


**University of Mosul**

**College of Computer Sciences and Mathematics**

**Department of Artificial Intelligence**

**Algorithms and Structured Programming (2)**

**First stage**

# Lecture 3

ا.م. بيداء سليمان بهنام

**Assist.Prof.Baydaa Sulaiman Bahnam**

# Lecture 3 Outline

1. Recursive Function in C++
2. Work of the Recursive Function
3. Examples

# Recursive Function in C++

***Recursion*** is a programming technique where a function calls itself over again and again with modified arguments until it reaches its base case, where the recursion stops.

It breaks a problem down into smaller, more manageable sub-problems, recursion allows for elegant and better solutions to complex problems.

A ***recursive function*** is a function which is particularly used for recursion where a function calls itself, either directly or indirectly, to address a problem. It must include at least one base case to terminate the recursion and one recursive case where the function invokes itself.

- **Base Case – It's a case where recursion stops or ends after reaching that particular condition.**
- **Recursive Case – It's a case where a function calls itself over again with decremented value until and unless it reaches its base case.**

## Creating a Recursive Function

The following syntax is used to implement a recursive function in C++ :

```
function name(param_1, param_2...)
{
  <base condition>
  <function body>
  <return statement>
}
```

→ **In the function body, a recursive case will be defined**

# Recursive Function in C++

Here,

- Where, function name(param_1, param_2..) is a function declared as "name" passing with multiple parameters in it as per requirement.
- Now the function body is being divided into three sub-categories : base condition, function body and return statement.
- In base condition we will define its base case, where recursion has to stop or end.
- In the function body, a recursive case will be defined, where we need to call the function over again and again as per requirement.
- At last, the return statement will return the final output of the function.

## Calling a Recursive Function

Calling a recursive function is just like calling any other function, where you will use the function's name and provide the necessary parameters in int main() body.

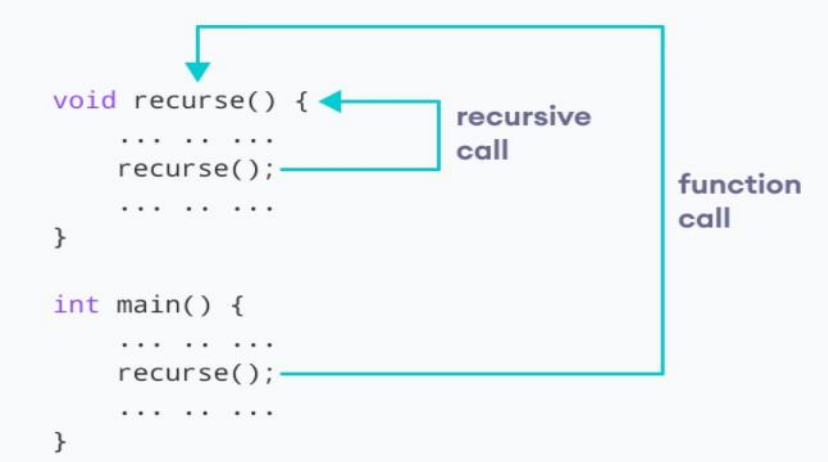
To call a recursive function, use the following syntax :

**func_name(value);**

# Work of the Recursive Function

```
void recurse)
{
   …
   recurse); //Function calls itself
   …
}

int main)
{
   …
   recurse); //Sets off the recursion
   …
}
```




The recursion continues until some condition is met.

To prevent infinite recursion, **if...else statement** (or similar approach) can be used where one branch makes the recursive call and the other doesn't.

تجزء المشكلة الى مشاكل صغيرة ومتكررة
وكل جزء يعتمد في حله على الجزء الاخر
الى ان نصل الى جزء يتوقف فيه التكرار

# Example of the Recursive Function

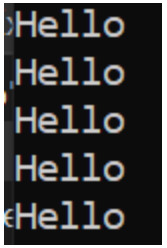
Write a c++ recursive function named (printHello) to print Hello five times and call it in the main program.

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

void printHello(int n)
{
    if (n == 0) return;
    cout << "Hello" << endl;
    printHello(n - 1);
}

int main()
{
    printHello(5);
    return 0;
}
```

| n | Action |
|---|---|
| 5 | Print `"Hello"` and call `printHello(4)` |
| 4 | Print `"Hello"` and call `printHello(3)` |
| 3 | Print `"Hello"` and call `printHello(2)` |
| 2 | Print `"Hello"` and call `printHello(1)` |
| 1 | Print `"Hello"` and call `printHello(0)` |
| 0 | Base case reached ( `n == 0` ), function stops |




هذه الطريقة من الاستدعاء الذاتي او التكراري فعالة لطباعة عدد معين من القيم، ولكن يمكن أيضًا استخدام الحلقات for أو while لتحقيق نفس النتيجة بطريقة أكثر كفاءة من حيث استهلاك الذاكرة، لأن الاستدعاء الذاتي او التكراري يستخدم المكدس stack لتخزين كل استدعاء للدالة.

**Base Condition**

if (n == 0) return;

**Recursive Case**

printHello(n – 1);

In the above program, we used a function named **printHello()** that takes a number **n** and prints **“Hello”** once and call itself for with decremented argument. This goes on till n = 0. When argument **n** is **5**, it prints **“Hello”** 5 times.

# Example of the Recursive Function (factorial)

5! = Factorial(5)= 5 * 4 * 3 * 2 * 1

Factorial(4)= 4*3 *2 *1

Factorial(3)= 3*2*1

Factorial(2)=2*1

Factorial(1)=1

Factorial (4)

5! = Factorial(5)= 5 * 4 * 3 * 2 * 1

Factorial(4)= 4*3 *2 *1

Factorial(3)= 3*2*1

Factorial(2)=2*1

Factorial(1)=1

Factorial (4)

5! = Factorial(5)= 5 * 4 * 3 * 2 * 1

Factorial (3)

Factorial(4)= 4*3 *2 *1

Factorial(2)=2*1

Factorial(1)=1

Factorial (4)

5! = Factorial(5)= 5 * 4 * 3 * 2 * 1

Factorial (3)

Factorial(4)= 4*3 *2 *1

Factorial (2)

Factorial(3)= 3*2*1

Factorial(2)=2*1

Factorial(1)=1

## إيجاد المضروب، Factorial

Factorial (4)

5! = Factorial(5)= 5 * 4 * 3 * 2 * 1

Factorial (3)

Factorial(4)= 4*3 *2 *1

Factorial (2)

Factorial(3)= 3*2*1

Factorial (1)

Factorial(2)=2*1

Factorial(1)=1

✓ نقسمها أجزاء متكررة (العدد ضرب مضروب/ factorial العدد الذي قبله)

Factorial (x) = x * Factorial (x-1)

✓ وصلنا لجزء يوقف فيه التكرار base case

Fact (x) = ?
Fact (5) = 120

Fact(x) = x * Fact(x-1)
Fact(1)= 1

```
Factorial (x) {

        return x * Factorial (x-1)

}
```

Fact (x) = ?
Fact (5) = 120

Fact(x) = x * Fact(x-1)
Fact(1)= 1

```
Factorial (x) {
  If (x==1)
      return 1
  else
      return x * Factorial (x-1)
}
```

Result= Factorial (5)

"Expected Result: 5!= 120"

5

```
Factorial (x) {

If (x==1)
      return 1;
else
      return x * Factorial (x-1)
}
```

Factorial (5) = 5 * Factorial (4)

Result= Factorial (5)

"Expected Result: 5!= 120"

5 4

```
Factorial (x) {

If (x==1)
      return 1;
else
      return x * Factorial (x-1)
}
```

Factorial (5) = 5 * Factorial (4)

Factorial (4)= 4 * Factorial (3)

Result= Factorial (5)

"Expected Result: 5!= 120"

5 4 3

```
Factorial (x) {

If (x==1)
      return 1;
else
      return x * Factorial (x-1)
}
```

Factorial (5) = 5 * Factorial (4)

Factorial (4)= 4 * Factorial (3)

Factorial (3)= 3 * Factorial (2)

Result= Factorial (5)

"Expected Result: 5!= 120"

5 4 3 2

```
Factorial (x) {

If (x==1)
      return 1;
else
      return x * Factorial (x-1)
}
```

Factorial (5) = 5 * Factorial (4)

Factorial (4)= 4 * Factorial (3)

Factorial (3)= 3 * Factorial (2)

Factorial (2)= 2 * Factorial (1)

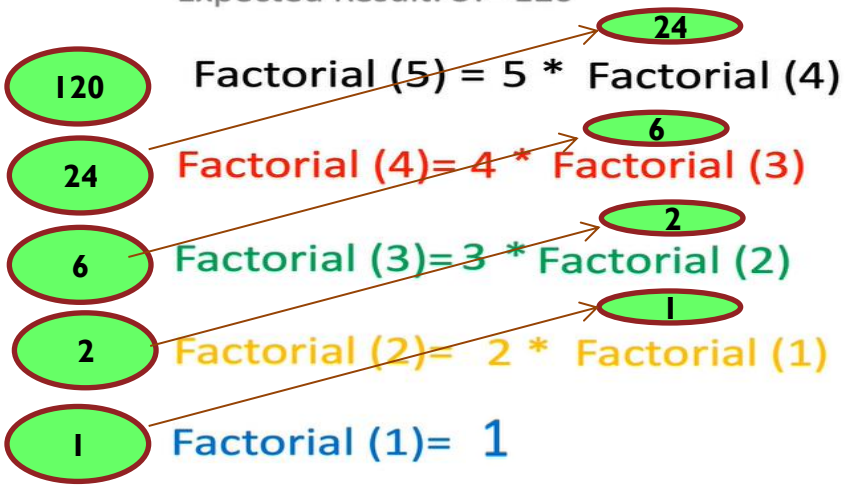
Result= Factorial (5)

"Expected Result: 5!= 120"

5 4 3 2 1

```
Factorial (x) {

If (x==1)
      return 1;
else
      return x * Factorial (x-1)
}
```



# Example of the Recursive Function

Write a c++ program to find the factorial of a number using recursive function named (factorial).

Or Write a c++ recursive function named (factorial) to find the factorial of a number and call it in the main program.

```cpp
#include <iostream>
using namespace std;
// Recursive Function to Calculate Factorial
int factorial(int n)
{
    // Base case
    if (n <= 1) //  if (n==0 || n==1)
    {
       return 1;
    }
    // Recursive case
    return (n * factorial(n - 1)); // return n * factorial(n - 1);
}

int main() {
    int num;
    cout << "Enter a positive integer: ";
    cin >> num;

    if (num < 0)
    {
        cout << "Wrong Input! Factorial is not defined for negative integers." << endl;
    }
    else {
        cout << "Factorial of " << num << " is " << factorial(num) << endl;
    }

    return 0;
}
```

```
Enter a positive integer: 4
Factorial of 4 is 24
```

Write a c++ program to calculate the sum of numbers from 1 to n using recursive function named (nSum).

Or Write a c++ recursive function named (nSum) to calculate the sum of numbers from 1 to n and call it in the main program.

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

// Recursive function to calculate the sum of numbers from 1 to n
int nSum(int n)
{

    // Base case to stop recursion when n = 0
    if (n == 0)
        return 0;
    // Recursive call
    return n + nSum(n - 1);
}
int main()
{
    // Calling the function
    int sum = nSum(5);
    // Printing the result
    cout << sum;
    return 0;
}
```

When calling nSum(5), the following sequence is executed: 5+4+3+2+1+0

nSum(5) = 5 + nSum(4)

nSum(4) = 4 + nSum(3)

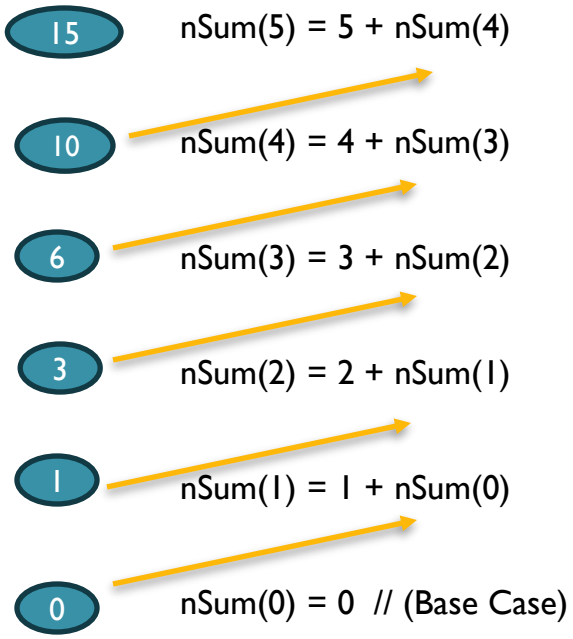
nSum(3) = 3 + nSum(2)

nSum(2) = 2 + nSum(1)

nSum(1) = 1 + nSum(0)

nSum(0) = 0 // (Base Case)

Now, calculating the result:




Final Output: 15

Write a c++ program to calculates the sum of the following series using Recursive function:

$$Z(n) = 1^2 + 2^2 + 3^2 + \ldots + n^2$$

```cpp
#include <iostream>
using namespace std;
// Recursive function to calculate the sum of squares
int sumOfSquares(int n)
{
    // Base case: if n is 0, return 0
    if (n == 0)
        return 0;

    // Recursive case: sum of squares formula
    return (n * n) + sumOfSquares(n - 1); // return pow(n,2) + sumOfSquares(n - 1);
}
int main()
{
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    if (n < 0)
        cout << "Invalid input! Please enter a non-negative number." << endl;
    else
        cout << "Sum of squares of first " << n << " numbers is: " << sumOfSquares(n) <<
endl;
    return 0;
}
```

```
Enter a positive integer: 3
Sum of squares of first 3 numbers is: 14
```

Enter a positive integer: 3

Recursive Calculation:

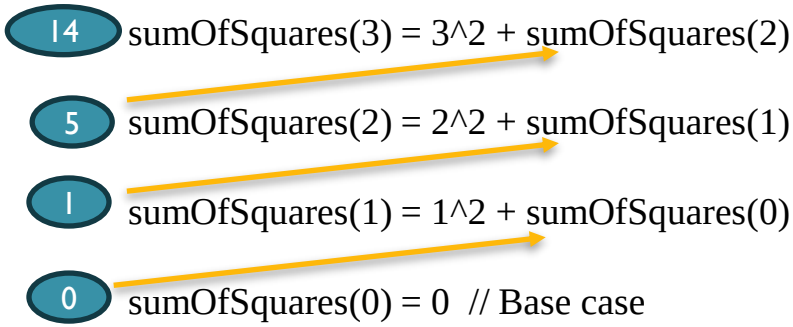
sumOfSquares(3) = 3^2 + sumOfSquares(2)

sumOfSquares(2) = 2^2 + sumOfSquares(1)

sumOfSquares(1) = 1^2 + sumOfSquares(0)

sumOfSquares(0) = 0  // Base case

Now, adding values back:




Sum of squares of first 3 numbers is: 14