



SNS COLLEGE OF TECHNOLOGY



Coimbatore – 35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AI & ML

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

UNIT IV – FUNCTIONS AND POINTERS

TOPIC 4 – RECURSION FUNCTION



- Recursion is the process which comes into existence when **a function calls a copy of itself to work on a smaller problem.**
- Any **function which calls itself is called recursive function**, and such function calls are called recursive calls.
- Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion.
- Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks.
- For Example, recursion may be applied to sorting, searching, and traversal problems.
- Generally, iterative solutions are more efficient than recursion since function call is always overhead.
- Any problem that can be solved recursively, can also be solved iteratively. However, some problems are best suited to be solved by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.



Recursive Function

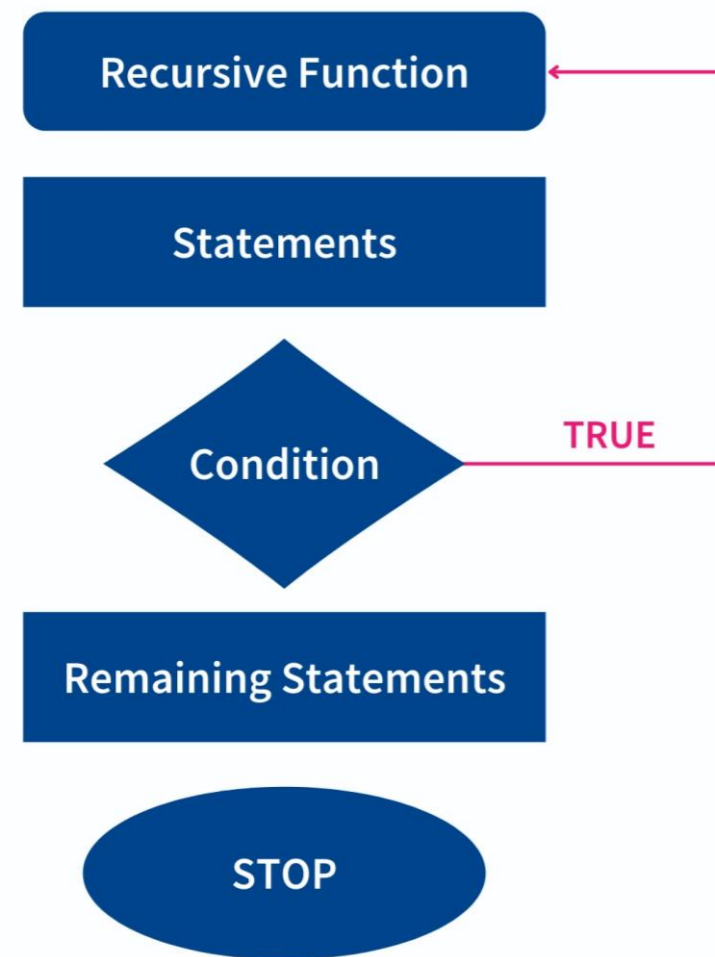
- ✓ A recursive function performs the tasks by dividing it into the subtasks.
- ✓ There is a termination condition defined in the function which is satisfied by some specific subtask.
- ✓ After this, the recursion stops and the final result is returned from the function.
- ✓ The case at which the **function doesn't recur is called the base case** whereas the **instances where the function keeps calling itself to perform a subtask, is called the recursive case.**
- ✓ All the recursive functions can be written using this format.

Basic Syntax

```
void recursive_fun()           //recursive function
{
    Base_case;                 // Stopping Condition
    recursive_fun();           //recursive call
}
int main()
{
    recursive_fun();           //function call
}
```



Flowchart of Recursion



In the following image, there is a recursive function inside which there is a recursive call that calls the recursive function until the condition of the problem is true. If the condition gets satisfied, then the condition is false, and the program control goes for the remaining statements and stops the program.

How does Recursion Work?

- ✓ The recursive function or method has two main parts in its body, i.e., the base case and the recursive case.
- ✓ While the recursive method is executed, first, the base case is checked by the program.
- ✓ If it turns out true, the function returns and quits; otherwise, the recursive case is executed.
- ✓ Inside the recursive case, we have a recursive call that calls the function inside which it is present.



Types of Recursion

There are two types of recursion in the C language.

Direct Recursion

Indirect Recursion

1. Direct Recursion in C

Direct recursion in C occurs when a function calls itself directly from inside. Such functions are also called direct recursive functions.

Following is the structure of direct recursion.

```
function_01( )  
{  
    //some code  
    function_01( );  
    //some code  
}
```



Example – Fibonacci Series

```
#include<stdio.h>
int fibonacci_01(int i)
{
    if (i == 0)
    {
        return 0;
    }
    if (i == 1)
    {
        return 1;
    }
    return fibonacci_01(i - 1) + fibonacci_01(i - 2);
}
int main()
{
    int i, n;
    printf("Enter a digit for fibonacci series: ");
    scanf("%d", & n);
    for (i = 0; i < n; i++)
    {
        printf(" %d ", fibonacci_01(i));
    }
    return 0;
}
```

1. In the given C program, we have declared a function named fibonacci_01().
2. It takes an integer i as input and returns the ith element of the Fibonacci series.
3. At first, the main() function will be executed where we have taken two variables i and n.
4. We will take input from the user that will be stored in n, and the for loop will execute till n iteration where with each iteration, it will pass the parameter to fibonacci_01() function where the logic for the Fibonacci series is written.
5. Now inside fibonacci_01() function, we have nested if-else.
6. If input = 0, it will return 0, and if the input = 1, it will return 1. These are the base cases for the Fibonacci function.
7. If the value of i is greater than 1, then fibonacci(i) will return fibonacci_01 (i - 1) + fibonacci_01 (i -2) recursively, and this recursion will be computed till the base condition.

```
Enter a digit for fibonacci series: 8
0 1 1 2 3 5 8 13
```



Indirect Recursion

Indirect recursion in C occurs when a function calls another function and if this function calls the first function again. Such functions are also called indirect recursive functions.

Following is the structure of indirect recursion.

```
function_01()  
{  
    //some code  
    function_02();  
}  
function_02()  
{  
    //some code  
    function_01();  
}
```

In the indirect recursion structure the function_01() executes and calls function_02(). After calling now, function_02 executes where inside it there is a call for function_01, which is the first calling function.

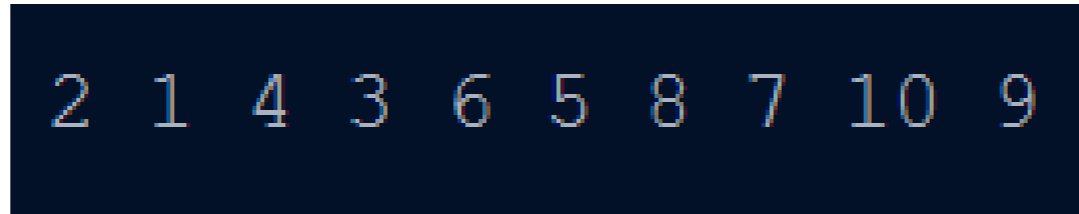


Example

```
#include<stdio.h>
void odd();
void even();
int n=1;
void odd()
{
    if(n <= 10)
    {
        printf("%d ", n+1);
        n++;
        even();
    }
    return;
}
void even()
{
    if(n <= 10)
    {
        printf("%d ", n-1);
        n++;
        odd();
    }
    return;
}
int main()
{
    odd();
}
```

Here is a C program to print numbers from 1 to 10 in such a manner that when an odd no is encountered, we will print that number plus 1.

When an even number is encountered, we would print that number minus 1 and will increment the current number at every step.



```
2 1 4 3 6 5 8 7 10 9
```

1. In this C program, we have functions named odd() and even().
2. A variable n is assigned with a value 1 as we have to take values from 1 to 10.
3. Now inside the odd() function, we have an if statement which states that if the value of n is less than or equals 10 add 1 to it and print.
4. Then the value of n is incremented by 1(it becomes even), and the even() function is called.
5. Now inside the even() function, we again have an if statement which states that if the value of n is less than or equals 10 subtract 1 from it and print.
6. Then the value of n is incremented by 1(it becomes odd, and the odd() function is called.
7. This indirect recursion goes on until the if condition inside both the functions becomes unsatisfied.
8. At last, we have the main() function inside, which we call the odd() function as the first number handle is 1, which is odd.



Difference between Recursion & Iteration

Recursion	Iteration
It is used with functions.	It is used generally with loops.
In each function call in recursion, extra Space is required in stack memory.	Here, in the case of each iteration, we do not require any space.
It terminates when the base condition is met.	It terminates when a condition becomes false.
Code size becomes smaller with recursion.	Code size is large in the case of iteration.



Advantages of Recursion

- ✓ The code becomes shorter and reduces the unnecessary calling to functions.
- ✓ Useful for solving formula-based problems and complex algorithms.
- ✓ Useful in Graph and Tree traversal as they are inherently recursive.
- ✓ Recursion helps to divide the problem into sub-problems and then solve them, essentially divide and conquer.

Disadvantages of Recursion

- ❑ The code becomes hard to understand and analyze.
- ❑ A lot of memory is used to hold the copies of recursive functions in the memory.
- ❑ Time and Space complexity is increased.
- ❑ Recursion is generally slower than iteration.



Conclusion



- There are two types of recursion in the C language.
- The first is Direct recursion and Indirect recursion.
- The Direct recursion in C occurs when a function calls itself directly from inside.
- Indirect recursion occurs when a function calls another function, and then that function calls the first function again.
- The function call to itself is a recursive call, and the function will become a recursive function.
- The stack is maintained in the memory to store the recursive calls and all the variables with the value passed in them.