Recursion

Data Structures: A Programming Approach with C, PHI,
Dharmender Singh Kushwaha & A.K.Misra

Introduction

- Recursion means applying a function as a part of the definition of that same function.

- For example, the natural numbers themselves are usually defined recursively.

- 0 is a natural number.
- If n is a natural number then s(n) (i.e. n+1) is a natural number, where s is the "successor function".

Factorial Function

- Factorial function is defined as being the product of all the numbers up to and including the argument.

- The other way to express this is that the factorial of N is equal to N times the factorial of (N-1).

- The definition of factorial function is as follows:
- 0! = 1
- for all n > 0, n! = n * (n-1)!

Factorial Function

- Thus

- $1! = 1$
- $2! = 1 \times 2 = 2$

- $3! = 1 \times 2 \times 3 = 2! \times 3 = 6$
- $N! = 1 \times 2 \times 3 \times .... (N-2) \times (N-1) \times N = (N-1)! \times N$

Factorial function

- Definition in c language

```
int factorial(n)
        {
                if (n == 1)
                {
                        return 1;
                }
                else
                {
                        return (n * factorial(n-1));
                 }
          }
```

Recursive Function Calls Definition

- A method is **recursive** if it can call itself; either directly or indirectly as follows

```
Void f()
  {
  f()
  }
```

```
void f()
        {
        g();
        }
void g()
        {
        f();
        }
```

How recursive function call works?

Let's take an example

```
void printInt( int k )
    {
    if (k == 1)
            {
      return;
    }
   printf(“%d”,k);
   printInt( k - 1 );
   printf("all done");
    }
```

Recursion Example

- For a value of k=4

Initial stage

```
Input :    K=4
void printInt(int k )
{
 if (k == 1) {
   return;
  }
 printf("%d",k);
 printInt( 3 );
 printf("all done");
}
```

Output = 4

function recursively called for k=3

```
Input :    K=3
void printInt(int k )
 {
 if (k == 1) {
  return;
    }
 printf("%d",k);
 printInt( 2 );
 printf("all done");
}
```

Output = 3

Recursion Example

function recursively called for k=2

```
Input :                K=2
void printInt(int k )
{
if (k == 1) {
return;
 }
printf(“%d”,k);
printInt( 1 );
printf("all done");}
Output = 2
```

function terminates as boundary condition is reached

```
Input :                K=1
void printInt(int k )
{
if (k == 1) {
return;
 }
printf(“%d”,k);
printInt( k-1 );
printf("all done");}
Output =
```

This code is not Reached as the function returns 1after finding k to be 1

Rules for Recursive Programs

- Every recursive method must have a condition (base case) under which no recursive call is made. This is to prevent infinite recursion.

```
void Display( int n ) {
if (n<0)
return;
printf("%d", n);
Display( n + 1 );
```

- This function does have a condition, but the call *Display(4)* will still cause an infinite recursion because of increased value of (n+1):
- Hence, the second rule.

Rules for Recursive Programs

- Every recursive method must make progress toward the base case to prevent infinite recursion.

Limitations

- Recursion does not make your code **faster**
- Recursion does not usually use **less memory**.
- Rather, it simplifies the code.

How Recursion Works

- At runtime, a stack of activation records (ARs) is maintained, one AR for each active function, where "active" means the function that has been called but has not yet returned.
- Each AR includes space for:
  - the function 's parameters,
  - the function 's local variables,
  - the return address in the code to start executing after the function returns.

How Recursion Works

- When a function is called, its AR is pushed onto the stack.
- The return address in that AR is the place in the code just after the call.
- When a function is about to return:
  - the return address in its AR is saved,
  - its AR is popped from the stack, and
  - control is transferred to the place in the code referred to by the return address.

Eg.: Iterative version of factorial

```
int factorial (int N)
{
    if (N == 0) return 1;
    int tmp = N,j;
    for (k=N-1; k>=1; k--)
        {
        tmp = tmp*k;
        }
    return (tmp);
}
```

Recursive version

```
int factorial (int N)
{
    if (N == 0) return 1;
    else return (N*factorial(N-1));
}
```

Recursion Types

Linear Recursion

- A linear recursive function is a function that only makes a single call to itself each time the function executes.

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

Recursion Types

Tail Recursion

- A recursive procedure where the recursive call is the last action to be taken by the function.

```
int gcd(int m, int n)
{
        int r;
        if (m < n) return gcd(n,m);
        r = m%n;
        if (r == 0) return(n);
        else return(gcd(n,r));
  }
```

Recursion Types

Binary Recursion

- A recursive function which calls itself twice during the course of its execution.

```
int choose(int n, int k)
{
        if (k == 0 || n == k) return(1);
        else return(choose(n-1,k) + choose(n-1,k-1));
}
```

Recursion Types

Exponential Recursion

- Recursion where more than one call is made to the function from within itself.

Recursion Types

Nested Recursion

- One of the arguments to the recursive function is the recursive function itself.

- These functions tend to grow extremely fast.

Recursion Types

Mutual Recursion

- A recursive function doesn't necessarily need to call itself.

- Example : Function A calls function B which calls function C which in turn calls function A.

Difference between Recursion & Iterations

- Recursive functions are partially defined by itself and consists of some simple case with a known answer.
- Example: Fibonacci number sequence, factorial function, quick sort and more.
- Some of the algorithms/functions can be represented in iterative way and some may not.

Difference between Recursion & Iterations

- Iterative functions are loop based imperative repetition of a process in contrast to recursion which has more declarative approach.
- Recursion is slower than iteration due to the overhead of maintaining the stack.
- Recursion uses more memory for the stack as compared to iteration.

Summary

- Recursion is the method of repeating process in a self-identical way.
- To realize recursion, one must do the distinction between a method and the running of that method.
- A method is a set of steps that are to be taken based on a set of rules.
- A method that goes through recursion is said to be 'recursive'. Conversely, a result that is the effect of a recursive method is said to be recursive.

Summary

- In recursion, function calls are executed using stack because stack follows the Last in First out (LIFO) technique.
- The recursive version of program causes an activation record to be pushed onto the runtime stack for every call.