INFO-I308 Laboratory Session #7

LAB GOALS

- Understanding recursion
- Iterative vs. recursive solutions

Step 1: Start with a simple C++ console project called `I308_Lab7`.

Step 2: Add a new C++ source file named (main.cpp).
Step 3: The objective of this lab is to understand and compare and contrast recursive vs iterative algorithms. We will achieve this by creating a number of examples that solve a problem using both approaches.

Let's begin with defining factorial.

The **factorial** of a non-negative integer $n$, denoted by $n!$, is the product of all positive integers less than or equal to $n$. For example:

$$3! = 3 \times 2 \times 1 = 6$$

This problem can easily be implemented as an iterative solution (step 4) or a recursive solution (step 5):

**Step 4:** Let's type the following lines of code to set things up:

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

// Forward declaration
int factorial_iterative(int value);
int factorial_recursive(int value);

void main(void)
{
    int value, result;
    value = 5;
    result = factorial_iterative(value);
    cout << "Factorial of " << value << " using (Iterative Calculation) is: " << result << endl;

    result = factorial_recursive(value);
    cout << "Factorial of " << value << " using (Recursive Calculation) is: " << result << endl;
    getchar();
}
```

**Step 5:** Now, let's create an **iterative version of factorial**.

```cpp
int factorial_iterative(int value)
{
    int result=1;
    for(int i = value; i>=1; i--) {
        result = result * i;
    }
    return(result);
}
```
Step 6: Now, let's create a **recursive version of factorial.**

\[
\text{factorial}(n) = \begin{cases} 
1, & \text{if } n = 0 \\
 n \times \text{factorial}(n-1), & \text{if } n > 0 
\end{cases}
\]

For example:

\[
3! = \text{factorial}(3) = 3 \times \text{factorial}(2) = 3 \times 2 \times \text{factorial}(1) = 3 \times 2 \times 1 \times \text{factorial}(0) = 3 \times 2 \times 1 \times 1
\]

```c
int factorial_recursive(int value)
{
    if (value ==0)
        return 1;
    else
        return(value * factorial_recursive(value -1));
}
```

Step 7: Run the program and note the results. Try running the program with different values and observe the results.

![Factorial output](image)

Step 8: Now just to convince ourselves, a little more, let's try adding the elements of an array using both iterative as well as recursive method.

Place the following code in the forward declaration section:

```c
int sum_iterative(int *MyArray, int ArraySize);
int sum_recursive(int *MyArray, int ArraySize);
```

Place this code in the main() function before the getchar() line:

```c
int MyArray[4] = {5,9,3,2};
result = sum_iterative(&MyArray[0], 4);
cout << "The sum of {5,9,3,2} using (Iterative Calculation) is: " << result << endl;
result = sum_iterative(&MyArray[0], 4);
cout << "The sum of {5,9,3,2} using (Recursive Calculation) is: " << result << endl;
```
Now, place the following modules in the program:

```c
int sum_iterative (int *MyArray, int ArraySize)
{
    int result = 0;

    for(int i=0; i < ArraySize; i++) {
        result = result + MyArray[i];
    }
    return (result);
}

int sum_recursive (int *MyArray, int ArraySize)
{
    if(ArraySize == 1)
        return(MyArray[0]);
    else
        return(MyArray[0] + sum_recursive(&MyArray[0], ArraySize-1));
}
```

Step 9: Run the program and note the results.

```
Factorial of 1 using <Iterative Calculation> is: 1
Factorial of 1 using <Recursive Calculation> is: 1
The sum of {5,9,3,2} using <Iterative Calculation> is: 19
The sum of {5,9,3,2} using <Recursive Calculation> is: 19
```