C – Introduction & Environment Setup
- Introduction to Language
- Introduction to Programming
- What is a Translator
- Types of Applications
- Platform Dependency in C
- Introduction to C Programming Language
- History of C Programming Language
- Library and IDE in C Language
- Structure of C Program
- Environment Setup for C Program
- Codeblocks IDE Setup in Windows for C Program Development
- Creating a new project using CodeBlocks IDE

C – Fundamentals
- Basic Syntax of C Program
- Different Parts of a C Program
- Constants in C
- Operators in C
- Unary Operators in C
- Variables in C
- Types of Variables in C
- Introduction to Functions in C
- Data Types in C
- Integer Data Types in C
- Character Data Types in C
- Sizeof and Limits of Data Type in C
- Type Casting in C

C – Control Statements

Recursive Functions in C

Back to: C Tutorials For Beginners and Professionals

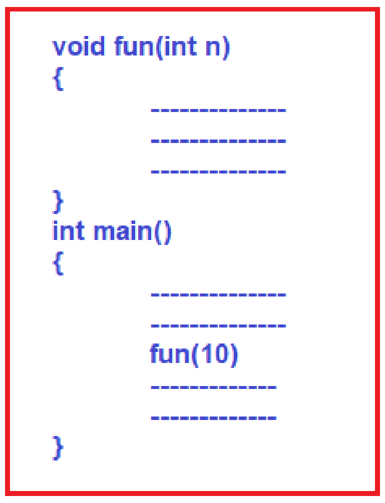
Recursive Functions in C

In this article, I am going to discuss the **Recursive Functions in C** with examples. Please read our previous articles, where we discussed the Local Vs Global Variables in C. At the end of this article, you will understand the following pointers.

1. **What is Recursion?**
2. **How does recursion work in C?**
3. **What are the advantages of recursive functions in C?**
4. **What are the disadvantages of recursion in C?**
5. **How recursive functions are classified?**
6. **Can we call the main function itself in C?**

What is Recursion?

Before understanding Recursion, let us have a look at the below image. Here we have the main function and one more function called function 'fun' and that fun function is called by the main function.



```
void fun(int n)
{
        --------------
        --------------
        --------------
}
int main()
{
        --------------
        --------------
        fun(10)
        -------------
        -------------
}
```

First, we need to understand how does this function call is made and how it works. Here, once I start executing the program, it will start executing the program from the main function. First, it will execute the first statement, then it will execute the second statement and then it will execute the third statement i.e. it will call the fun function. Here, the control will move to the fun function definition and start executing that fun function. Inside the fun function, it will start executing the first statement, then the second, and then the third statement. Once it has finished (once the third statement is executed inside the fun function) and the control again comes back to the same line i.e. third line of the main function. If any other operations are present in that line they will execute. Otherwise, it will execute the fourth statement and then the fifth statement.

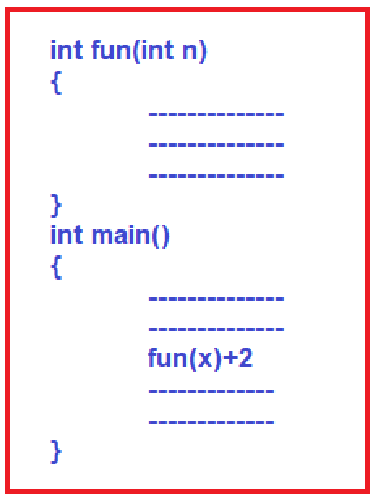
What does it mean by any other operations?

Let say the fun function is returning something and in the main function, I have written added by 2. So, the value of returning from the fun function has to be added by two. So, this addition needs to be done once the function has returned to the main function with some value. Assume that the fun function has a return value of 10. So, 10+2 can be done, only if the fun(10) has returned the value. This is the important point that you should remember for understanding the recursion. For better understanding, please have a look at the below image.

Control Statements in C
If Else Statements in C Langauge
Nested If-Else Statement in C
Switch Statements in C
While Loop in C
Nested While Loop in C
Do While Loop in C
For Loop in C Program
Break Statement in C Language
Continue Statement in C Language
Return Statement in C Language
Goto Statement in C

C – Functions & Storage Classes
Functions in C
Types of User Defined Functions in C
Call by Value and Call by Address in C
Local vs Global Variables in C
Recursive Functions in C
How Recursion Uses Stack in C
Time Complexity of Recursive Function in C
Tail and Head Recursion in C
Adding user defined functions in C Library
Storage Classes in C

C – PreProcessing
Pre-Processing in C
Macro Substitution Directives in C
File Inclusion Directives in C
Conditional Compilation Directives in C
Miscellaneous Directives in C



```
int fun(int n)
{
        --------------
        --------------
        --------------
}
int main()
{
        --------------
        --------------
        fun(x)+2
        -------------
        -------------
}
```

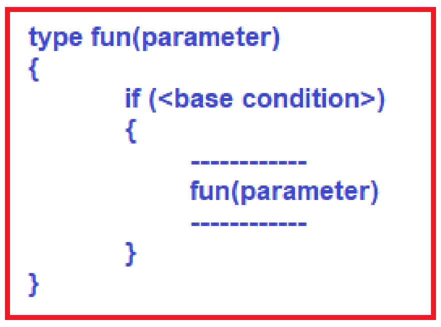
With this keep in mind, let us proceed and understand what is a recursive function.

What does it mean by Recursive Function?

Function calling itself is called recursion. The function in which control is present, if it calls itself again then it is called recursion process. Recursion is a process by which function calls itself repeatedly until some specified condition has been satisfied. The process is used for repetitive computation in which each action is stated in terms of a previous result. Many iterative problems can be written in this form.

In order to solve a problem recursively, two conditions must be satisfied. First, the problem must be written in a recursive form, and second, the problem statement must include a stopping condition. If a recursive function contains local variables, a different set of local variables will be created during each call. The variables will represent a different set of values each time the function is executed. Each set of values will be stored on the stack, so that they will be available as the recursive process “unwinds” i.e., as the various function calls are “popped” off the stack and executed.

The general form of recursion is given below.



```
type fun(parameter)
{
        if (<base condition>)
        {
              ------------
              fun(parameter)
              ------------
        }
}
```

This is a recursive function i.e. a function is calling itself. If a function is calling itself then it is called a recursive function. Inside the function body, if you see it is calling itself again, then it is a recursive function.

One important point that you need to remember is, inside recursion, you can see that there is a base condition. So, there must be some base condition that will terminate the recursion. There must be some method to terminate the recursion otherwise it will go into infinite calling. First, we need to call the function the first time, then it will call itself repeatedly again and again. So, there must be some condition at which it must stop.

In this example, the function will call itself as long as the base condition is true or it can stop if the base condition is true. Here if the condition becomes false it will not call further and it stops. So, let us take some examples of the recursive function and study how it works.

How does Recursion Work in C?

Let us look at an example to understand how recursion works. Please have a look at the following code. Here, I have a main function which is having some value in the variable x, and calling function fun1 bypassing that variable ‘X value. The function fun1 that taking parameter ‘n’ which will accept the x value and if the condition is ‘true, it is printing and then calling itself. I have used a C language code here. So, it’s printing and again calling itself for a reduced value of n.

```
void fun1(int n)
{
 if(n>0)
 {
      printf("%d ",n);
      fun1(n-1);
 }
}
void main()
{
 int x=3;
 fun1(x);
}
```

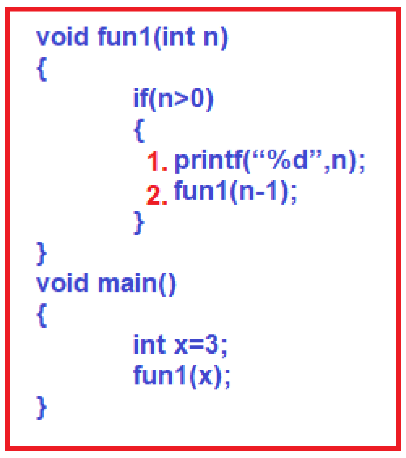
Output: **3 2 1**

In the above example, we are passing 3 to the fun1 function from the main function. Let us see what will be the result and how it works? Let’s, trace this recursive function and check.

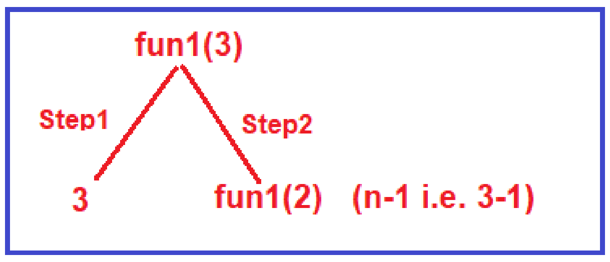
How to trace a Recursive Function?

C – Arrays
Array in C
One Dimensional Array in C
Multi-Dimensional Array in C
Functions using Array in C
Array Exercise in C
C – Strings
Introduction to Strings in C
Strings in C Language with Examples
PreDefined String Functions in C
How to Find Length of a String in C
How to Change Case of Alphabets in a String in C
How to Count Vowels and Consonants in a String in C
How to Validate a String in C
How to Reverse a String in C
How to Compare String and Checking Palindrome in C
How to Find Duplicates in a String in C
Finding Duplicates in a String using Bitwise Operations in C
How to Check if 2 Strings are Anagram in C
Permutation of Strings in C
C – Structure & Union
Structure in C
Union in C
C – Pointers
Pointers in C
Arithmetic Operations on Pointers in C
Pointer-to-Pointer in C
Pointer to Array of functions in C
Passing Pointer to Function in C

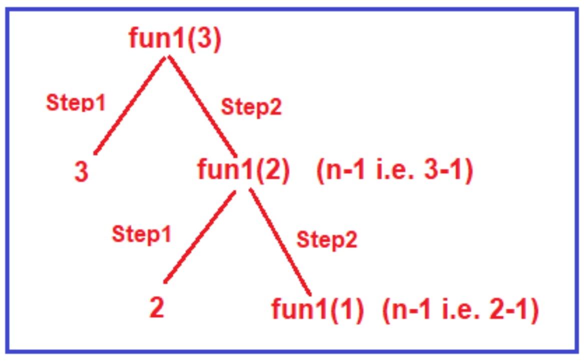
A recursive function is traced in the form of a tree. So, let us start tracing the above example. When the condition is true inside the fun1 function there are two statements to be executed. In the first statement, it will print the n value and in the second statement it will call itself bypassing (n-1) and this has to be done only when n is greater than 0.


```
void fun1(int n)
{
        if(n>0)
        {
          1. printf("%d",n);
          2. fun1(n-1);
        }
}
void main()
{
        int x=3;
        fun1(x);
}
```

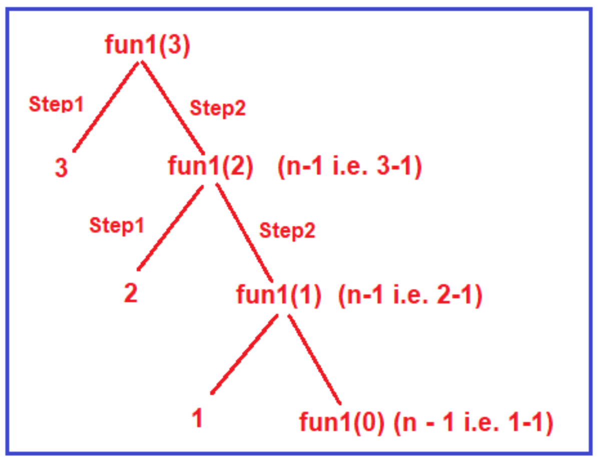
Let's start tracing, from the main function, we are calling function fun1 bypassing 'X i.e. value 3. So, the first time it has got 3, and 3 is greater than 0 and hence the condition becomes true. So, the first step is to print 'n' and the second step is to call itself again fun1 for 3-1 i.e. 2. Here, the fun1(3) call has not been completed. It calling itself again.



So, it will call itself again bypassing fun1(2). So, let us execute fun1(2), again it will start, 2 is greater than '0' and hence the condition becomes true. So, the first step is to print 2 and then call itself again bypassing fun1(1). Now, the fun1(2) call has not finished, it has printed 2 and it has to call fun1(1).



So again, a new call, a fresh call, that fresh call is fun1(1). 1 is greater than 0, so we have to perform the two steps. The first step is to print 1 and then call itself by passing fun1(0). Now, the fun1(1) call has not finished, it has printed 1 and it has to call fun1(0).



Now, fun1(0), 0 is greater than 0, no it's not greater than 0. So, it will not enter inside, it will not perform those two steps and it does nothing. So, no printing and no calling and it will not enter inside and it will come outside of the function. So, the fun1(0) call doesn't do anything. So, it will go back to the previous function call and so on and finally come out from the fun1 to the main function where it is initially called. So, a recursive function forms a tree and this is called a tracing tree of a recursive function. Now we will take one more example.

Example:

Please have a look at the below example which is also an example of the recursive function.

```
void fun2(int n)
{
 if(n>0)
 {
     fun2(n-1);
     printf("%d",n);
 }
}
void main()
{
 int x=3;
```

Character Pointer in C
Pointer to Constant in C
Null Pointer in C
Void Pointer in C
Dangling Pointer in C
Function Pointer in C

Pointer Structure Array & Functions
Pointer to Structure in C
Passing Array as a Parameter to a Function in C
How to pass Structure as a Parameter in C

C – CLA Enum Typedef Dynamic Memory
Command Line Arguments in C
Enum in C
Typedef in C
Dynamic Memory Management in C
Dynamic Array Creation in C

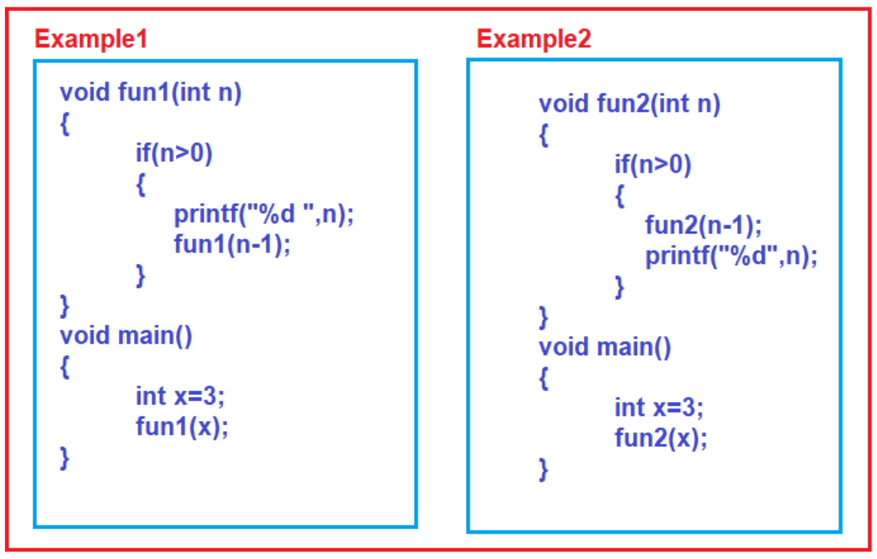
C – File & Error Handling
File Handling in C Language
Error Handling in C

C – Interview
C Interview Questions and Answers

Popular C Books
Most Recommended C Books

```
  fun2(x);
}
```

The above example is very similar to the first example. Let me compare both the example and show you the difference.


Example1

```
void fun1(int n)
{
        if(n>0)
        {
            printf("%d ",n);
            fun1(n-1);
        }
}
void main()
{
        int x=3;
        fun1(x);
}
```

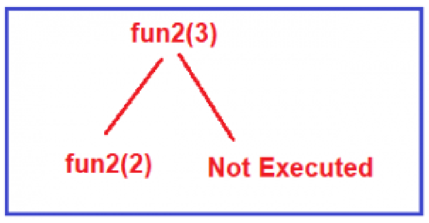
Example2

```
void fun2(int n)
{
        if(n>0)
        {
            fun2(n-1);
            printf("%d",n);
        }
}
void main()
{
        int x=3;
        fun2(x);
}
```

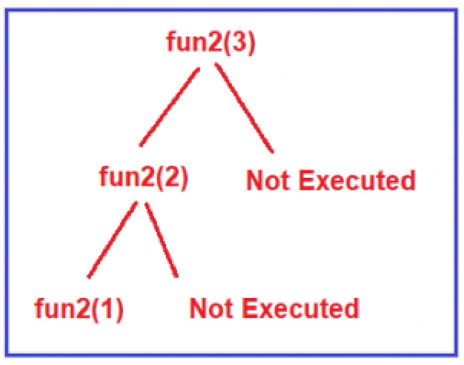
If you look at the main function of both the example, they are having one variable called x and calling one function (Example1 calling fun1 function and Example2 calling fun2 function) bypassing that x value.

The difference in both the example is that, in example 1 inside the fun2 function if the condition is true (i.e. n > 0), first it is printing the n value and then calling itself but in example 2, first it is calling itself, and then printing the n value, then what will be the output. Let's trace example 2 and find out the output.

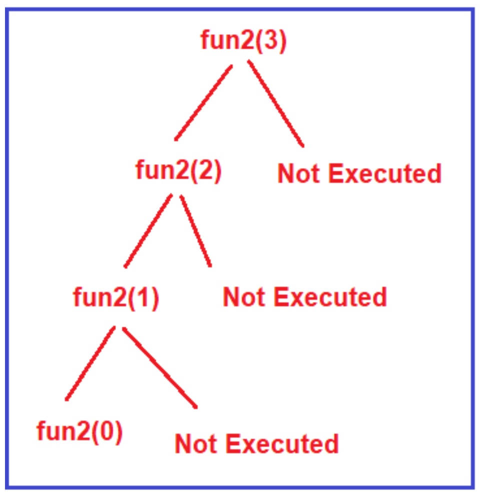
The program execution will start from the main function. The main function calling the function fun2 by passing value 3 i.e. fun2(3). Inside the fun2 function, first, it will check whether n > 0 and here, n is 3, so 3 is greater than 0 and the condition is satisfied. The first statement within the if block is executed i.e. it calls the fun2 function by passing n-1 i.e. 2. What about the second statement i.e. printing? It will not be executed at this point in time. The point that you need to remember is first the first statement has to be finished in order to execute the second statement i.e. printing. For better understanding please have a look at the below image.



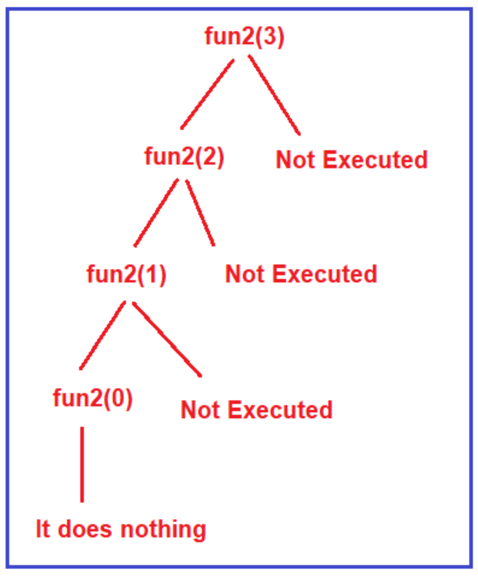
Let us take fun2(2) call, with n=2, the condition again satisfied as 2 is greater than 0. Again, two steps, first it will call fun2 with n-1 i.e. it will call itself for fun2(1) and the second statement will not be executed at this point. Once the first statement is finished, the second statement will execute. At this point, the tracing tree will be like below.



Let us trace fun2(1). Again 1 is greater than 0 and hence the condition is satisfied and again two steps. In the first step, it will call itself bypassing n-1 i.e. fun2(0), and similarly, the second statement will only execute once the first statement completes its execution. So, at this point, the tracing tree of this recursive function is like the below.



The next call is fun2(0). Now fun2(0), 0 is greater than 0, no. So, it will not enter inside this 'if' block and it will come out i.e. it does nothing. So, this call with parameter 0 has terminated.



Now once this call has been terminated the control should go back to the previous call. The previous call was fun2(1), it will go back to the function call and execute the next statement i.e. the second statement which is nothing but to print the n value. At this call, the n value is 1 and hence it will print 1. For better understanding, please have a look at the below image.

Then it will go back to the previous call i.e. fun2(2), and the second thing that is remaining here is printing, so the value 2 is printed then it will come out of this function and finish. For better understanding, please have a look at the following image.

Once the fun2(2) call finishes, it goes back to the previous call i.e. fun2(3), and the second thing that is remaining here is printing, so the value 3 is printed. And the output you will get from this function is 1 2 3 as shown in the below image.

The output of example 1 was 3, 2, 1 and the output of example 2 is 1, 2, 3.

Now, let us compare both of the examples, in example 1, first, the printing was done and then the recursive call was made but in example 2, first the recursive call was made and then the printing was done at returning time.

Note: The most important thing that you need to understand in recursion is that recursion has two phases. One is the calling phase and the other one is returning phase.

Real-Time Example of Recursion:

Now, let us take one example and see the differences between them. Suppose there is a room and in that room, there is a bulb and a door. From that room, you can enter into one more room, and again in that room, there is a door and there is also a bulb fixed in it. Again, from that room, you can enter into one more room and in that room also there is a door and a fixed bulb. So, there are a total of 3 rooms i.e. room 1, room 2, room 3 and each room is having a bulb and a door.

Now if I give you the following two instructions,

1. **Switch on the bulb,**
2. **Go to the next room.**

Now if I make you stand in room 1 and ask you to perform the above two 2 steps recursively, then what will you do. Let us check. In the first room, you will switch on the bulb i.e. first step, the second step you will go to the next room i.e. 2. So, the first bulb that you have switched on.

Then you will enter into next room i.e. room 2 and again what you will do is recursively is switch on the bulb i.e. second bulb and go to the third room. Similarly, in the third room, recursively you will switch on the bulb i.e. bulb 3, and go to the next room. Now, there is no next room further. So, it was more like a recursive call, a function calling itself again and again. This is termination there is no next room. What do you do now?

You will definitely come out of those rooms. Your work has finished. So, from the third room, you will come back to the second room. Now, will you do anything in the 2nd room? No, you'll simply come out from the 2nd room to the 1st room and you are also not going to do anything in the 1st room. So, you will come out from the first room also.

So, third room to second room, second room to first room and then you'll simply come out. It means the first step was you were entering into the rooms; it was just like calling. Once you have into the third room, you cannot go further. So, you have to return back, this is nothing but returning.

So, you have gone through two phases. One is the calling phase and another one is returning phase or we can say that entering one is ascending and returning one is descending. So, you are going to next, next, next ascending and then you came back, back, back, so descending. So, you have done ascending and descending and, in this example, descending time we have not done anything.

Example: calculate factorial using Recursive Functions in C

```
int factorial (int n)
{
    if(n==1)
       return (1);
    return(n*factorial(n-1));
}
```

Here, the factorial function will call itself but with a smaller value of n. The complete program is given below.

```
#include <stdio.h>
int factorial(int number);
int main()
{
    int x = 6;
    printf("The factorial of %d is %d\n", x, factorial(x));
    return 0;
}
int factorial(int number)
{
  if (number == 1)
        return (1); /* exiting condition */
   else
     return (number * factorial(number - 1));
}
```

Output:

We declare our recursive factorial function which takes an integer parameter and returns the factorial of this parameter. This function will call itself and decrease the number until the exiting, or the base condition is reached. When the condition is true, the previously generated values will be multiplied by each other, and the final factorial value is returned. We declare and initialize an integer variable with the value “6” and then print its factorial value by calling our factorial function.

What are the advantages of Recursive Functions in C?

1. Function calling-related information will be maintained by recursion.
2. Stack evaluation will take place by using recursion.
3. Prefix, postfix, infix notation will be evaluated by using recursion

What are the disadvantages of Recursion in C?

1. It is a very slow process due to stack overlapping.
2. The recursive program can create stack overflow.
3. The recursive program can create infinite loops.

How recursive functions are classified?

Recursions are classified into two types

1. Internal recursive process
2. External recursive process

If one recursive function is calling itself then it is called the internal recursive process and if one recursive function calling another recursive function then it is called an external recursive process.

Can we call the main function itself in C?

The main() function can be called itself but if we are using auto variable then it becomes stack overflow error. Let us see the program for better understanding.

```c
#include <stdio.h>
int main()
{
    int a = 5;
    ++a;
    printf("%d", a);
    if(a <= 6)
        main();
        printf("%d", a);
    return 0;
}
```

Output:

Example: to calculate the power of a number using the Recursive Function in C.

```
#include <stdio.h>
int power(int b, int e)
{
    if(e < 0)
        return 0;
    else if(e == 0)
        return 1;
    else
        return( b * power(b, e-1));
}
int main()
{
   int a, b, p;
   printf("Enter the value of a : ");
   scanf("%d" , &a);
   printf("Enter the value of b : ");
   scanf("%d" , &b);
   p = power(a, b);
   printf("%d^%d value is %d", a, b, p);
   return 0;
}
```

Output:

In the next article, I am going to discuss [Adding user-defined functions in C Library](#)with Examples. Here, in this article, I try to explain **Recursive Functions in C**. I hope you enjoy this Recursive Functions in C article. I would like to have your feedback. Please post your feedback, question, or comments about this Recursive Functions in C article

← Previous Lesson
Local vs Global Variables in C

Next Lesson →
How Recursion Uses Stack in C

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name*

Email*

Website

Post Comment