive algorithms are particularly appropriate when the underlying problem or the data to be treated are defined in recursive terms. This does not mean, however, that small recursive definitions guarantee that a recursive algorithm is the best way to solve the problem. In fact, the explanation of the concept of recursive algorithm by such inappropriate examples has been a chief cause of creating widespread apprehension and antipathy towards the use of recursion in programming, and of equating recursion with inefficiency. This happens in addition to the fact that the widespread programming language such as FORTRAN forbids the recursive use of subroutines, thus preventing the invention of recursive solutions even when they are appropriate.

Recursion compared to iteration

Recursion is an alternation form of program control. It is essentially repetition with a loops are used. When a loop body could be specified. The repetition of the loop body is controlled by the loop control structure.

In recursion, the method itself is called repeatedly. A selection statement must be used to control whether to call the method recursively or not. Recursion bears substantial overhead each time. The program calls a method; the system must assign space for all the methods local variables and parameters. This can consume considerable memory and requires extra time to manage the additional space. One of the bad aspects of recursion is that it requires too much memory space and takes much time to compute a solution.

Tower of Hanoi an example of recursion

This is a mathematical puzzle which consists of three towers (pegs) and more.

Properties

A recursive function can go infinite like a loop. To avoid infinite running of recursive function, there are two properties that a recursive function must have:-

a: Base criteria, which means there must be at least one criteria or condition, such that, when this condition is met the function stops calling itself recursively.

b: Progressive approach; here the recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base criteria.

[1]Department of Computer Science, Faculty of Science, Imo State University, Owerri

[2]Department of computer Science, School of Science Federal University of Technology, Owerri

HOW TO CITE THIS ARTICLE:

Amanze, B.C., Onukwugha .C. G.(2018). Recursions, a Programming Concepts that Allows Functions and Modules to Call itself: Design, Implementation and Analysis. , International Educational Journal of Science and Engineering (IEJSE), Vol: 1, Issue: 5, 01-03

Research Paper

Recursive Relations

a. The Fibonacci sequence is given by the equations.

$f_o = 1$

$f_1 = 1$

$f_{n+1} = f_n + f_n - 1$

b. When differential equations are to be solved numerically, recursion relations such as;

$$F(x_0 + n1_3) = F\big(f(x_0 + (n-1)h)\big),$$
$$F(x_0 + (n-2)h), \ldots = f(x_0 + (n-1)h))$$

Arise where fa vector of real numbers is in general.

c. When linear differential equations are solved by series, recursion relations for the coefficients of the powers of the independent variables arise.

2. LITERATURE REVIEW

2.1 Body of Knowledge

The systematic study of recursion began in the 1920's when mathematical logic began to treat questions of definability, computability, and decidability. An important role is played by primitive recursive functions. Primitive recursive functions are integer functions of integers built up from addition and multiplication of integers, and previously defined primitive recursive functions by the primitive recursion scheme.

$$f(0, x_2, \ldots, x_k) = g(x_2, \ldots x_k)$$
$$f(x_1 + 1, x_2, \ldots x_k) = h(f(x_1, \ldots, x_k)\, x_1, \ldots x_k)$$

Here, g and h are primitive recursive functions of k=1 and k+1 arguments, respectively. As an example, we define n!, where n is positive integer, by n! = f (n) where f (0) = 1 and f (n+1) = (n+1). f (n). So in this case g is a function of 0 arguments, namely the constant 1, and h (u,v) = (v+1) u.

All the common functions of number theory are primitive recursive. Moreover, many important function countable domains other than the integers correspond to primitive recursive functions when one chose a specific enumeration for the domain.

Primitive recursive functions are included in general recursive functions. The definition of general recursive function is like that given above for primitive recursive functions, except that the relations are replaced by an arbitrary finite collection of equations relating the values f for different arguments, and the function is considered defined if and only if a unique value of $f(x_1, \ldots x_k)$ can be deduced from the equation for each k-tuplet $(x_1, \ldots k_k)$. Naturally, if someone gives you an arbitrary collection of such relations, you may not be able to determine whether you have a general recursive function. This difficulty is unavoidable. There is no way to give a definition scheme that is always guaranteed to give a function but which will give all computable functions. This fact is itself expressed in the terminology of recursive function theory by the statement that the set of computable functions is recursively enumerable but not recursive. The famous example of a general recursive function that is primitive recursive is the Ackermann functions defined by the equations.

A (0, n,p) = n + p, A (1,0,p) = 0
A (m+z, 0, p) = 1
and
A (m + 1, n + 1, p) = A (m + 1, n, p), P)

2.2 Related Literature

[1] Has some discussion of the implementation of recursion in LISP, and [2] discussed the implementation of recursion in Algol. [3] discussed when to use recursion and when to use iteration. [4] Has a thorough treatment of subclasses of general recursive functions. The standard reference on recursive function theory was written by [5], who gave a more elementary treatment in a later book (1967). Two aspects of recursion are current research topics in Computer Science now unlike before various programming languages favor recursion in solving problems especially some object oriented languages. That is the notion of recursive program being extended in various ways, and methods implementing these extensions by compilers and interpreters [6]. Also, the formal properties of recursive programs are still a subject of study in mathematical theory of computation.

3. METHODS AND PROCEDURES

Many programming languages implement recursion by meaning of stacks. Generally, whenever a function (caller) calls another function (caller) or itself as caller, the caller function transfers execution control to the caller. This transfer process may also involve some data to be passed from the caller to the caller. This implies, the caller function has to suspend its execution temporarily and resume later when the execution control returns from the caller function. Here, the caller function needs to start exactly from the point of execution where it puts itself on hold. It also needs the exact same data values it was working on. For this purposes, an activation record (or stack frame), created for the caller function. This activation record keeps the information about local variables, formal parameters, return address and all information passed to the caller function.

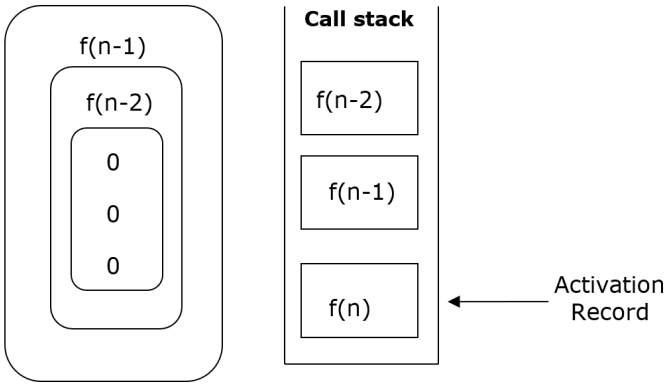

Figure 1: Recursive function

Recursive Procedures

In programming, it is frequently convenient to have a procedure use itself as a sub- procedure. If the procedure does this, it is called recursive. Recursive procedures are particularly natural

in dealing with symbolic expression because the structure of the programs often matches the structure of data. As far as programming languages are concerned, recursive procedures are quite natural, it requires a special statement in the definition of the language to forbid them.

However, implementing them requires that a special kind of object code be compiled, and early programming languages like FORTRAN do not allow them. The problem is the variables in the program correspond to location in the machines and when the program is called by itself; it will use the same locations, overwriting their previous contents. Therefore, recursive programs use a data structure called a stack; to store the contents of registers that must be saved. This storage can be done by the calling routine before it enters the sub-routine before it uses the registers, the latter being more common.

After the registers have been saved on the stack, the index into the stack is increased by the number of registers stored, so that subsequent saving on the stack will use fresh registers. When the subroutine exists, the contents of the saved registers are restored from the stack to their previous values, and the stack pointer is reduced by the amount it was previously increased. This is done by the caller or by the subroutine, according to whether the caller or subroutine did the original storing. An alternative technique of to use the stack for all temporary registers.

Implementation and Analysis

There are examples to show the application of recursion or recursive programs. The attractive graphic pattern shown below consists of the super position of five curves. These curves follow a regular pattern and suggest that they might be drawn by a plotter under control of a computer. The goal here is to discover the recursion schema, according to which plotting program might be constructed. Inspection reveals that three of the super imposed curves have the shapes shown below fig. 2, one shall denote them by H_1, H_2 and H_3. The figures show that $H_i + 1$ is obtained by the composition of four instances of Hi of half size and appropriate rotation and by typing together the four His by three connecting lines. Notice that Hi may be considered as consisting of four instances of an empty H0 connected by three straight lines. Hi is called the Hibert curve of order i after its inventor.

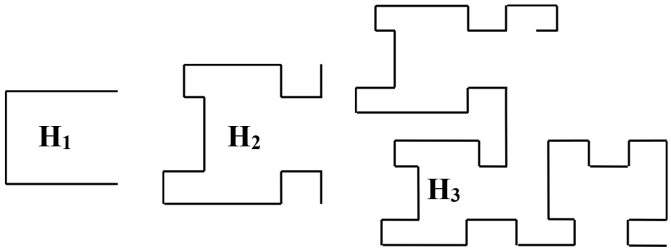

Figure 2: Composition of three super curves

4. CONCLUSION

This work has shown us the programming concepts of the allowing modules, functions as well as procedures and subroutines to call themselves. Originally, many programmers did not see the usefulness of using recursive programs because of their sluggishness and what have you. Many have argued that iteration could accomplish the same task recursion does but there is still a difference between them. One reason is that recursion makes a program more readable and its efficiency could be more reliable than iteration. On the other hand, we can use the number of iterations to count the time complexity. One of the disadvantages of recursion is the need for large memory capacity, but this is readily available as one has many storage systems that one can back up to, for inexhaustibly memory space. Also, the technology of expanded and extended and extended memory management system is an added advantages. It should be clear that most object oriented programming languages are able to support recursion, with the use of functions.

5. REFERENCES

1. McCarthy, J. (2002). “Actions and other events in situation calculus”. Proceedings KR-2002, 615 -628
2. B. Randell and C.J. Russel, the implementations of Algol 60, Academic Press, 1964
3. Wirth, N. (1976). Algorithm + Data Structures = Programs. Englewood’s Cliff, N.J. Prentice-Hall.
4. Peter, R. (1967). “Recursive functions”, Academic Press New York.
5. Kleene, S.C. (1967). “Mathematical Logic”, Wiley and Sons Press, New York.
6. Daniel Y. Liang (2011). Introduction Java Programming, Pearson Books N.J. U.S.A.
7. Deitel, P.J., & Deitel, H.M. (2009). Internet and World wide web-How to program, Pearson books N.J., U.S.A.
8. Deordney, A.K. (1993). The Tuning Omnibus, W.H. Freeman books New York, 2nd edition.
9. Manna, Z. (2000). Mathematic theory of computation Mc Grawhill, New York.
10. Mc Vite, D.G. & Wilson, L.B. (1971). The stable marriage problem, comm. A.C.M 14, N0. 7, 486-92.
11. Roberts, E.S. (1986). Thinking recursively, John Wiley and Sons New York.