DATA STRUCTURES
USING "C"

For

BCA Part-II (Session 2018-21) Students

BY

ANANT KUMAR
MCA, M. Phil, M. Tech.
Faculty Member
Department of Computer Science
J. D. Women's College, Patna

Chapter 2 : Recursion

Recursion is the process of defining something in terms of itself. Recursive function is a function that calls itself to solve a problem. Every recursive solution has two major cases:

(i) Base case
(ii) Recursive Case

Base case – A problem to be solved directly without making any further call to the same function.

Recursive case – In this case First, the problem is divided into sub-parts. Second, the function call itself but with sub-parts of the problem obtained in the first step. Third, the result is obtained by combining the solution of simpler sub-parts.

Differences between recursion and iteration

Iteration	Recursion
Iteration explicitly user a repetition structure.	Recursion achieves repetition through repeated function calls.
Iteration terminates when the loop continuation.	Recursion terminates when a base case is recognized.
Iteration keeps modifying the counter until the loop continuation condition fails.	Recursion keeps producing simple versions of the original problem until the base case is reached.
Iteration normally occurs within a loop so the extra memory assigned is omitted.	Recursion causes another copy of the function and hence a considerable memory space's occupied.
It reduces the processor's operating time.	It increases the processor's operating time.

Examples

1. Write a recursive function to find out the factorial of give number.

```
#include <stdio.h>
int factorial (int);
main()
{
        int num, fact;
        printf ("Enter a positive integer value: ");
        scanf ("%d", &num);
        fact = factorial (num);
        printf ("\n Factorial of %d =%d\n", num, fact);
}
```

```
int factorial (int n)
{
        int f;
        if (n ==0 || n==1)
                return (1);
        else
                f= n * factorial (n-1);

        return (f);
}
```

Dry run of above function call –

When the factorial function is first called with, say, N = 5, here is what

happens: FUNCTION:
Does N = 0? No
Function Return Value = 5 * factorial (4)

At this time, the function factorial is called again, with N =

4. FUNCTION:
Does N = 0? No
Function Return Value = 4 * factorial (3)

At this time, the function factorial is called again, with N =

3. FUNCTION:
Does N = 0? No
Function Return Value = 3 * factorial (2)

At this time, the function factorial is called again, with N =

2. FUNCTION:
Does N = 0? No
Function Return Value = 2 * factorial (1)

At this time, the function factorial is called again, with N =

1. FUNCTION:
Does N = 0? No
Function Return Value = 1 * factorial (0)

At this time, the function factorial is called again, with N =

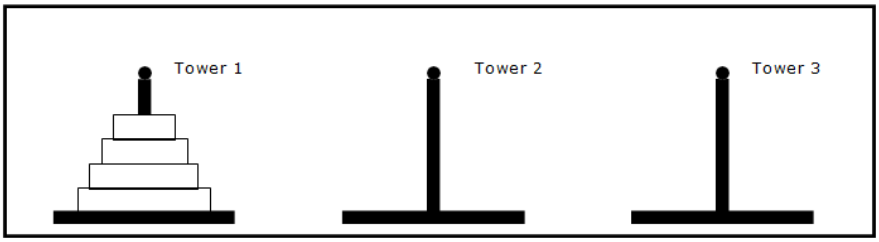
0. FUNCTION:
Does N = 0? Yes
Function Return Value = 1

5! = 5*4! = 5*4*3! = 5*4*3*2! = 5*4*3*2*1! = 5*4*3*2*1*1 =120

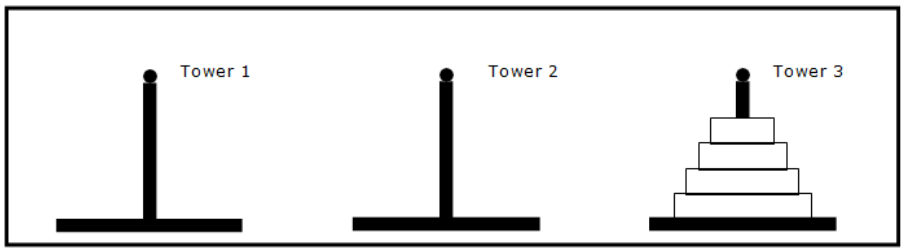
2. Write a recursive function for the Towers of Hanoi.

The rules to be followed in moving the disks from tower 1 tower 3 using tower 2 are as follows:

- Only one disk can be moved at a time.
- Only the top disc on any tower can be moved to any other tower.
- A larger disk cannot be placed on a smaller disk.



Final result is




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

void towersofhanoi (int n, char *a, char *b, char *c);

int cnt=0;

int main (void)
{
        int n;
        printf("Enter number of discs:
        "); scanf("%d",&n);
        towersofhanoi (n, "Tower 1", "Tower 2", "Tower
        3"); getch();
}
```

```
void towersofhanoi (int n, char *a, char *b, char *c)
{
        if (n == 1)
        {

                ++cnt;
                printf ("\n%5d: Move disk 1 from %s to %s", cnt, a, c);
                return;
        }
        else
        {

                towersofhanoi (n-1, a, c, b);
                ++cnt;
                printf ("\n%5d: Move disk %d from %s to %s", cnt, n, a, c);
                towerofhanoi (n-1, b, a, c);
                return;
        }
}
```

Dry run of above function call –

RUN 1:

Enter the number of discs: 3

1: Move disk 1 from tower 1 to tower 3.
2: Move disk 2 from tower 1 to tower 2.
3: Move disk 1 from tower 3 to tower 2.
4: Move disk 3 from tower 1 to tower 3.
5: Move disk 1 from tower 2 to tower 1.
6: Move disk 2 from tower 2 to tower 3.
7: Move disk 1 from tower 1 to tower 3.

RUN 2:

Enter the number of discs: 4

1: Move disk 1 from tower 1 to tower 2.
2: Move disk 2 from tower 1 to tower 3.
3: Move disk 1 from tower 2 to tower 3.
4: Move disk 3 from tower 1 to tower 2.
5: Move disk 1 from tower 3 to tower 1.
6: Move disk 2 from tower 3 to tower 2.
7: Move disk 1 from tower 1 to tower 2.

8: Move disk 4 from tower 1 to tower 3.
9: Move disk 1 from tower 2 to tower 3.
10: Move disk 2 from tower 2 to tower 1.
11: Move disk 1 from tower 3 to tower 1.
12: Move disk 3 from tower 2 to tower 3.
13: Move disk 1 from tower 1 to tower 2.
14: Move disk 2 from tower 1 to tower 3.
15: Move disk 1 from tower 2 to tower 3.

3. Write a recursive function to print Fibonacci series up to n terms.

0 1 1 2 3 5 8 13 21 . . . . . .

```
#include <stdio.h>
int fibo(int);
void main()
{
        int i,num;
        clrscr ();
        printf("Enter terms");
        scanf("%d", &num);
        for(i=0;i<=num;i++)
          printf ("%d ", fibo(i));
}

int fibo(int n)
{
        int x;
        if (n==0 | | n==1)
                return n;
        x=fibo(n-1) + fibo(n-2);
        return (x);
}
```

Dry run of above function call -

Fib (n) = n if n = 0 or n = 1
Fib (n) = fib (n-1) + fib (n-2) for n >=2

fib(5) = fib(4) + fib(3)

fib(3) + fib(2) + fib(3)

fib(2) + fib(1) + fib(2) + fib(3)

fib(1) + fib(0) + fib(1) + fib(2) + fib(3)

1 + 0 + 1 + fib(1) + fib(0) + fib(3)

1 + 0 + 1 + 1 + 0 + fib(2) + fib(1)

1 + 0 + 1 + 1 + 0 + fib(1) + fib(0) + fib(1)

1 + 0 + 1 + 1 + 0 + 1 + 0 + 1 = 5