International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

C-LANGUAGE PROGRAMMING-RECURSION

AUTHOR-R.SATISHKUMAR
CO-AUTHOR- DR. P. GANESHBABU
V.VANEESHWARI

The recursion process in C refers to the process in which the program repeats a certain section of code in a similar way. Thus, in the programming languages, when the program allows the user to call any

function inside the very same function, it is referred to as a recursive call in that function. Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
recursion(); /* function calls itself */
}

int main() {
recursion();
}
```

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

Number Factorial

The following example calculates the factorial of a given number using a recursive function −

```
#include <stdio.h>

unsigned long longint factorial(unsigned int i) {

if(i <= 1) {
return 1;
   }
return i * factorial(i - 1);
}
```

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

```
int  main() {
int i = 12;
printf("Factorial of %d is %d\n", i, factorial(i));
return 0;
}
```

When the above code is compiled and executed, it produces the following result −

Factorial of 12 is 479001600

Fibonacci Series

The following example generates the Fibonacci series for a given number using a recursive function −

Live Demo

```
#include <stdio.h>

intfibonacci(int i) {

if(i == 0) {
return 0;
   }

if(i == 1) {
return 1;
   }
returnfibonacci(i-1) + fibonacci(i-2);
}

int  main() {

int i;

for (i = 0; i < 10; i++) {
printf("%d\t\n", fibonacci(i));
   }

return 0;
}
```

When the above code is compiled and executed, it produces the following result −

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

0
1
1
2
3
5
8
13
21
34

The recursion process in C refers to the process in which the program repeats a certain section of code in a similar way. Thus, in the programming languages, when the program allows the user to call any function inside the very same function, it is referred to as a recursive call in that function.

In this article, we will take a look at the recursion in C according to the GATE Syllabus for CSE (Computer Science Engineering). Read ahead to learn more.

Table of Contents

When Does the Recursion Occur in C?

The recursion process comes into existence whenever any function calls some copy of itself, when it wants to work on a smaller problem. Thus, any function that performs a calling of itself is

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

known as a recursive function. Also, the calling of such functions is known as recursive calls. The process of recursion consists of various recursive calls. But note that a programmer must impose some termination condition of the recursion.

The iterative code is comparatively longer as compared to the recursion code. And yet, the recursion code is comparatively much more difficult to understand.

Why Do We Need Recursion in C?

One cannot apply recursion on any problem – it is mainly beneficial for those tasks that can get defined with similar types of subtasks. For instance, a programmer can apply recursion on searching, sorting, and even traversal problems. Keep in mind that the iterative solutions are comparatively much more efficient than the recursion. It is because the function call is overhead almost all the time. Thus, if we can solve any problem recursively, then we can also solve it iteratively. Yet, some of the problems are better solved recursively in a program. For example, the Fibonacci series, tower of Hanoi, factorial finding, etc.

How Does Recursion Work?

The recursion in a C program continues further until and unless it meets some condition to ultimately prevent it. Thus, if we want to prevent recursion in a code, we can use the if…else statement or some similar type of approach where one of the branches makes a recursive call while the other one doesn't.

Syntax

```
voidrecurse()

{

… .. …

recurse(); /* calling of the function by itself */

… .. …

}
```

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

```
int main()

{

… .. …

recurse();

… .. …

}
```

In the C programming language, we can make recursion happen, i.e. a function can call itself in C. But a programmer must be very careful while defining the exit condition for the function that will undergo recursion. Or else, this recursion may create an infinite loop in the program.

One can solve various mathematical problems using these recursive functions, for instance, generating the Fibonacci series, performing the calculation of the factorial of any number, etc.

The Recursive Function

The recursive functions perform certain tasks dividing them into various subtasks. Then there is also a termination condition that we need to define in the function. Some specific subtasks satisfy this termination condition. Due to this, the recursion ultimately stops, and we get the final results from the function as a return. Know more about recursive function in C.

The Number Factorial

Let us take a look at an example where we calculate the factorial of any number with the use of a recursive function in C:

```
#include <stdio.h>

unsigned long longint factorial(unsigned int x) {

if(x <= 1) {

return 1;
```

```
}

return x * factorial(x – 1);

}

int main() {

int x = 12;

printf("The factorial of the number %d is equal to %d\n", x, factorial(x));

return 0;

}
```

The compilation and execution of the code mentioned above will ultimately produce the result given below:

The factorial of the number 12 is equal to 479001600

The Fibonacci Series

Let us take a look at an example where we generate a Fibonacci series for any available function by utilising the recursive function in a code:

```
#include <stdio.h>

intfibonacci(int x) {

if(x == 0) {

return 0;

}

if(x == 1) {

return 1;
```

```
}

returnfibonacci(x-1) + fibonacci(x-2);

}

int main() {

int x;

printf("%d\t\n", fibonacci(x));

}

return 0;

}
```

The compilation and execution of the code mentioned above will generate the result as follows:

0

1

1

2

3

5

8

13

21

34

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

The Allocation of Memory in Recursive Methods

Every recursive call in a program leads to the creation of a new copy of the method in use in the memory. Thus, once the method returns some data, the copy gets finally removed from the memory. And since the stack stores all declared variables and everything (declared inside a function), each recursive call maintains a separate stack in itself. These stacks get progressively destroyed once all the values return to us from their corresponding functions.

The process of recursion involves much complexity when tracking and resolving the values available in every recursive call. And thus, we have to maintain their individual stacks and also track the values of all the defined variables in these stacks.

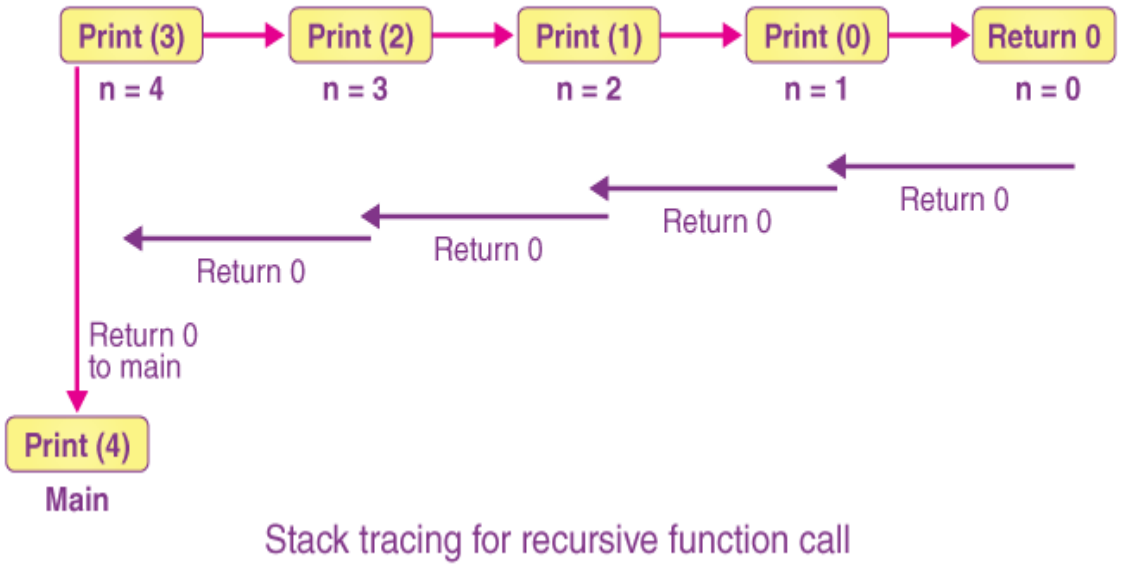
Let us look at another example to understand how the recursive functions get allocated with memory in a program:

```
int display (int x)

{

if(x == 0)

return 0; // the termination of a condition for recursion

else

{

printf("%d",x);

return display(x-1); // the recursive call in the code

}

}
```

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

Explanation of Code

In this code, we are examining the recursive function for the value x = 4. Here, first of all, every stack will get maintained, thus, printing the corresponding values available with x until the value of x becomes 0 in the end. Once the program reaches this termination condition, all the stacks available in the program will get destroyed one after the other when it returns 0 to the calling stack. Let us take a look at the image given below to understand more about the trace of stacks for any recursive function.



Stack tracing for recursive function call

The Disadvantages of Recursion Processes in C

- Any recursive program is generally much slower than any non-recursive program. It is because the recursive programs have to make the function calls. Thus, in this case, the program needs to save its current state entirely and then retrieve it later (again). It ultimately consumes much more time and makes the recursive programs comparatively much slower.
- In the case of recursive programs, more memory needs to hold the intermediate states in any stack. On the other hand, the non-recursive programs don't consist of any form of intermediate states. Thus, they also don't require extra memory in any case.

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

Practice Problems on Recursion in C

1. What would be the output obtained out of the program mentioned below if we input a value of 12?

```
#include<stdio.h>

intfibonacci(int);

void main ()

{

intx,a;

printf("Please enter the preferred value of the variable x?");

scanf("%d",&x);

a = fibonacci(x);

printf("%d",a);

}

intfibonacci (int x)

{

if (x==0)

{

return 0;

}

else if (x == 1)
```

```
{

return 1;

}

else

{

returnfibonacci(x-1) + fibonacci(x-2);

}

}
```

A. Please enter the preferred value of the variable x?12

144

B. Please enter the preferred value of the variable x: 12

120

C. Please enter the preferred value of the variable x is?12

120

D. Please enter the preferred value of the variable x is:12

144

Answer: A. Please enter the preferred value of the variable x?12

144

2. What would be the output obtained out of the program mentioned below?

```
#include<stdio.h>
```

International Journal of Research in Engineering and Applied Sciences(IJREAS)
Available online at http://euroasiapub.org/journals.php
Vol. 11 Issue 8, August -2021,
ISSN(O): 2249-3905, ISSN(P) : 2349-6525 | Impact Factor: 7.196| Thomson Reuters ID: L-5236-2015

```
#include<conio.h>

intfibo(int x) {

if (x == 0)

{ return 0;

}

if (x == 1)

{ return 1;

}

returnfibo(x – 1) + fibo(x – 2);

}

int main(void) {

int x;

printf(“The calculated Fibonacci sequence would be\n”);

for (x = 0; x < 10; x++) {

printf(“%d \t “, fibo(x));

} getch();

return 0;

}
```

A. The calculated Fibonacci sequence would be

0 1 1 2 3 7 8 13 23 34

B. The calculated Fibonacci sequence would be

0 1 2 2 3 5 8 13 23 34

C. The calculated Fibonacci sequence would be

0 1 1 2 3 5 8 13 21 34

D. The calculated Fibonacci sequence would be

0 1 2 2 3 5 11 13 21 34

Answer: C. The calculated Fibonacci sequence would be

0 1 1 2 3 5 8 13 21 34

REFERENCE-

1.AUTHENTIC GUIDE TO C-LANGUAGE-YASHWANT KANETKAR,
2.C-BASIC LANGUAGE FOR BEGINERS-A.J.GONZALEZ.
3.C PROGRAMMING-BRIAN W,KENNINGAM AND RITCHIE.