https://dx.doi.org/10.4314/dujopas.v9i1b.20
ISSN (Print): 2476-8316
ISSN (Online): 2635-3490
Dutse Journal of Pure and Applied Sciences (DUJOPAS), *Vol. 9 No. 1b March 2023*

Evaluation, Computation and Coding of Iterative Function Using Recursive Approach

G. A. Otu[1*], M.S. Oyebanji[1], F. I. Okonkwo[2], R.U. Ugbe[4], A. C. Okafor[1], S. A. Usman[3] & O. A Ubadike[1]

[1.] Department of Computer Science
Air Force Institute of Technology Kaduna,
Nigeria

[2.] Department of Information and Communication Engineering
Air Force Institute of Technology Kaduna,
Nigeria

[3.] Department of Cyber Security
Air Force Institute of Technology Kaduna,
Nigeria

[4.] Department of Physics
Nigerian Defence Academy Kaduna,
Nigeria

Email: godwinotu@afit.edu.ng

Abstract
Iteration and recursion are very pivotal concepts in understanding the logic and building blocks of all computer programs across all programming paradigms. Although the theory of iteration together with the development and implementation of iterative algorithm is easy to grasp, that of recursion remains elusive to programmers especially novice programmers. In this research, functions composition is applied in the explanation of iteration using recursion. The method demonstrates an easy and elaborate way of writing iterative programs using recursion by identifying the significant variables in both constructs. Function composition is used to write the recursive function, the recursive variable is identified as a variable that converges towards the base case, also the base case is also identified as being the terminating point of the function call else the function call runs and fill the stack memory causing stack overflow. The recursive part is the loop update and base case the termination condition in iterative programs. The results obtained simplifies how to write iterative codes using recursion.

Keywords: Base case, Composition, Iteration, Recursion, Recursive case

INTRODUCTION
Iteration and recursion are fundamental concepts in computer science and understanding them is important for students in foundation programming courses. The various approaches used to teach and comprehend iteration and recursion has been studied widely by many computer science education research groups. However, students still struggle with both concepts and instructors still find it difficult to teach these theories effectively. Endres *et al.*(2021) surveyed that computer science educators identified recursion and loops as two of the most challenging areas to teach. While a recursive algorithm is one that calls itself to solve

**Author for Correspondence*

a given problem, an iterative algorithm uses a repeated set of instructions in order to solve the same problem.(Darsh, 2019).

Many researches and authors have elegantly taught and discussed iteration and recursion but understanding both concepts still pose a problem to amateur programmers. (Liu, et al., 2019). Describes that transforming recursion to iteration eliminates the use of stack frames during program execution. This eliminates the space consumed by the stack frame as well as the time overhead of allocating and de-allocating the frames, yielding significant performance improvement in both time and space. According to Darsh (2019), outlined that recursion and iteration algorithms are used as a base model or method for many other algorithms. Recursive algorithms tend to be more elegant when an algorithm is much bigger. This means that the code is smaller as opposed to an iterative algorithm, making it much easier when debugging a program. Recursion is particularly useful and favoured when searching through specific types of data structures like trees whereas iterative algorithms tend to search through data structures like arrays. Iterative algorithms are generally favoured over recursive algorithms as they tend to execute faster, and do not require extra memory. Recursion stores all its function calls in a stack which means that more memory needs to be allocated, making recursion a much slower algorithm to use. These existing researches have mostly studied the degree of understanding both concepts but have failed to point the relationship between both theories in terms of how solutions to problems can be built with respect to each method. In this research function composition will be treated elaborately so that computer programmers or algorithm developers will understand program logic which is the requisite skill for anyone venturing into programming. Also make computer researchers to understand that proper understanding of functions and function composition is the foundation of the greatest pillar of programming which is modular programming. Mastering modular programming technique leads to the development of bug-free codes because of incremental design. These are many reasons are so vital to really understand this important concepts.

Many researches have been conducted to bring sheath light on recursion and iteration. While some have explained recursion through iteration and other have talked about speed of recursive and iteration algorithms, some space complexity analysis of the algorithms of these two concepts, others have worked on the choice of implementation of various tasks based on these concepts. In this we are learning how to program iteratively using recursion with function composition method. Let us take a look at the existing body of research on these concepts.

Sharma and Vishwas (2022) explained how easy it is to transform an iterative algorithm to a recursive type but, as evident, it usually requires slightly more effort to convert an algorithm, which is recursive by nature, into an equivalent iterative algorithm. They tried to develop a technique to convert recursive algorithms to iterative ones. Although this method might not be completely generalized, but it can be considered as being applicable in a large class of algorithms. The procedure was demonstrated on some well-known recursive algorithms such tower of Hanoi, quicksort, binary tree traversals and more. Rinderknecht (2014) opined that embedded recursion is wrongly conceived as an expression of the familiar counting or accumulation technique within loops, not the consequence of the analysis of the original problem. As a remedy, students could be taught to think declaratively when programming recursively in imperative languages that is, to distinguish specification from evaluation. Myreen and Gordon (2009) proposed a unique method of translating programs, through complete automatic deduction, into recursive functions defined directly in the native logic of a theorem prover. This technique has a lot of benefits for program verification. Lokshtanov et

al. (2018) discussed exhaustively when recursion is better approach to iteration using a linear-time algorithm for acyclicity with few error vertices. The results of an empirical study was carried out on 130 students in Introduction to programming classes. Their initial preference, success rate, comprehension and subsequent preference were identified and studied when dealing with programming tasks which could be programmed using either iteration or recursion. It showed that students prefer iteration to recursion (Sulov, 2016). Johnson-Naird et al. (2021) explained how naïve individuals devise informal programs in natural language and is itself implemented in a computer program that creates programs using recursion. Jinping (2013) focused on the analytical program writing technique of the recursive algorithm, as well as the optimization of recursive program. Zhu and Sun (2023) analyzed the concept and function of C language recursion, process description of recursion algorithm, and finally gives the implementation strategy of C language recursion algorithm. Liu and Stoller (1999) states a powerful and systematic method, based on incrementalization, for transforming general recursion into iteration. An industrial algorithm implemented using java programming language was able to automatically change iterative loops to corresponding recursive methods and was generalized to cover other languages that support recursion (Insa and Silva, 2015).

We can scientifically use function composition to describe recursive programs and from which iterative program can be written without difficulty.

METHODOLOGY

Function composition will be used to describe how to solve computable feasible functions using recursion and then use the technique to program the given problem iteratively. Given two functions if one function becomes an input for the other, then function composition has taken place. Example given the function f(x) and another g(x) if g(x) becomes an input of f(x) then the composition will be written as f(g(x)), conversely it is also true if f(x) becomes the input of g(x) the composition will be written as g(f(x)). It can be buttressed with the following example: given $f(x) = 2x + 3x^2$ and $g(x) = x^2$, f(g(x)) will be equal to $2(x^2) + 3(x^2)^2$. Note that the output of f(g(x)) is not same as that of g(f(x)).

Let apply the method in the solution of the product of two positive integer values using recursion. The product of numbers is the sum of products which is function composition. We solve for sum recursively and then use product as an input for the sum function.

Sum(a, b)…….....the sum function.
Sum(a,0)…….....the base case for the sum function returns a
1 + Sum(a, b-1)…….....the recursive case for sum function
Product(a, b)…….....the product function
Product(a,1)…….the base case for the product function returns a
Sum(a, Product(a, b))…….the recursive for product function which is function composition.

Iterative procedure can be easily deduced from the recursive technique thus:
The initial value is derived from the recursive variable b which will be assigned 1 from the base case. The recursive value b will also be used to set the loop condition, then the loop is incremented towards making the condition false so that it can terminate. With these a computer programmer can have deep knowledge of iteration through recursion.

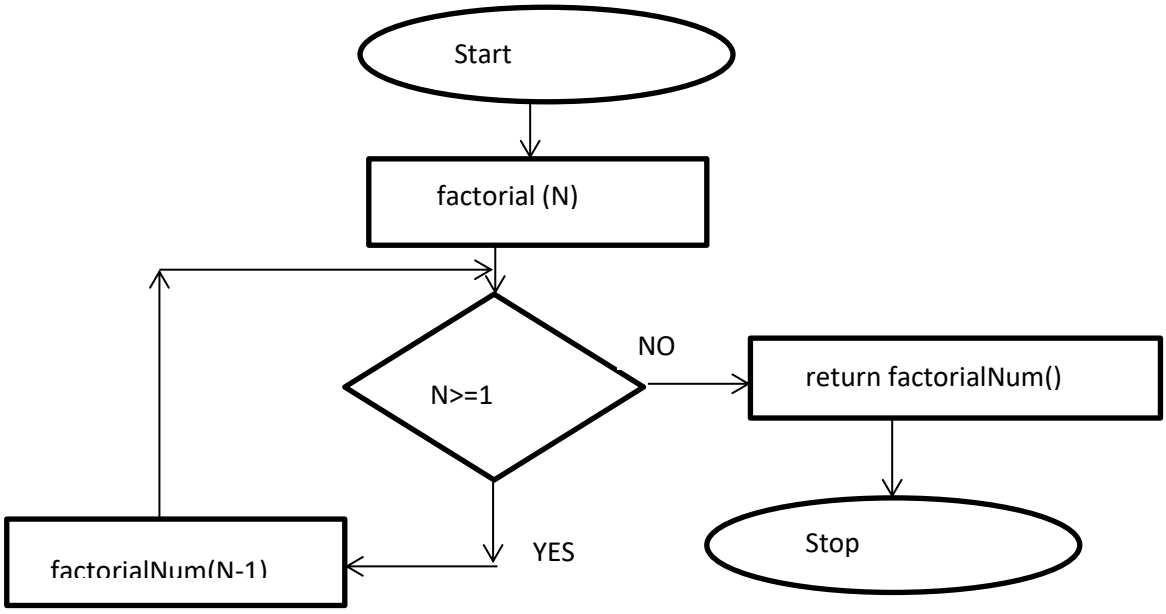

Fig. 1: Flowchart to represent factorial function using recursion

The pseudo code for solving problems recursively

(a) Set up a for loop:

(i) This should initialize a counter which will iterate towards an end condition.

(ii) Insert your instructions into the for loop.

(b) When the for loop ends, print the solution to the problem.

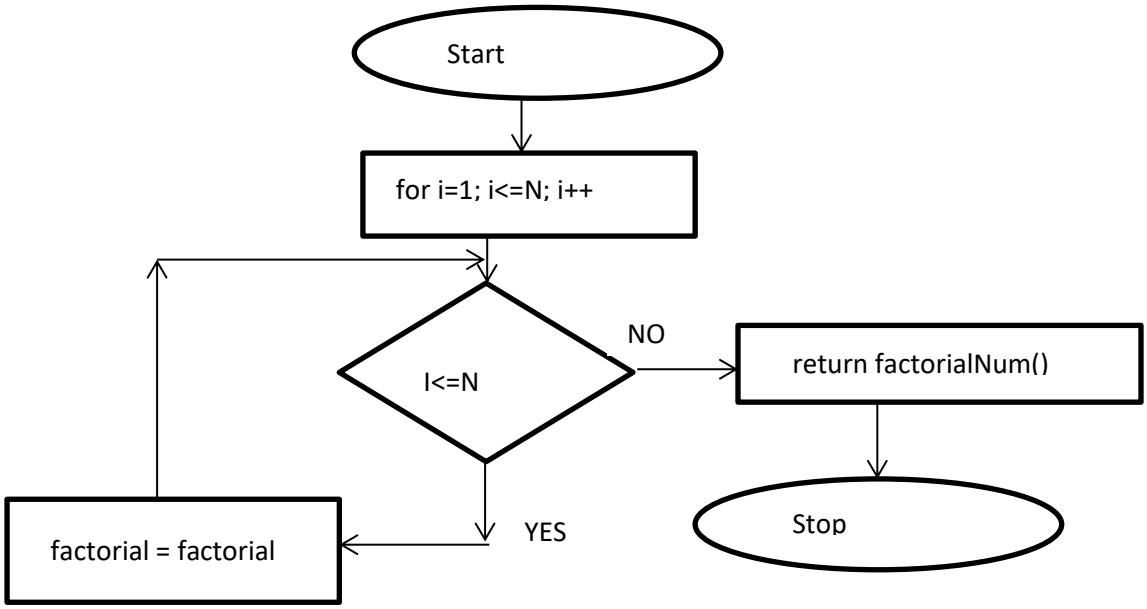

Fig. 2: Flowchart to represent factorial function using iteration

The pseudo code for solving problems recursively.

(1) Set up the recursive function

(a) Create an if statement

(i) This will contain an instruction which will call itself repeatedly to solve the problem whilst the condition is true.

(ii) Once condition is false, end the function.

Example 1

Let solve product (3, 2)

product (3, 2) = sum (3, product(3.2)) from the function composition.

product (3, 1) = sum (3, product(3,1)) the product function decrementing towards the base case of one (1).

product (3, 1) = 3 the base case for the product function, this clearly explains that the product of 3 and 1 is 3.

1 + sum (3, 3) substituting the value of the product function into the sum of products.

1+ sum (3, 2) the recursive part of the sum function decrements towards the base case.

1 + sum (3, 1) the sum function continues decrementing towards the base case.

1 + sum (3, 0) the sum function gets to the base case and returns the value three.

Sum (3, 0) = 3 this value is substituted backwards into 1 + sum (3, 1) which gives four. The value four (4) is the substituted backwards into 1 + sum (3, 2) which gives five. The value five (5) is then substituted into 1 + sum (3.3) which gives the value six (6).

Example 2.

Factorial

The factorial function can also be realized either recursively or iteratively. The recursive approach can be written down using function composition thus:

f(x) is the factorial function. When x is either zero or one is the base case and in both conditions one is returned, x * f(x-1) is the recursive case which continues to compose until it gets to the base case before computation to get the result is initiated. For example for x equal to 5 the composition function can be written thus: f(f(f(f(f(x))))) the function starts processing from the highest value of x which is 5 and accumulation the intermediate results in the stack until it gets to the base case which can be implemented to be either one or zero. For iteration the loop control variable is initialized to the base case which is one preferably and the product of the loop control variable is compounded on a designated initialized variable.

RESULTS AND DISCUSSION

Computer program for the multiplication operation using recursion

```
package tutorials;
import java.util.Scanner;
public class recursion {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int firstNumber;
        int secondNumber;
        int counter=0;
            int max = 2;
            do {
        System.out.print( "enter first number ");
        firstNumber = input.nextInt();
```

```
                System.out.print( "enter second number ");
                secondNumber = input.nextInt();
                System.out.print("the product of "+firstNumber+ " and "+secondNumber + "
= "+multNumbers(firstNumber,secondNumber));
                counter = counter + 1;
                System.out.println();
                        }
                        while(counter<max);
        }
        public static int addNumbers(int x, int y){
                if (y == 0)
                return x;
                else
                return 1+ addNumbers(x, y-1);
        }
                        public static int multNumbers(int x, int y){
                        if (y == 1)
                        return x;
                        else
                        return addNumbers(x, multNumbers(x, y-1));
                        }

        }
```

Program Output
enter first number 3
enter second number 3
the product of 3 and 3 = 9
enter first number 2
enter second number 6
the product of 2 and 6 = 12

Computer program for the multiplication operation using iteraion

```
package tutorials;
import java.util.Scanner;
public class MultiplicationIteration {
                public static void main (String [] args){
                        Scanner input = new Scanner(System.in);
                        int counter=0;
                        int max = 2;
                        do {
                                int firstNumber, secondNumber;
                        System.out.print( "enter first number ");
                        firstNumber = input.nextInt();
                        System.out.print( "enter second number ");
                        secondNumber = input.nextInt();
                        System.out.print("the product of "+firstNumber+ " and
"+secondNumber + " = "+multNumbers(firstNumber,secondNumber));
                        counter = counter + 1;
                        System.out.println();
                                }
```

```
                    while(counter<max);
        }
                    public static int multNumbers(int x, int y){ // function
definition
                    int secondNumber=0;
                    for (int i = 1; i <=x; i++){
                     secondNumber +=1;
                    }
                    return secondNumber * y;
                    }
        }
```

Program Output

```
enter first number 12
enter second number 3
the product of 12 and 3 = 36
enter first number 4
enter second number 4
the product of 4 and 4 = 16
```

Computer program for factorial operation using recursion

```
package tutorials;
import java.util.Scanner;
public class FactorialRecursion {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int counter=0;
        int max = 4;
        do {
            int Number;
        System.out.print( "enter number ");
        Number = input.nextInt();
        System.out.print("the factorial of "+Number+ " using recursion"+  " = "+
performFactorial(Number));
        counter = counter + 1;
        System.out.println();
            }
            while(counter<max);
    }

            public static int performFactorial(int x){ // function definition
            if (x==1)
            return 1;
            else
            return x * performFactorial(x-1);
            }
            }
```

Program output

```
the factorial of 12 using recursion = 479001600
enter number 11
the factorial of 11 using recursion = 39916800
```

```
enter number 3
the factorial of 3 using recursion = 6
```

Computer program for factorial operation using iteration.

```
package tutorials;
import java.util.Scanner;
public class IterationFactorial {
        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                int counter=0;
                int max = 4;
                do {
                        int Number;
                System.out.print( "enter number ");
                Number = input.nextInt();
                System.out.print("the factorial of "+Number+ " using iteration"+  " = "+
performFactorial(Number));
                counter = counter + 1;
                System.out.println();
                        }
                        while(counter<max);
        }
                        public static int performFactorial(int x){ // function definition
                        int fact = 1;
                        for (int i = 1; i<=x; i++) {
                         fact = fact * i;
                        }
                        return fact;
                        }
                        }
```

Program output.

```
enter number 19
the factorial of 19 using iteration = 109641728
enter number 12
the factorial of 12 using iteration = 479001600
enter number 6
the factorial of 6 using iteration = 720
enter number 2
the factorial of 2 using iteration = 2
```

The implementations and outputs clearly show both logically and in simplicity how the concepts of recursion and iteration relates, and how the knowledge of recursion strengthens the computer programmer on not only the programming of tasks using iteration from knowledge derived from recursion, but also assist the programmer to understand logic in programming which is the salient ingredient any developer should acquire. Recursion is slightly contrasting and tricky to iteration. While recursion follows LIFO (Last-In-First-Operation) which is a stack operation. The stack produces intermediate results in the stack memory as the recursive operations moves towards the base case. The intermediate results will not be processed to completion until the program execution gets to base case. The base case is the terminating point in a recursive function, so, the value derived from the base case

is used to substitute backwardly to where the recursive call started. If the base case is not properly written it might either make the program to run without terminating and thereby exhausting the stack memory allocated for the function call or erratic result will be produced. So, programming recursively reduces code complexity, increases code readability and gives the programmer a detail understanding of the task and how the computer solves it. The problem is defined and solved based on itself. Since the programmer knows how to define the base case which is the termination condition, and also can define the recursive condition which is the fragmentation of the problem into smaller incremental parts, iteration can easily be implemented from the knowledge of recursion. The programmer can now easily identify the loop initialization, condition and update when solving a task using iteration. Because recursion uses a divide and conquer strategy. Recursive codes can easily be debugged, because if an integral part of the code is correct then increasing the input of the subsequent recursive function calls will be correct as well.

Finally, having good knowledge on recursion will not only assist in the understanding and implementation of tasks iteratively and comprehension of programming logic but will assist the programmer to understand how to approach a task and when to either use iteration or recursion without abusing both concepts.

CONCLUSION

The research elaborately describes the fundamental confusing concepts of iteration and recursion by approaching loops using recursion. These core theories ones understood can assist both beginners and professional programmers to implement many algorithms using recursion and iteration without abusing the use of both concepts.

REFERENCES

Darsh, J. (2019). Algorithms: Recursion and Iteration. 10.13140/RG.2.2.31991.50080.

Endres, M., Weimer, W., Kamil, A. (2021). An Analysis of Iterative and Recursive Problem Performance. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432391

Insa, D., Silva, J. (2015). Automatic transformation of iterative loops into recursive methods,Information and Software Technology,Volume 58,2015, Pages 95-109,ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2014.10.001.

Jinping, S. (2013). International Journal of Hybrid Information Technology Vol.6, No.6 (2013), pp.127-134 http://dx.doi.org/10.14257/ijhit.2013.6.6.11 ISSN: 1738-9968 IJHIT Copyright ⓒ 2013 SERSC Discussion on Writing of Recursive Algorithm

Johnson-Laird, P.N., Bucciarelli, M., Mackiewicz, R., & Khemlani, S.S. (2021). Recursion in programs, thought, and language. *Psychonomic Bulletin & Review, 29*, 430 - 454.This article presents a theory of recursion in thinking and language

Liu,Y. A., Stoller, S., D. (1999). From recursion to iteration: what are the optimizations. Computer science department, lindley hall 215, Indiana university,Bloomington, IN 47405.

Lokshtanov, D., Ramanujan, M.S., Saurabh, S. (2018). When Recursion is Better than Iteration: A Linear-Time Algorithm for Acyclicity with Few Error Vertices. Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (pp.1916-1933) DOI:10.1137/1.9781611975031.125

Myreen, M. O., Gordon, M. J.C. (2009). Transforming Programs into Recursive Functions, *Electronic Notes in Theoretical Computer Science*, Volume 240,2009,Pages 185-200,ISSN 1571-0661,https://doi.org/10.1016/j.entcs.2009.05.052.

Rinderknecht, C. (2014). A Survey on the Teaching and Learning of Recursive Programming. Informatics in Education. 13. 87-119.

Sharma, Y., Vishwas, S. (2022) Generating Equivalent Iterative Versions for Recursive Algorithms. *International Research Journal of Modernization in Engineering Technology and Science* Volume: 04/Issue: 07/ www.irjmets.com.

Sulov, V. (2016) Iteration vs Recursion in Introduction to Programming Classes: An Empirical Study, *Cybernetics and Information Technologies* Volume 16, No 4 Sofia 2016 Print ISSN: 1311-9702; Online ISSN: 1314-4081 DOI: 10.1515/cait-2016-0068.

Zhu, Z., Sun, W.(2021). Research on the Implementation of Recursive Algorithm Based on C Language. *Journal of Physics: Conference Series*. 2023. 012015. 10.1088/1742-6596/2023/1/012015.