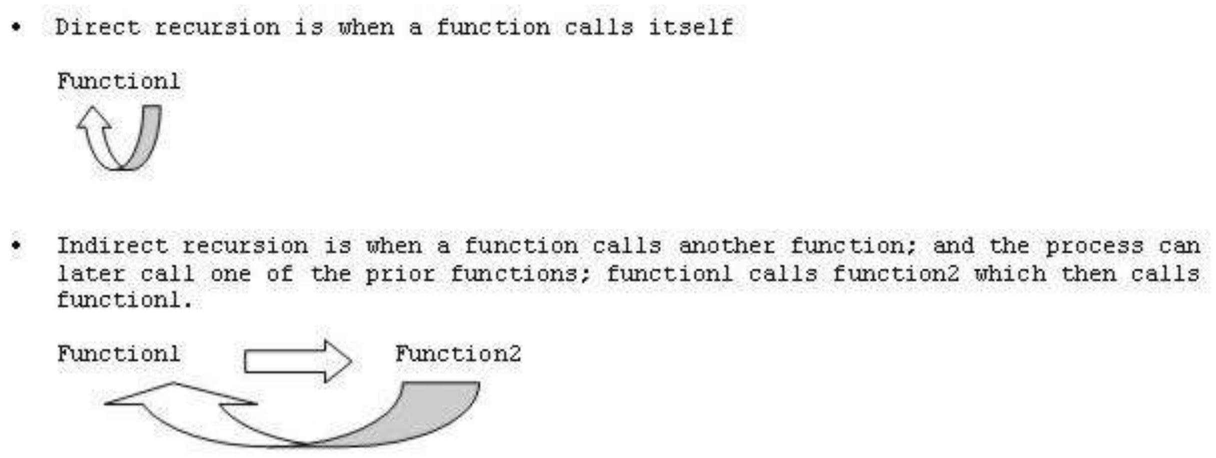
Recursion

Recursion is when a function can call itself; either directly or indirectly.



- Direct recursion is when a function calls itself
- Indirect recursion is when a function calls another function; and the process can later call one of the prior functions; function1 calls function2 which then calls function1.

Recursion must have

- A terminating point and
- The algorithm of the recursion must move to that point.

The following is an example of a factorial function.

5! is 5*4*3*2*1
5! can also be thought of as 5 * 4!
4! can be thought of as 4 * 3! And so forth

```
using namespace std;
#include <iostream>

int factorial(int n);

int main ()

  { int n =5;
    cout << factorial(5);                         // Initial value of 5!
    return 0;
  }
//-----------------------------------------------------------------------
```

```
int factorial(int n)
{ if (n==1)                                   // The bottom or terminating point
      return (1);
     else return(factorial(n-1)*n);       //Recursion occurs here !! The function calls itself

                                                    //The algorithm now multiplies n*(n-1)!
}
```

The next section shows how the stack works for a factorial call for 3!.

Memory location	Value	identifier
4		
3		
2	3	n
1	Unknown so far but will be 3 * 2!	factorial

Pushes onto stack like a cafeteria stack of trays
Current Activation (call #1)

n	Location 2	Value is 3
factorial	Location 1	Unknown so far but will be 3 * 2!

Memory location	Value	identifier
4	2	n
3	Unknown so far but will be 2 * 1!	factorial
2	3	n
1	Unknown so far but will be 3 * 2!	factorial

Pushes onto stack like a cafeteria stack of trays
Current Activation (call #2)

n	Location 4	Value is 2
factorial	Location 3	Unknown so far but will be 2 * 1!

Memory location	Value	identifier
6	1	n
5	Returns a value of 1	factorial
4	2	n
3	Unknown so far but will be 2 * 1!	factorial
2	3	n
1	Unknown so far but will be 3 * 2!	factorial

Pushes onto stack like a cafeteria stack of trays
Current Activation (call #3)

n	Location 6	Value is 1
factorial	Location 5	Returns a value of 1

Memory location	Value	identifier
6	1	n
5	Returns a value of 1	factorial
4	2	n
3	Returns a value of 2	factorial
2	3	n
1	Unknown so far but will be 3 * 2!	factorial

Pops off the stack like a cafeteria stack of trays
Returns to Prior Activation (call #2)

n	Location 4	Value is 2

factorial	Location 3	Returns a value of 1*2!

Notice that values on the stack are not deleted; this fact is a major reason that you must initialize variables!!!

Memory location	Value	identifier
6	1	n
5	Returns a value of 1	factorial
4	2	n
3	Returns a value of 2	factorial
2	3	n
1	Returns a value of 6 (3 * 2!)	factorial

Pops off the stack like a cafeteria stack of trays
Returns to Prior Activation (call #1)

n	Location 2	Value is 3

factorial	Location 1	Returns a value of 3*2!