IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Vol.-7, Issue-7, July 2019 E-ISSN: 2347-2693

Seven Fundamental Principles in the Effectuation of Recursion

Rishi Saxena

Department of Computer Science, Sophia Girls' College, Ajmer (Autonomous), India

Corresponding Author: rishi.522.in@gmail.com

DOI: https://doi.org/10.26438/ijcse/v7i7.6367 | Available online at: www.ijcseonline.org

Accepted: 13/Jul/2019, Published: 31/Jul/2019

***Abstract*:** Recursion is a programming methodology used to write efficient computer programs. A recursion based solution to a given programming problem can be translated to an iteration based solution or by using some other programming constructs which do not accord with the methodology of recursion. Computer programmers and software engineers usually face this kind of confusion that which methodology to implement in a particular case or to some specific problem class as there are no candid guidelines regarding the implementation of recursive solution, leaving the developers in a state of taking decisions upon their choices and preferences. This paper explores the issue of recursion in various aspects of developing a computer program and discovers the seven fundamental principles behind the use and implementation of recursion in generating a solution for a problem of specific nature and class. These seven principles can be referred as pillars of foundation on which recursion is employed by many programmers as a tool to generate an effective solution which adheres to the quality standards of software engineering.

***Keywords*:** Recursion, Principles, Programming, Functions, Methods.

I. INTRODUCTION

The classification of any programming language, on the basis of distinct features it possesses, is referred as Programming paradigm. Recursion comes under the type of programming paradigm called as "Structured Programming", where the main objectives are (a) To improve the clarity of the source code, (b) Quality of the produced computer program and (c) Time required to develop a solution for the given problem. These objectives are achieved by exploring the functionality of branching, looping, blocks and methods.

Divide & Conquer is basically a design paradigm for algorithms. It operates by decomposing the main problem into sub problem of the same type recursively until the problem becomes so small that it can be solved directly without the requirement of further break down. And then all the solutions to these smaller problems are integrated to produce the solution for the main problem.

The paper is organized as follows, Section I introduces various aspects of the paper – Programming paradigm, Structured programming, Section II defines the concept of Recursion in real world as well as in the field of Computer Science, Section III gives the first principle in implementing recursion by addressing the logical deduction of the solution, Section IV gives the second principle in implementing recursion by taking into account the lines of code required to generate the solution, Section V gives the third principle in implementing recursion by considering operations performed over Tree data structure, Section VI gives the fourth principle in implementing recursion by following the control flow of the program in dynamic environment, Section VII gives the fifth principle in implementing recursion by observing the simulated behavior, Section VIII gives the sixth principle in implementing recursion by considering the aspects of process communication and data sharing, Section IX gives the seventh principle in implementing recursion by taking into account the issue of stack overflow. Section X concludes the research study and specifies the future scope of the study.

II. RECURSION [1]

When something is defined in its own term or type of its term recursion happens. For example to develop an object 'A' if you require an amount of object 'A' then recursion takes place. It is basically a process when some procedure passes through one of its own steps that involve arousal of the same procedure. Such a procedure is referred as "recursive" as it is implementing the methodology of recursion to produce the solution.

Mathematicians define recursion as an expression containing terms where generation of every term is done by repeating

some mathematical operation. Consider the mathematical sequence given by the relation a(n) = 2 * a(n-1) + 1, with a(0) = 2.

a(1) = 2 * a(0) + 2 = 6
a(2) = 2 * a(1) + 2 = 14
a(3) = 2 * a(2) + 2 = 30
now if query comes that what is a(10)?
Then to find the answer you must know a(9), and to find the answer of a(9) you must know a(8), and so on until you reach a(0).

In the field of computer science recursion is defined as creating a function call to same function within the function.

```
returnType recursiveFunction (Parameter(s)){
        if baseCase then
                return someValue
        else
                return recursiveFunction(argument(s))
}
```

Any computerized recursion is based on two properties:
1. Inclusion of a base case that is the state when no further recursive calls are made and an answer is produced and returned.
2. Defining a protocol which directs all other recursive scenarios towards the base case.

It is a methodology in which some bigger problem is solved by decomposing it into smaller problems which are basically instances of the bigger problem only and producing solution to each smaller problem.

Recursion types:
1. Single recursion – when only one self call is created.
2. Multiple recursion – when multiple self calls are created.
3. Direct recursion – when self call is placed inside the same method definition.
4. Indirect recursion – when a call is created in some other method which is called by this method. For example A->B, B->C, C->D, D->A
5. Anonymous recursion – when a call is made by context and not by method name.
6. Structured recursion – when data for recursive call is produced by decomposing the arguments.
7. Generative recursion - when data for recursive call is produced by creating new data from the arguments.

III. PRINCIPLE #1 OF ORGANIC SOLUTION

A program can be written in many ways to obtain the solution. Good programming demands to achieve the solution by maintaining its logical deduction steps to reflect in the algorithm implementation.

For example to print a natural number series from 1 to 100, a programmer can opt for by using the print statement for 100 times, and a programmer can opt for by using a loop construct, in either implementation the output to the end user will not change but the logic lies in the latter implementation as the natural solution of the problem is repetitive in nature.

For example Fibonacci series [2], which is a sequence of numbers, begins with two seed values 0 and 1, and then any following value is the addition of the two previously established values.

The formal mathematical definition is $F_0=0$, $F_1=1$, and $F_N = F_{N-1} + F_{N-2}$, for all $N > 1$
If we write in terms of a series then first two terms a0 = 0 and a1 = 1 are the seed values, the following terms are a2 = 1, a3 = 2, a4 = 3, a5 = 5, a6 = 8, and so on.

Query: In Fibonacci series find the value of a5?
Iterative Solution: a0 and a1 are given, so begin calculation from a2 by adding the values of a0 and a1, then calculate the value of a3 by adding the values of a2 and a1, then calculate the value of a4 by adding the values of a3 and a2, and then calculate and return as answer the value of a5 by adding the values of a4 and a3.
Recursive Solution: Return as answer the addition of a4 and a3.

The answer to the question is given as now the question bifurcates into two new questions that is to find the value of a4 and a3. And again the same reply that the value of a4 is the addition of a3 and a2, and the value of a3 is the addition of a2 and a1, and it goes on until seed values are reached. Every time it is a one line solution and in complete accordance with the mathematical definition. Therefore for certain problems writing recursive implementation is the natural way of logical deductions as opposed to the optimal solution which may give better time complexity but lacks in the understanding of solving a problem which is fundamental to analogical learning.

IV. PRINCIPLE #2 OF SLOC DIMINUTION

Acronym SLOC [3] stands for Source Lines of Code, it is also referred as Lines of code, is basically a standard of measure that quantifies the physical magnitude of a computer program by counting the source code' text lines.

It is generally used to (a) auspicate required programming effort amount in order to develop a computer program, (b) Programming productivity estimation and (c) Programming maintainability estimation once the program has become operational. A computer program with lesser SLOC requires (a) less effort to develop thus resulting in an easier and simpler solution, (b) less time to produce the solution and (c) easier to maintain.

Producing a solution based on recursion results in much less source lines of codes as compared to any optimal solution. For example, to develop a solution for the classical problem of "Tower of Hanoi", the non-recursive iterative solution and recursive solution are developed. Following are the results for testing against SLOC:

Iterative Solution:

Symbol	Count	Definition
Source Files	1	Source Files
Directories	1	Directories
LOC	155	Lines of Code
BLOC	23	Blank Lines of Code
SLOC-P	108	Physical Executable Lines of Code
SLOC-L	61	Logical Executable Lines of Code
MVG	13	McCabe VG Complexity
C&SLOC	0	Code and Comment Lines of Code
CLOC	24	Comment Only Lines of Code
CWORD	150	Commentary Words
HCLOC	1	Header Comment Lines of Code
HCWORD	7	Header Commentary Words

Recursive Solution:

Symbol	Count	Definition
Source Files	1	Source Files
Directories	1	Directories
LOC	28	Lines of Code
BLOC	4	Blank Lines of Code
SLOC-P	23	Physical Executable Lines of Code
SLOC-L	14	Logical Executable Lines of Code
MVG	1	McCabe VG Complexity
C&SLOC	3	Code and Comment Lines of Code
CLOC	1	Comment Only Lines of Code
CWORD	19	Commentary Words
HCLOC	1	Header Comment Lines of Code
HCWORD	5	Header Commentary Words

The comparative results of SLOC-P, which is an acronym for Physical executable source lines of code that doesn't include the lines for comments and the blank lines, shows a significant reduction of 78.70 % (108 in Iterative solution has been reduced to 23 in Recursive solution). Therefore, recursive solution ensues in lesser programming effort, increases the productivity by minifying the time required and improving the maintenance of the computer program after being operational.

V. PRINCIPLE #3 OF RCC GRAPH OPERATIONS

RCC, Rooted, Connected and Cycle-free graphs are referred as Trees [4] with the following properties (a) One distinguishable node is designated as "Root", (b) Throughout the data structure one and only one connection is allowed between any pair of nodes by using a directed edge only, (c) the direction of the edge is always from parent node to child node only.

Tree data structure has implementations in several vital areas of computer science, one of the most popular implementation is the "File System". Any implementation of Tree data structure calls for abiding to the parent-child relationship. This relation is the fundament of Tree data structure, because by maintain this relationship one can ensure a graph to be cycle-free, and the same rationale goes for all the operations performed over any Tree data structure. For example "Traversal of Tree" operation can be achieved by DFS, BFS, In-order, Pre-order, Post-order, etc. all the above algorithms can be implemented by looping as well as with recursion. In iteration, to make the operation aligned with the centre concept of Tree, i.e. parent-child relationship, produces an additional overhead whereas in recursion parent-child relationship is maintained inherently. Every recursive call always returns to the caller method thus making the caller method as parent and the called method as child.

Therefore by making a recursive solution for Rooted, Connected and Cycle-free graphs a programmer guarantees the implementation is in complete accordance with the basis concept on which Trees are built.

VI. PRINCIPLE #4 OF DYNAMIC CONTROL FLOW

In a computer program the order of executable statements and function calls is referred as Control flow. Dynamic control flow implies that the decision of the order of execution can be taken at runtime.

A Programming language which is dynamic in nature is basically a high-level language which has the ability to execute common programming behaviors like new code addition, object extension, changes in type system, etc at runtime.

By Dynamic method dispatch [5] Java programming language implements run-time polymorphism. A method call to a method which is overridden is resolved at run time rather than compile time by using this mechanism. By definition recursion can appear in dynamic control flow as compared to iteration which is designed for fixed control flow and translation of dynamic control flow nature of recursion is not gently possible in iteration.

Therefore, working in dynamic programming environment which is implied in current era programming, the solution generated by recursion will be more feasible as compared to any other optimal solution.

VII. PRINCIPLE #5 OF SIMULATION

Simulation, by definition is an act of behavior imitation of some process by means of something befittingly analogous. It is also referred as an act of giving a false appearance, and that is what exactly happens in some cases when a programmer employs iteration in lieu recursion. For example in State space search [6] problems, A state-space representation is a 4-tuple ⟨Q, q0, F, O⟩, where:
Q: is the set of states, $Q \neq \emptyset$,
$q0 \in Q$: is the initial state,
$F \subseteq Q$: is the set of goal states,
O: is the set of the operators, $O \neq \emptyset$
If we create a graph referred as the State-Space Graph, then the vertices/nodes of such a graph are the states. An edge between two vertices exists if and only if one state is directly accessible from another state. We label the edges with the operator. A solution of a problem is a path that leads from a vertex (initial vertex) to some vertex (goal vertex). Precisely, the solution is the sequence of labels (operators) of the edges that formulate this path.
Algorithm for solution using Recursion:
[Input root node as Q0, Solution node(s) as F]
1. Begin
2. If $Q0 \in F$, return Q0
3. Else, if generate-child-nodes (Q0) $\neq \emptyset$ then
4. Recursive call (child-node)
5. Else return not-found
6. End.

Algorithm for solution using Iteration:
[Input root node as Q0, Solution node(s) as F]
1. Begin.
2. Declare Found ← False [Flag variable]
3. While Found ≠ True repeat the following
4. If $Q0 \in F$, exit while
5. Else, if generate-child-nodes (Q0) $\neq \emptyset$ then push in stack [Q0]
6. If next-child $\in$ F, Found ← True, Traverse the stack by pop operation
8. Else exit while.
9. Repeat from Step 5 with next-child.
10. End.

If we analyze the Iterative algorithm as solution then it is nothing but a simulation of the Recursive algorithm for solution. Iterative algorithm producing a imitation of Recursive algorithm by explicitly performing push and pop operations on stack, that is stacking and un-stacking. These stack operations are performed inherently in Recursive solution. Besides this, by performing Mccabe's cyclomatic complexity analysis it can be very well seen that the complexity of the Iterative solution is higher as compared to the Recursive solution. Thus, if for generating a solution, an algorithm has to simulate the behavior of some other algorithm then it would always be a better choice to prefer the original one.

VIII. PRINCIPLE #6 OF IPC

IPC, Inter-process communication, is a mechanism by which processes can share and manage data between them.
Typically implemented as client-server model, where some processes act as service providers and others as service consumers. In distributed computing often processes do both the roles.

In object oriented programming Inter Process Communication model is employed for message passing, a way by which objects can communicate with each other. This sort of communication (two-way) requires values to be passed to the receiver and returned to the sender for an efficient messaging set up. Recursion, which is built on the foundation of method mechanism, supports the message passing model inherently, whereas any other solution like with iteration the implementation of this message passing model is inconceivable as loops are executable statements and they don't accept or return values according to their definition. Therefore, solutions targeted for messaging passing behavior should consider recursion, as it facilitates the proper implementation of the inter process communication model.

IX. PRINCIPLE #7 OF STATE STORAGE

[Addressing the issue of Stack overflow]
Call stack or Execution stack is a data structure of stack type that stores the data and information about the running methods of a computer program.
When a computer program attempts to consume more memory than the available memory in a call stack then the stack point goes beyond the upper bound of the call stack resulting in a runtime error situation called as "Stack overflow" which may ultimately lead to program crash.

Because recursion uses call stack for the storage of parent method's states, there is always a risk of stack overflow.
Corresponding Iterative solution to recursion also needs to store the intermediate states by creating their own stack or some other mechanism. In user defined stacks also there exists the problem of overflow which the iterative solution needs to manage. Recursion fails only when stack overflow happens, so like iterative solution, if the programmer takes into account the memory management recursion will become error prone.

Therefore switching to iterative or any other mechanism in place of recursion just to avoid stack overflow is not a wise move as handling stack overflow is much easier as compared to developing any other non recursive solution.

X. CONCLUSION

The current era is an age of Information Technology. Everyday human life goes one step closer to a world driven entirely by computer systems. To match this demand of high growth of technology, software sector is working under tremendous pressure. Several programming paradigms are developed and are developing to ease the job of programmers and yet maintaining the quality of service. By gaining better understanding of recursion a programmer can deliver an efficient high quality solution within the deadlines. This paper has successfully established the fundamental principles which can be referred anytime in decision making process of implementing a recursive solution. The study done in the above paper is limited to the current programming frameworks only and provides the future scope of discovering new principles in the developing programming paradigms which are yet to be released.

REFERENCES

[1] Ronald L. Graham, Donald E. Knuth, Oren Patashnik, "Concrete mathematics", Addison-Wesley Publishing Company, United States of America, pp. 1-20, 1990, ISBN-13: 978-0201558029.
[2] John Hudson Tiner, "Exploring the World of Mathematics: From Ancient Record Keeping to the Latest Advances in Computers", New Leaf Publishing Group, United States of America, pp. 81-83, 2004, ISBN 978-1-61458-155-0.
[3] Nguyen, V., Deeds-Rubin, S., Tan, T., & Boehm, B. (2007, October). A SLOC counting standard. In Cocomo ii forum (Vol. 2007, pp. 1-16). Citeseer.
[4] Seymore Lipschutz and Marc Lipson, "Schaum's Outline of Discrete Mathematics", McGraw-Hill Education, United States of America, pp. 164-263, 2007, ISBN-13: 978-1259062537
[5] H. Schildt, "Java The Complete Reference Ninth Edition", McGraw-Hill Education, United States of America, pp. 178-179, 2014, ISBN: 978-0-07-180856-9
[6] E. Rich, K Knight, S. B. Nair, "Artificial Intelligence", McGraw-Hill Education, United States of America, pp. 25-30, 2009, ISBN: 978-0-07-008770-5
[7] U. Ray, T.K. Hazra, U.K. Ray, "Matrix Multiplication using Strassen's Algorithm on CPU & GPU", International Journal of Computer Sciences and Engineering, Vol.4, Issue.10, pp.98-105, 2016.
[8] N. Karthikeyan, "Shortest Route Algorithm Using Dynamic Programming by Forward Recursion", International Journal of Computer Sciences and Engineering, Vol.2, Issue.2, pp.44-48, 2014.

Authors Profile

Mr. Rishi Saxena is working as an Asst. Professor in the Department of Computer Science, Sophia Girls' College, Ajmer (Raj). He has PG & UG teaching experience of 9 years. His qualifications are BSc, Diploma in Multimedia, PGDCA, MSc CS, MCA. He has done Google certification in digital marketing and NPTEL certification in Java Programming. He has cleared NET-2015 & SET-2013 examinations in Computer Science. He has written two research papers in International journals and participated in fifteen National and International conferences/seminars/workshops/FDPs. His areas of interest are AI, Programming and Android App Development..

