

## #10 Recursion

By Saurabh Shukla | 2017@mysirg.com

### What is recursion?

In C language term recursion is used in the context of function. Functions can be of recursive nature. Recursive function is a function which calls itself. A task which requires performing a job by repeatedly using same steps can be implemented either using loops or using recursion.

- Not every problem can be solved using recursion
- If a problem can be solved using recursion, then it must have an iterative solution too.
- Recursion is easy to implement as compare to iterative solution
- Recursion takes more memory
- Simple code which is easy to understand is always considered as a good code, therefore recursive functions are used.
- You can think of a function which is calling itself, like below:

```
int fun()
{
    ...
    fun();
    ...
}
```

- Without caring about the termination case (better known as base case), recursion will be never ending. To avoid such situation, programmer must think of a situation when function stops calling itself.
- A recursive method solves a problem by calling a copy of itself to work on a smaller problem
- It is important to ensure that the recursion terminates
- Each time the function call itself with a slightly simpler version of the original problem.
- There are several types of recursion
  - Linear Recursion
    - A linear recursive function is a function that only makes a single call to itself each time the function runs (as opposed to one that would call itself multiple times during its execution). The factorial function is a good example of linear recursion.
  - Tail Recursion

- A function call is said to be tail recursive if there is nothing to do after the function returns except return its value
- Binary Recursion
  - Some recursive functions don't just have one call to themselves, they have two (or more). Functions with two recursive calls are referred to as binary recursive functions.
- Exponential Recursion
  - An exponential recursive function is one that, if you were to draw out a representation of all the function calls, would have an exponential number of calls in relation to the size of the data set (exponential meaning if there were  $n$  elements, there would be  $O(a^n)$  function calls where  $a$  is a positive number).
- Indirect recursion
  - A recursive function doesn't necessarily need to call itself. Some recursive functions work in pairs. Function A calls B and B calls A.


### Learn with example

The best way to learn about recursion is to trace a recursive code. For those who are beginners, give time while tracing the code and make memory diagram to understand the separation among the various calls of the same function.

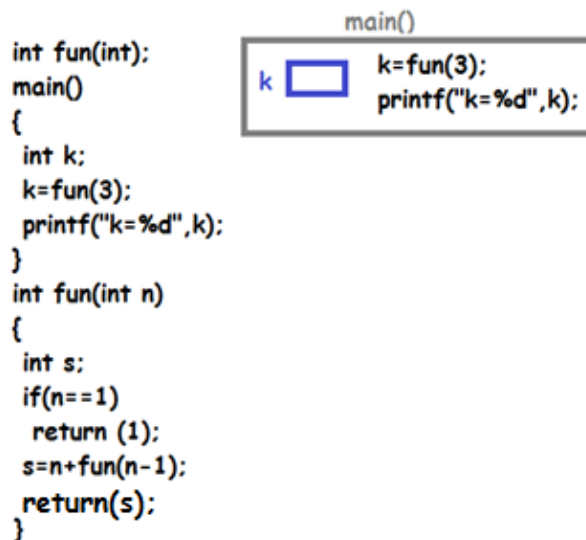
Below is the program which consists of two functions, main() and fun(). Function fun() is the recursive function, as you can see it is calling to itself.

```
int fun(int);
main()
{
    int k;
    k=fun(3);
    printf("k=%d",k);
}
int fun(int n)
{
    int s;
    if(n==1)
        return (1);
    s=n+fun(n-1);
    return(s);
}
```

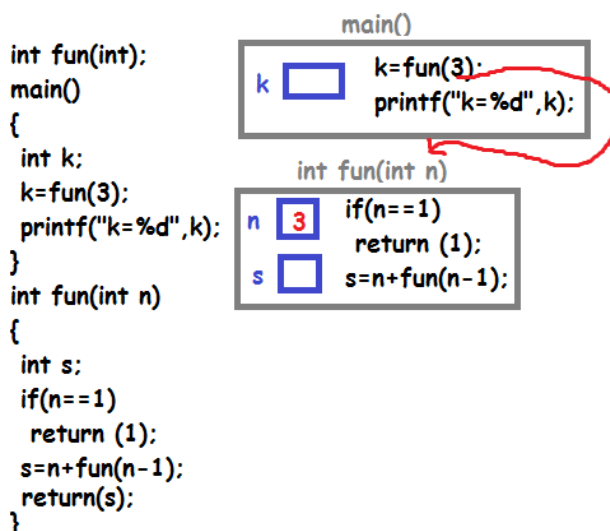
function is calling  
itself



- Execution begins with function main().
- Function occupies memory on its call. Therefore main function occupies space in programs memory. Variable k is getting memory in this section only.



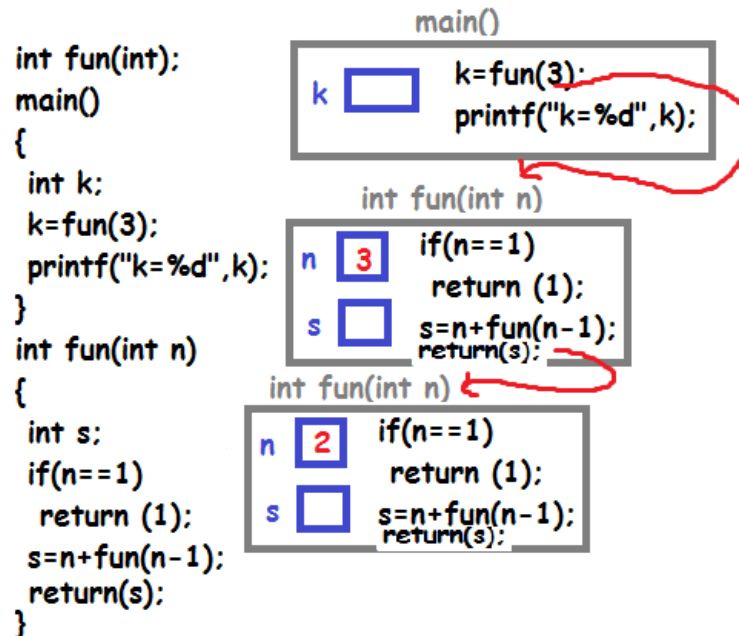
- Function takes space in the memory for its variables and instructions.
- In the first line of main function after declaration of k, function fun is invoked.  
k=fun(3);  
This is called function call by passing value. (value is 3 in the example)
- Function fun has the turn to get arrived in the memory.



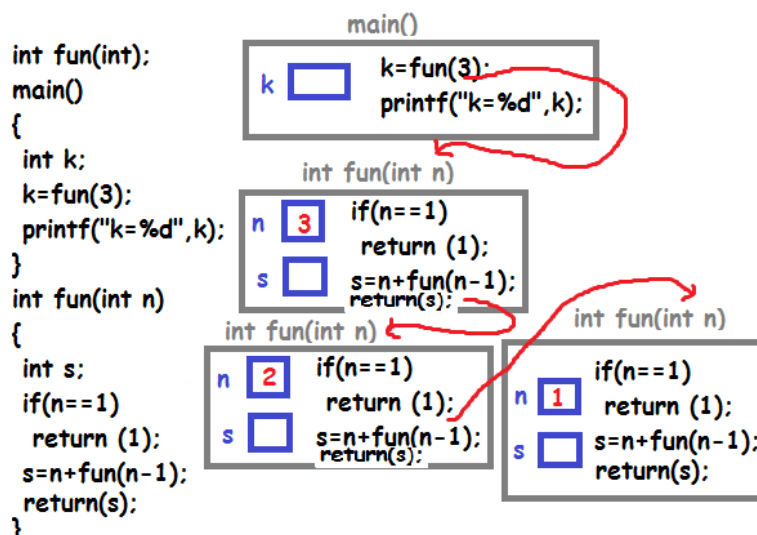
- Variable n of function fun contains 3, as you can see in the above diagram.
- Now execution of fun function begins.  
if(n==1)  
condition of if is evaluated as false, as the value of n is 3. Skipping the return(1) statement and move control to the line

`s=n+fun(n-1);`

function fun is invoked again but by passing a value one less than the value of n.

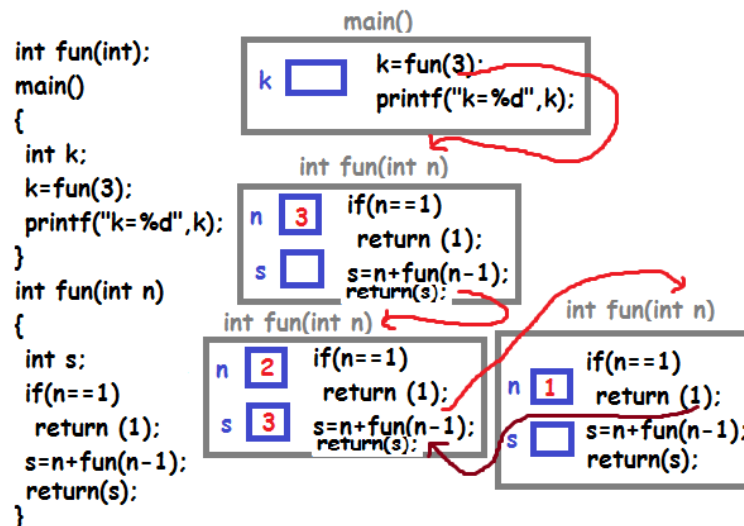


- Notice the separate memory for `fun()` function. Different set of variables are created with possibly different values.
- Execution of second instance of `fun()` function begins. Condition `n==1` is false as the value of `n` is 2. Control skipped the `return(1)` statement and moves to the line `s=n+fun(n-1);`.  
Again `fun()` function is called by passing value one less than the current value of `n`.

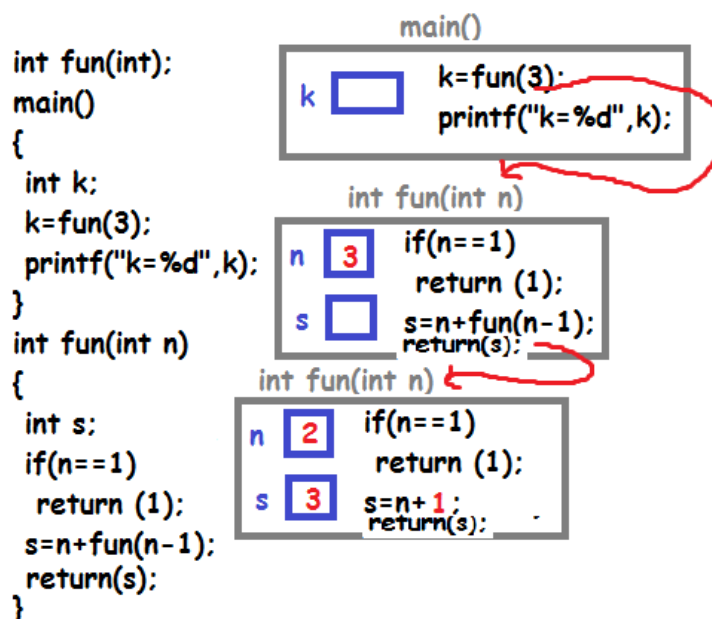


- Third time function `fun()` is getting memory.
- The value of `n` in the third `fun()` function is 1.
- Execution begins from `if(n==1)`, the condition is true for the first time.

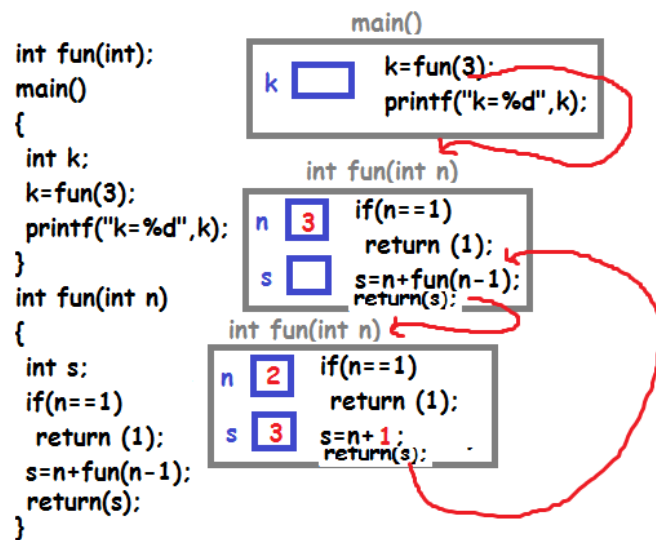
- `return(1)` is executed. You know that the `return` terminates the execution of function and send value back to the calling function at the same place from where the function was invoked.



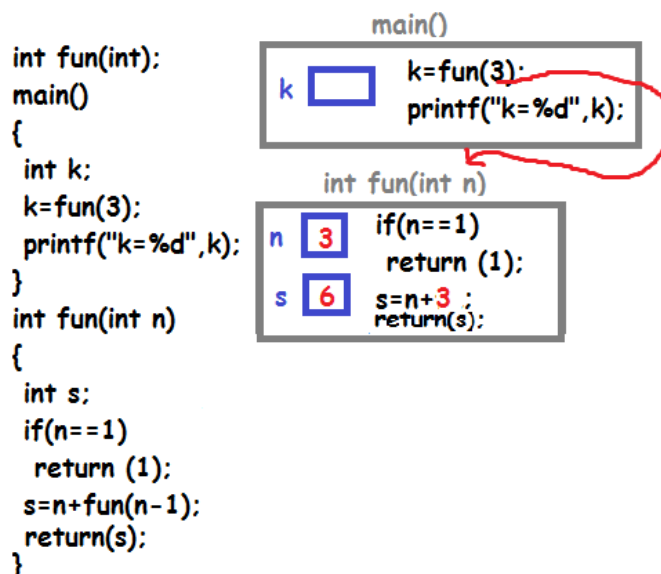
- Once the value is returned back, memory of third `fun()` is released.
- The value of `s` becomes 3



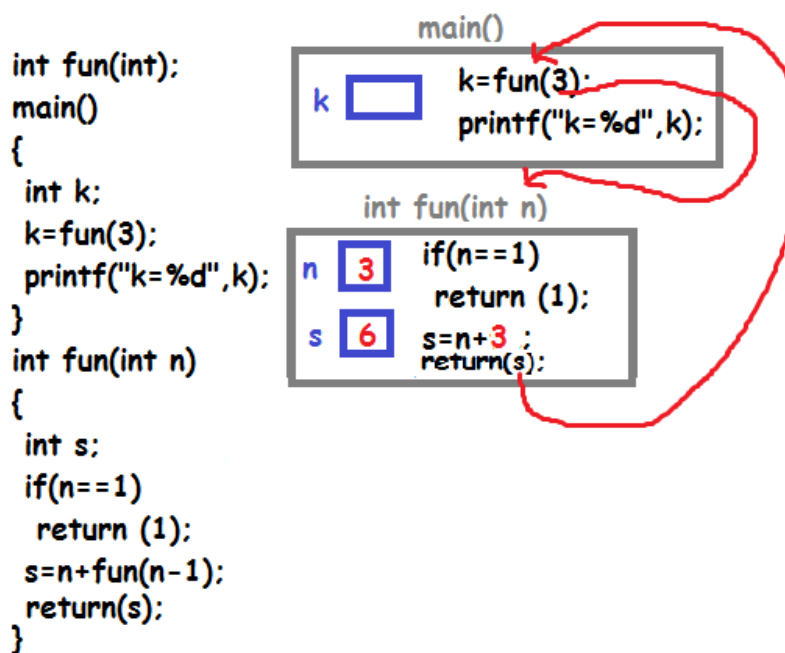
- The value of `s` is now returned back to the calling function.



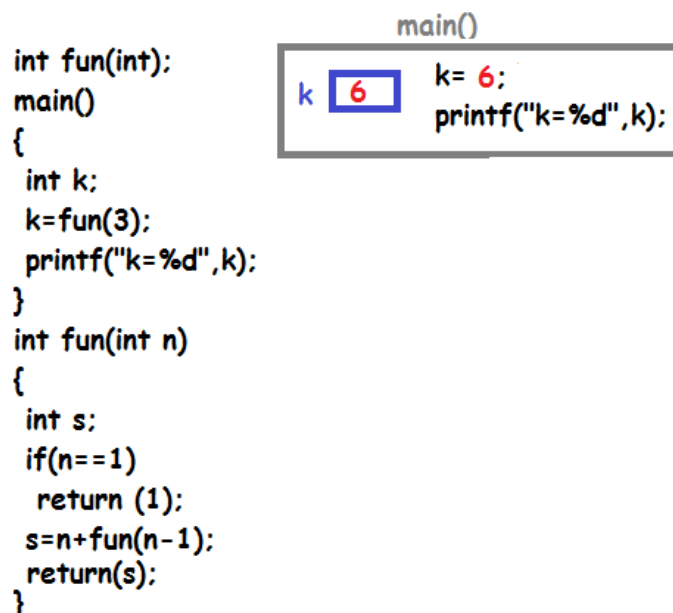
- Once return statement is executed, second `fun()` function is removed from the memory.



- Variable `s` in the first `fun()` function becomes 6.
- The statement `return(s)` is executed and returns the control and value back to the main function.



- Memory of first `fun()` function is released.



- The value of `k` becomes 6 and finally printed on the screen.

### Conclusion

- Recursion happens when function call itself.
- On each call new instance of function's memory is created.
- More call requires more space.

- Termination case or base case is the condition should be used to end the recursion. In the above example  
`if(n==1) return(1);`  
is the base case.
- Two line `s=n+fun(n-1); return (s);` can be merged logically as  
`return(n+fun(n-1));`
- Statement `n+fun(n-1)` is called recursive case.
- Above recursive function `fun()` is a function to calculate sum of first n natural numbers.
- You passed 3 in the function and it gives the sum of first 3 natural numbers (1+2+3).

### References:

#### YouTube video links

- Lecture 10 Recursion in C
  - [https://youtu.be/b\\_L1PHY7j0Q?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW](https://youtu.be/b_L1PHY7j0Q?list=PL7ersPsTyYt2Q-SqZxTA1D-melSfqBRMW)

#### Exercise

1. Write a recursive function to calculate sum of squares of first N natural numbers.
2. Write a recursive function to calculate sum of cubes of first N natural numbers.
3. Write a recursive function to calculate sum of first N even natural numbers.
4. Write a recursive function to calculate sum of first N odd natural numbers.
5. Write a recursive function to calculate factorial of N.
6. Write a recursive function to calculate  $x^y$ .
7. Write a recursive function to convert decimal number into binary number.
8. Write a recursive function to print first N natural numbers
9. Write a recursive function to print first N natural numbers in reverse order.
10. Write a recursive function to print first N even natural numbers
11. Write a recursive function to print first N even natural numbers in reverse order.
12. Write a recursive function to print first N odd natural numbers.
13. Write a recursive function to print first N odd natural numbers in reverse order.
14. Write a recursive function to find  $N^{\text{th}}$  term of Fibonacci series.
15. Write a recursive function to calculate determinant of order N.
16. Write a recursive function to calculate HCF of two numbers.



## #10 Recursion

C Notes Vol-2 by Saurabh Shukla

[www.mysirg.com](http://www.mysirg.com)

---