Recursive Functions

A function that calls itself is known as recursive function and the process of a function calling itself is known as "recursion".

Recursion can be quite elegant; requiring fewer variables which make the program clean. Recursion can be used to replace complex nesting code by dividing the problem into same problem of its sub-type. On other hand, it is hard to think the logic of a recursive function. It may be also difficult to debug the code containing recursion.

Example

A function to factorial of 'n' is follows

```
1  int fact(int n) {
2    if (n <= 1) {
3      return 1;
4    } else {
5      return n * fact (n - 1);
6    }
7  }
```

Assume n is 3; since it is not 1, the statement `fact = n * fact(n - 1);` will be executed with n = 3. The sequence of operations can be summarized as follows:

```
return  3 * fact(2)
return  3 * 2 * fact(1)
return  3 * 2 * 1
```

Every recursive function must have a terminating condition otherwise it may lead to infinite calls. In the above code, the termination condition occurs when n = 1 or n = 0 (1! = 1 and 0! = 1).

Attributes of recursive function

1. A recursive function is a function which calls itself.
2. The recursive program may be slower because of stack overheads.
3. The recursive program may consume more memory.

4. A recursive function must have recursive conditions, terminating conditions, and recursive expressions.

Example

This program prints the required term in the Fibonacci series using recursion. In the Fibonacci series, each term is the sum of the two preceding terms. The first few terms are: 0, 1, 1, 2, 3, 5, 8, ...

```
#include <stdio.h>
int fibo(int);
int main() {
    int num;
    int result;
    printf("Which fibonacci number is wanted? ");
    scanf("%d", &num);
    if (num > 0) {
        result = fibo(num);
        printf("The %d number in fibonacci series is %d\n", num, result);
    }
    return 0;
}

int fibo(int num) {
    if (num <= 1) {
        return 1;
    } else {
        return(fibo(num - 1) + fibo(num - 2));
    }
}
```



```
$
$ c99 Program-09-1.c
$ ./a.out
Which fibonacci number is wanted? 12
The 12 number in fibonacci series is 233
$ ./a.out
Which fibonacci number is wanted? 7
The 7 number in fibonacci series is 21
$
```

Figure 1: Fibonacci Sequence

Summary

- Use recursion for **clarity**, and (sometimes) for a reduction in the time needed to write and debug code, not for space savings or speed of execution.
- Remember that every recursive method must have a **base case**
- Also remember that every recursive method must make progress towards its base case.
- Sometimes a recursive method has more to do following a recursive call. It gets done only after the recursive call (and all calls it makes) finishes.
- Recursion is often simple and elegant, can be efficient, and tends to be under-utilized. Consider using recursion more!