RECURSION

NATIONAL UNIVERSITY OF TECHNOLOGY (NUTECH)

DR. SAMAN RIAZ

LECTURE # 6

INTRODUCTION TO RECURSION

- A recursive function is one that calls itself.

```
void Message(void)
{
    cout << "This is a recursive function.\n";
    Message();
}
```

The function above displays the string "This is a recursive function.\n", and then calls itself.

Can you see a problem with the function?

RECURSION

- The function is like an infinite loop because there is no code to stop it from repeating.
- Like a loop, a recursive function must have some algorithm to control the number of times it repeats.

RECURSION

- Like a loop, a recursive function must have some algorithm to control the number of times it repeats. Shown below is a modification of the `message` function. It passes an integer argument, which holds the number of times the function is to call itself.

```
void Message(int times)
{
    if (times > 0)
    {
        cout << "This is a recursive function.\n";
        Message(times - 1);
    }
    return;
}
```

RECURSION

- The function contains an `if/else` statement that controls the repetition.
- As long as the `times` argument is greater than zero, it will display the message and call itself again. Each time it calls itself, it passes `times - 1` as the argument.

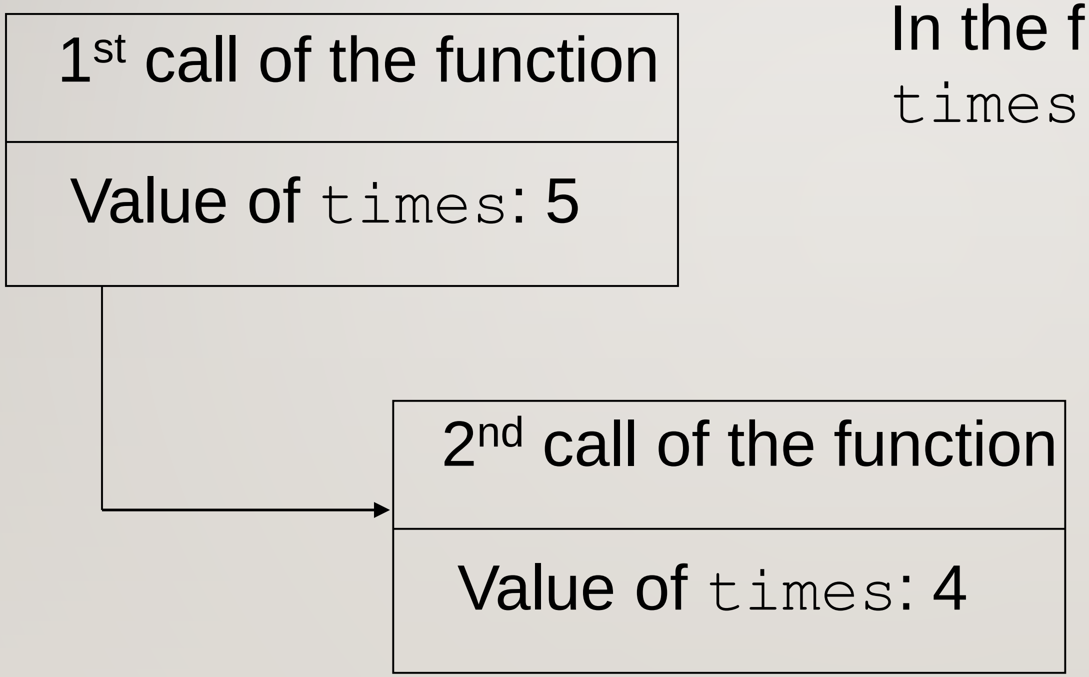
RECURSION

- For example, let's say a program calls the function with the following statement:

```
Message(5);
```

The argument, 5, will cause the function to call itself 5 times. The first time the function is called, the if statement will display the message and then call itself with 4 as the argument.

Each time the function is called, a new instance of the times parameter is created.



1st call of the function

Value of `times`: 5

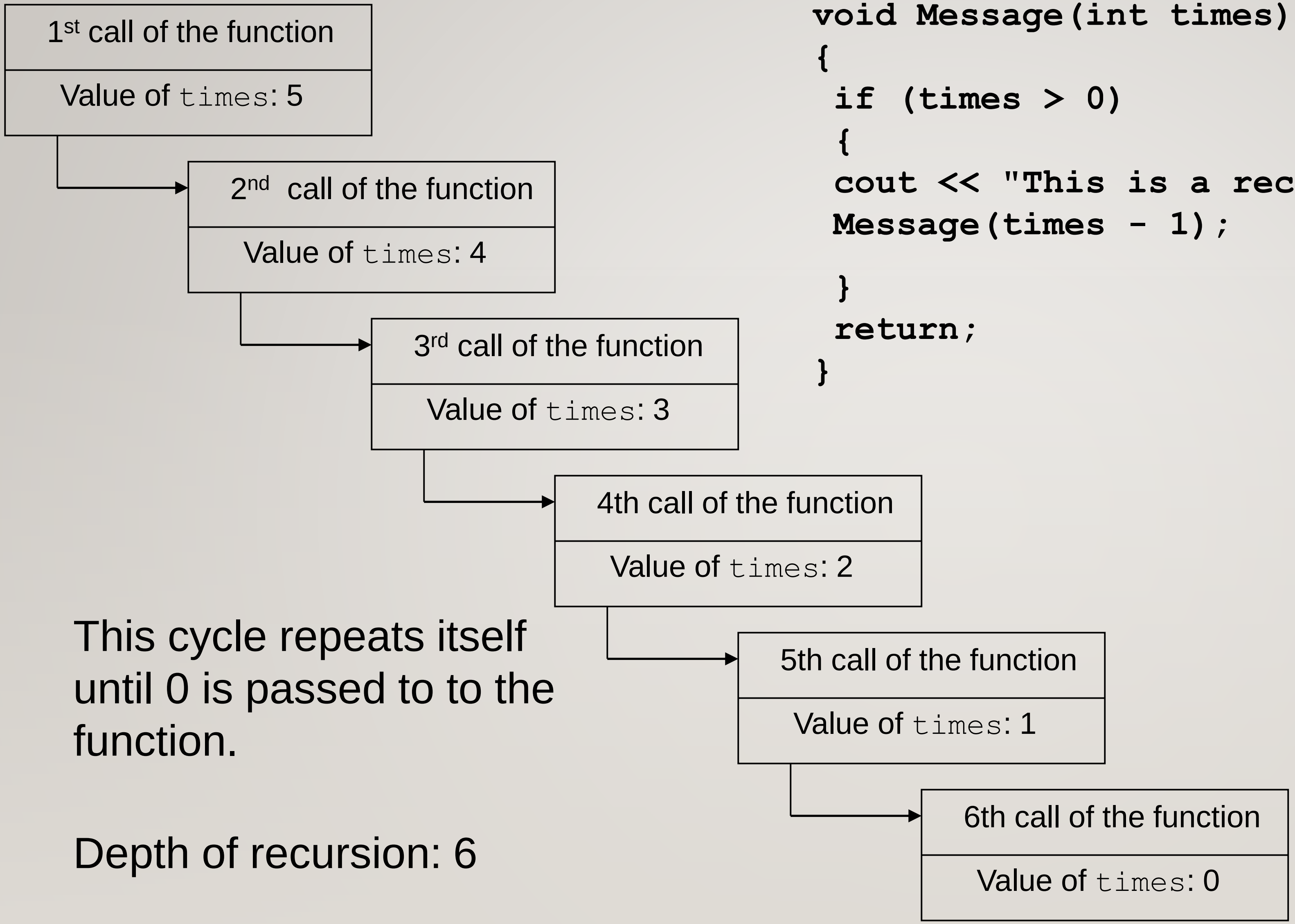
2nd call of the function

Value of `times`: 4

In the first call to the function, `times` is set to 5.

When the function calls itself, a new instance of times is created with the value 4.

```
void Message(int times)
{
 if (times > 0)
 {
 cout << "This is a recursive function.\n";
 Message(times - 1);

 }
 return;
}
```



1st call of the function

Value of `times`: 5

2nd call of the function

Value of `times`: 4

3rd call of the function

Value of `times`: 3

4th call of the function

Value of `times`: 2

5th call of the function

Value of `times`: 1

6th call of the function

Value of `times`: 0

This cycle repeats itself until 0 is passed to to the function.

Depth of recursion: 6

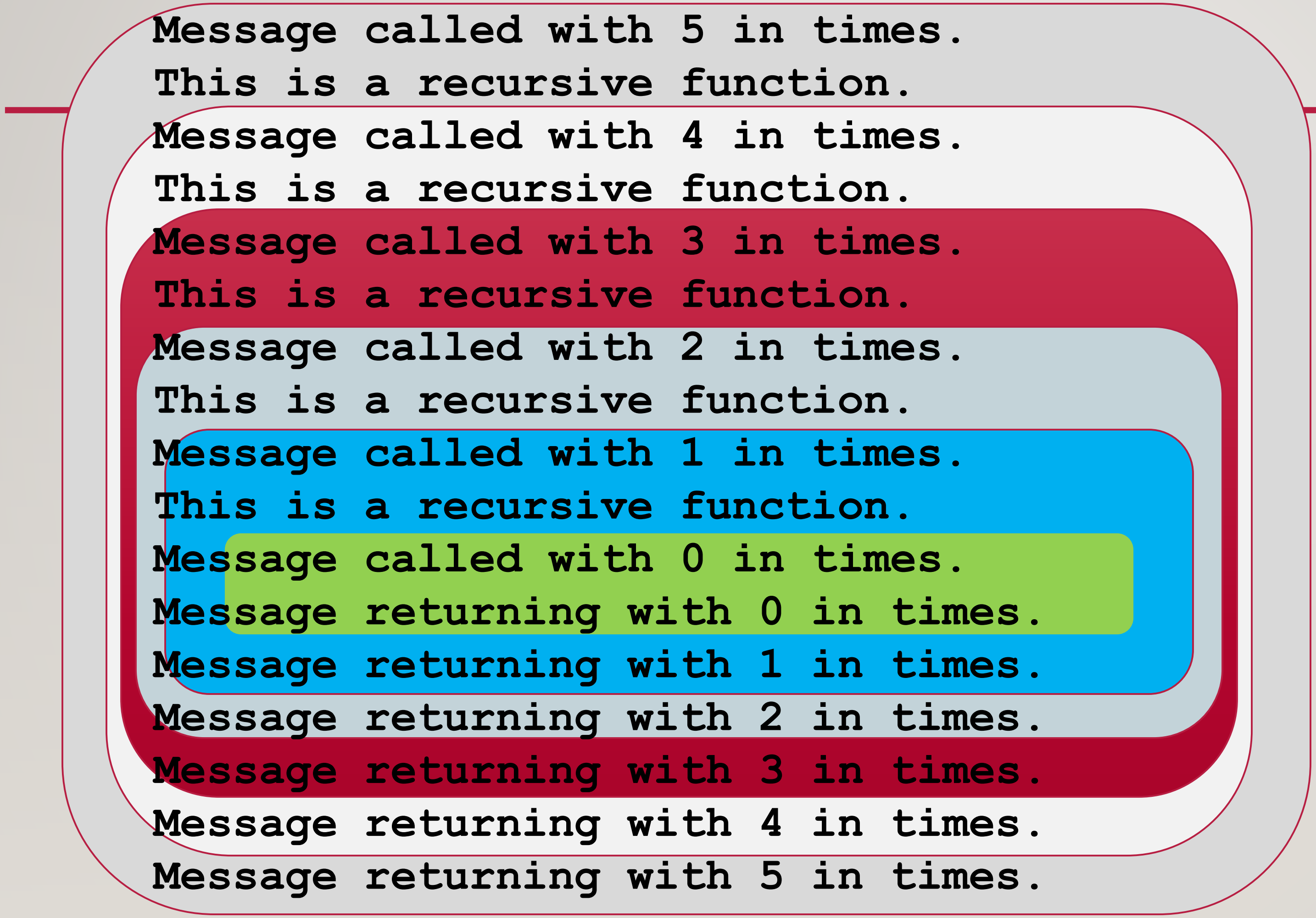
```
//***********************************************************
// Definition of function Message. If the value in times is *
// greater than 0, the message is displayed and the         *
// function is recursively called with the argument         *
// times - 1.                                               *
//***********************************************************

void Message(int times)
{
   if (times > 0)
   {
      cout << "This is a recursive function.\n";
      Message(times - 1);
   }
   return;
}
```

Program Output

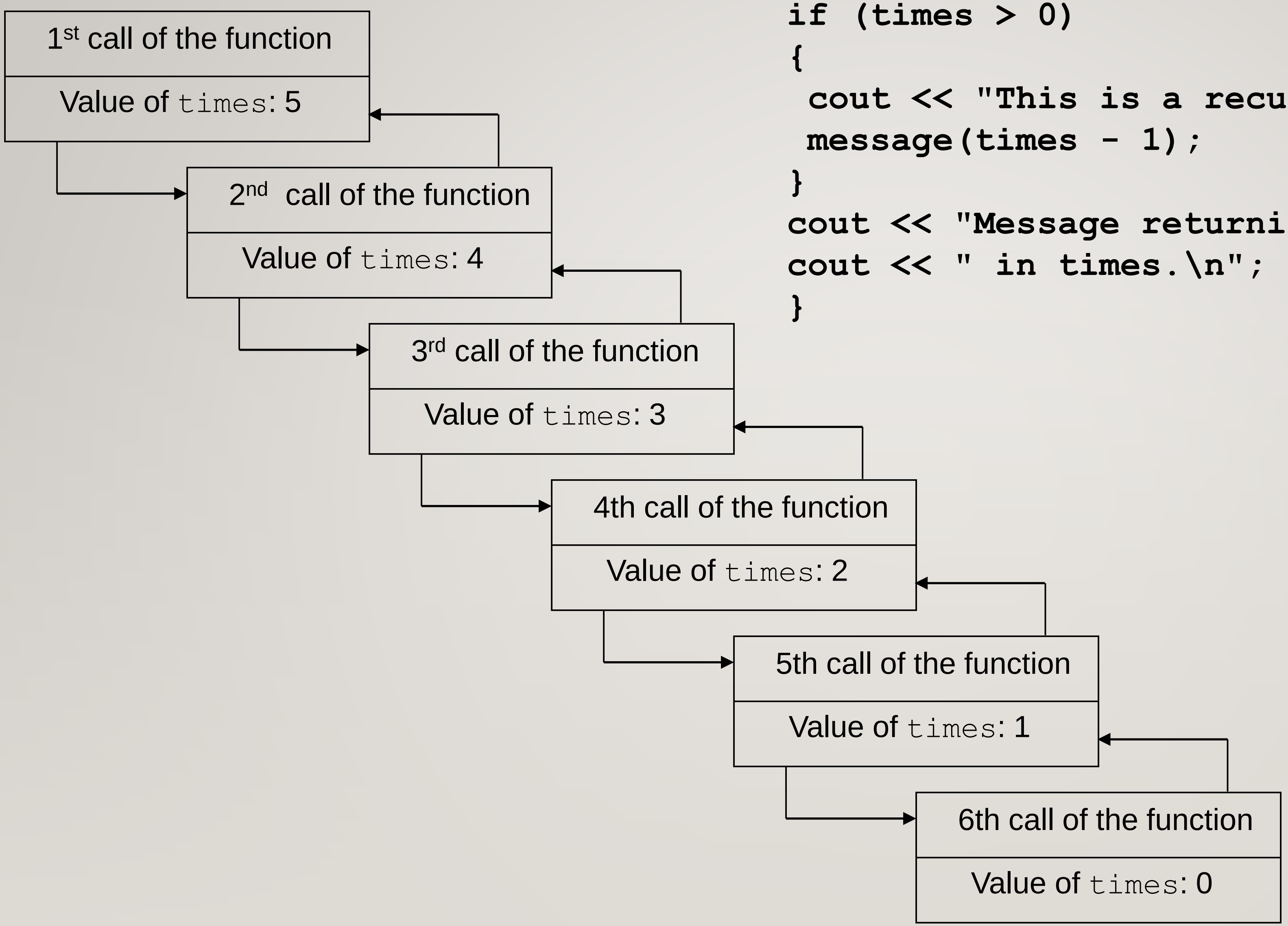
```
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
```

```
//***********************************************************
// Definition of function Message. If the value in times is *
// greater than 0, the message is displayed and the         *
// function is recursively called with the argument         *
// times - 1.                                               *
//***********************************************************

void Message(int times)
{
    cout << "Message called with " << times << " in times.\n";
    if (times > 0)
    {
        cout << "This is a recursive function.\n";
        Message(times - 1);
    }
    cout << "Message returning with " << times;
    cout << " in times.\n";
}
```

Program Output



```
Message called with 5 in times.
This is a recursive function.
Message called with 4 in times.
This is a recursive function.
Message called with 3 in times.
This is a recursive function.
Message called with 2 in times.
This is a recursive function.
Message called with 1 in times.
This is a recursive function.
Message called with 0 in times.
Message returning with 0 in times.
Message returning with 1 in times.
Message returning with 2 in times.
Message returning with 3 in times.
Message returning with 4 in times.
Message returning with 5 in times.
```



1st call of the function

Value of `times`: 5

2nd call of the function

Value of `times`: 4

3rd call of the function

Value of `times`: 3

4th call of the function

Value of `times`: 2

5th call of the function

Value of `times`: 1

6th call of the function

Value of `times`: 0

```
…
if (times > 0)
{
 cout << "This is a recursive function.\n";
 message(times - 1);
}
cout << "Message returning with " << times;
cout << " in times.\n";
}
```

RECURSION

- The role of recursive functions in programming is to break complex problems down to a solvable problem.
- The solvable problem is known as the *base case*.
- A recursive function is designed to terminate when it reaches its base case.

RECURSION

To build *all* recursive functions:

1. **Define the base case(s)**
2. **Define the recursive case(s)**
   a) **Divide the problem into smaller sub-problems**
   b) **Solve the sub-problems**
   c) **Combine results to get answer**

Sub-problems solved as a recursive call to the same function

RECURSION

Note:

the sub-problems must be "smaller" than the original problem otherwise the recursion never terminates.

```
-- loop function
loop x = 1 + loop x
```

RECURSION

Trace:

loop 5

→ 1 + loop 5

→ 1 + loop 5

→ 1 + loop 5

→ …

infinite loop – no termination

In the rest of this lecture we would just look at different examples of recursion

and

some comparison between recursive and iterative functions

RECURSION

Consider the functions for taking the second, third and fourth powers of a given Float:

```
-- 2nd power function
square x = x*x
```

```
-- 3rd power function
cube x = x*x*x
```

RECURSION

```
-- 4th power function
fourthPower x = x*x*x*x
```

1. How would you compute the *10th power* ?
2. What about x^n ($n \geq 0$) ?

RECURSION

$x^4 = x*x*x*x = x*(x*x*x) = x*x^3$

$x^5 = x*x*x*x*x = x*(x*x*x*x) = x*x^4$

$x^6 = x*x*x*x*x*x = x*(x*x*x*x*x) = x*x^5$

In general

$$\boxed{x^n = x*x^{n-1}}$$

RECURSION

Unfortunately, this function is still not quite complete.

2^3

→ $2 * 2^{3-1} = 2 * 2^2$

→ $2 * 2 * 2^1$

→ $2 * 2 * 2 * 2^0$

→ $2 * 2 * 2 * 1 * 2^{-1}$

→ …

When does it stop ?

RECURSION

Remember that power should only be defined for *$n \geq 0$.*

Therefore we need to stop at

power < 0

This is called the *base case*.

RECURSION

New power function:

```
power x n
    | n == 0          = 1
    | otherwise       = x * power x (n-1)
```

The function definition is said to be *recursive*, since it calls itself.

THE RECURSIVE FACTORIAL FUNCTION

In mathematics, the notation n! represents the factorial of the number n. The factorial of a number is defined as:

```
n! = 1 * 2 * 3 * ... * n   if n > 0
   1           if n = 0
```

THE RECURSIVE FACTORIAL FUNCTION

Another way of defining the factorial of a number, using recursion, is:

```
Factorial(n) = n * Factorial(n - 1)      if n > 0
       1              if n = 0
```

The following C++ function implements the recursive definition shown above:

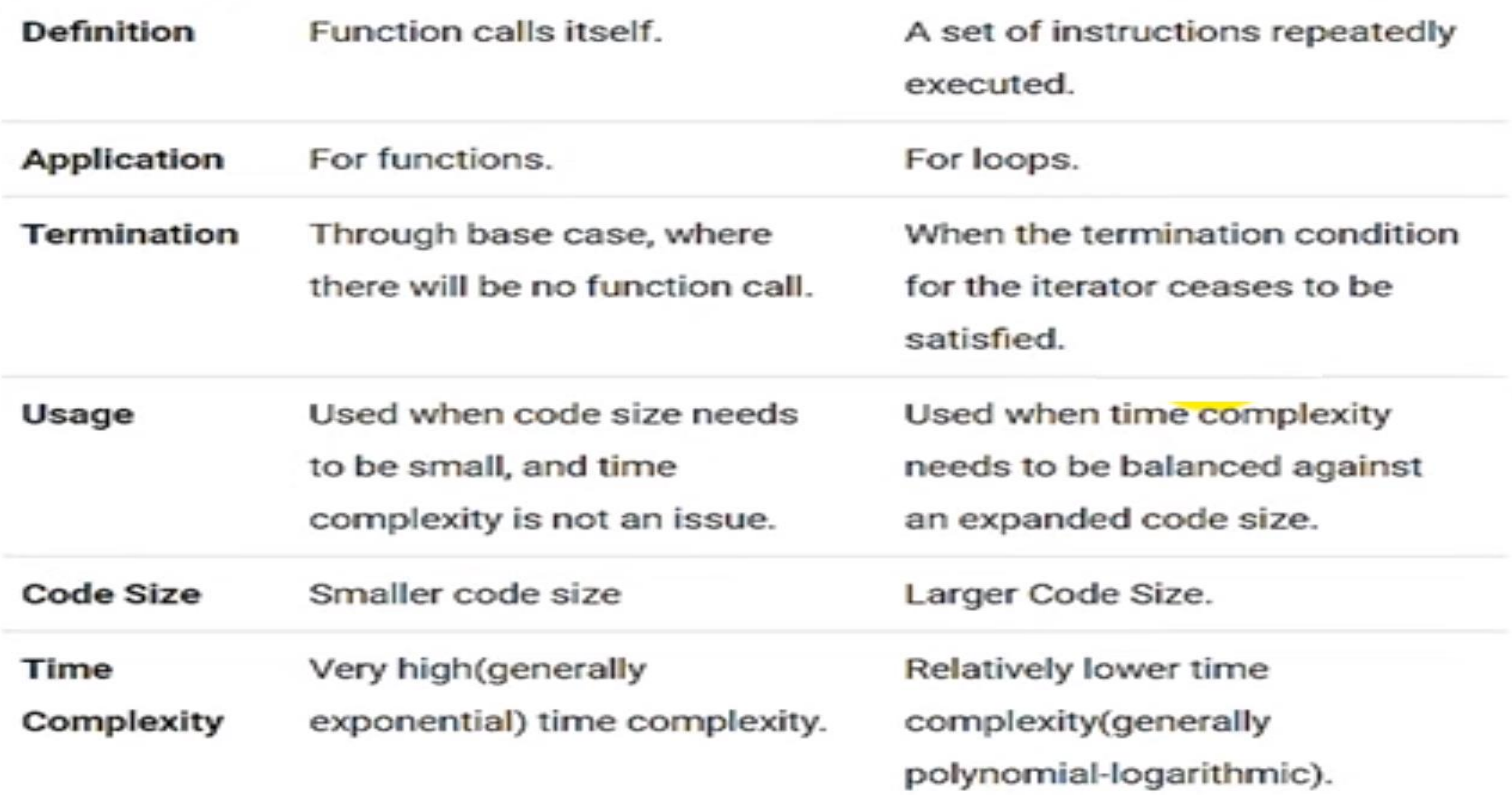
```
int factorial(int num)
{
  if (num > 0)
     return num * factorial(num - 1); else
     return 1;
}
```

```
// This program demonstrates a recursive function to
// calculate the factorial of a number.
#include <iostream>

// Function prototype
int factorial(int);

void main(void)
{
   int number;

   cout << "Enter an integer value and I will display\n";
   cout << "its factorial: ";
   cin >> number;
   cout << "The factorial of " << number << " is ";
   cout << factorial(number) << endl;
}
```

```
//*******************************************************
********
// Definition of factorial. A recursive function to
calculate  *
// the factorial of the parameter, num.
*
//*******************************************************
********

int factorial(int num)
{
   if (num > 0)
      return num * factorial(num - 1);
   else
      return 1;
}
```

Program Output with Example Input

```
Enter an integer value and I will display
its factorial: 4
The factorial of 4 is 24
```



	RECURSIVE	ITERATIVE
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Time Complexity	Very high(generally exponential) time complexity.	Relatively lower time complexity(generally polynomial-logarithmic).

CREATING A SUM FUNCTION

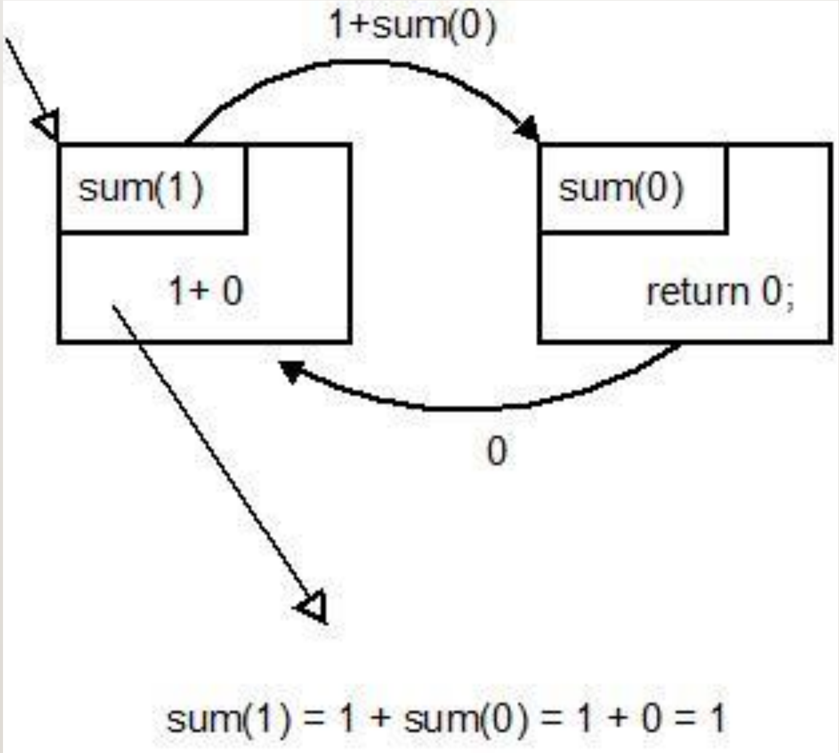
- sum(10) = 10+9+…2+1 = 55

CREATING A SUM FUNCTION (ITERATIVE)

```
//Our initial total is zero
int total = 0;

//We want the sum from 1 + 2 + ... + 9 + 10
int n = 10;

//The following for loop will calculate the summation from 1 - n
for ( int i = 1; i <= n; i++ ) {
    total = total + i;
}
```

CREATING A SUM FUNCTION (RECURSIVE)

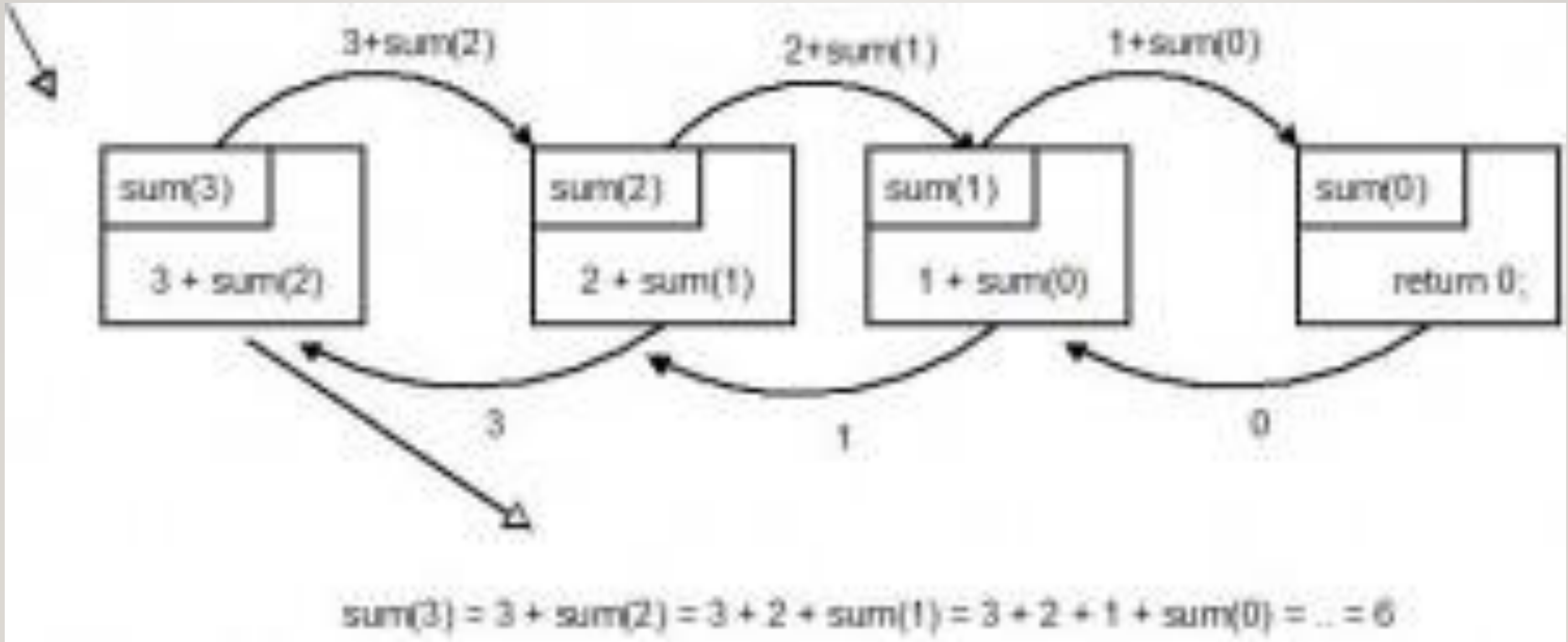
```
int sum(int n) {


    //Return 0 when n is 0

    if ( n <= 0 ) return 0;

}
```



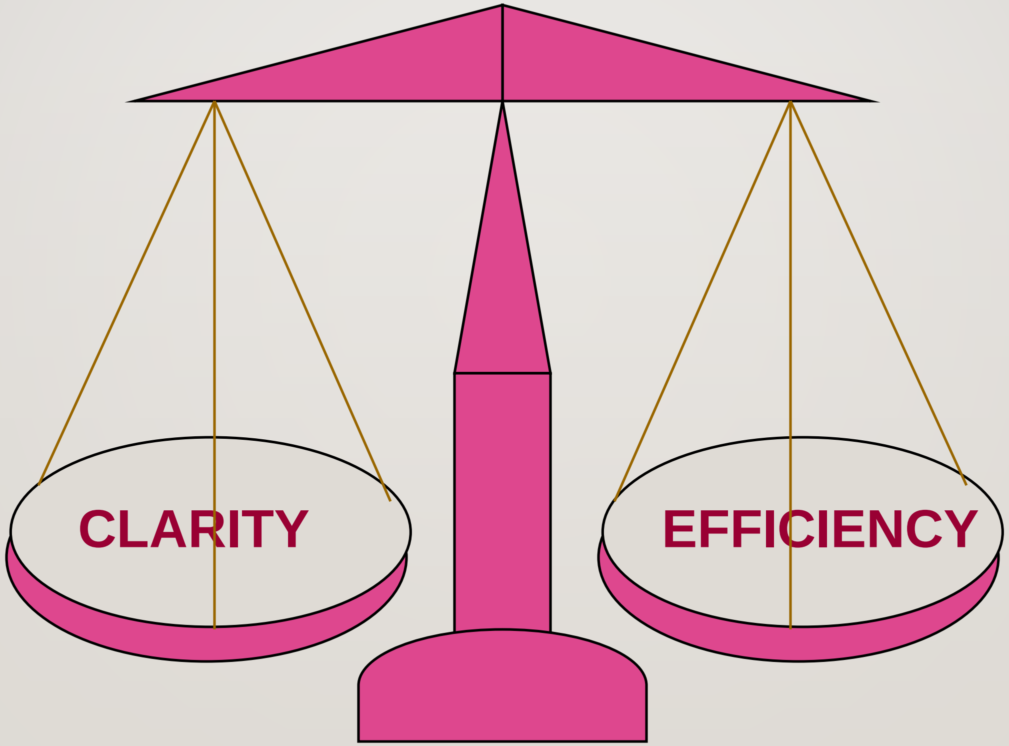
CREATING A SUM FUNCTION (RECURSIVE)

```
int sum(int n) {

    //Return 0 when n is 0
    if ( n <= 0 ) return 0;
    else
        return n + sum(n-1);

}
```

CREATING A SUM FUNCTION (RECURSIVE)



RECURSION OR ITERATION?


CLARITY
EFFICIENCY