Counter Loops And Dependent Sub-problems

Counter Loop: Single Instruction

We have seen FOR loop used for repeating 1 instruction.

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

main(){

 for(int x = 0; x < 5; x = x + 1)
 {
    cout << "Welcome to UET!!" << endl;
 }
}
```

Counter Loop: Multiple Instructions

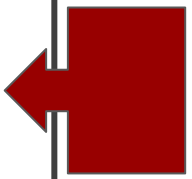
We can also use multiple instructions within the FOR loop.

Suppose, we want to write a C++ program that repeats multiple instructions to print the table of 5.

Counter Loop: Multiple Instructions

Suppose, we want to write a C++ program that repeats multiple instructions to print the table of 5.

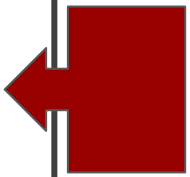
```
#include <iostream>
using namespace std;
main()
{
    int multiple;
    for (int num = 1; num <= 10; num = num + 1)
    {



    }
}
```



Counter Loop: Multiple Instructions

Suppose, we want to write a C++ program that repeats multiple instructions to print the table of 5.

```
#include <iostream>
using namespace std;
main()
{
    int multiple;
    for (int num = 1; num <= 10; num = num + 1)
    {
        multiple = 5 * num;
        cout << "5 * " << num << " = " << multiple << endl;
    }
}
```



Counter Loop: Multiple Instructions

Write a C++ program that finds sum of numbers between 1 and 100

Counter Loop: Multiple Instructions

Write a C++ program that finds sum of numbers between 1 and 100

```
C:\C++ Programming>c++ program.cpp -o program.exe

C:\C++ Programming>program.exe
The sum of Numbers from 1 to 100 is: 5050
```

Counter Loop: Multiple Instructions

Write a C++ program that finds sum of numbers between 1 and 100

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

main()
{
    int sum = 0;
    for (int x = 1; x <= 100; x = x + 1)
    {

    }
}
```

Counter Loop: Multiple Instructions

Write a C++ program that finds sum of numbers between 1 and 100

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

main(){
    int sum = 0;
    for (int x = 1; x <= 100; x = x + 1)
    {
        sum = sum + x;
    }
    cout << "The sum of Numbers from 1 to 100 is: ";
    cout << sum;
}
```

Counter Loop: Independent Iterations

In first two problems the individual iterations are **Independent** of each other.

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

main(){

 for(int x = 0; x < 5; x = x + 1)
 {
    cout << "Welcome to UET!!" << endl;
 }
}
```

```
#include <iostream>
using namespace std;
main()
{
    int multiple;
    for (int num = 1; num <= 10; num = num + 1)
    {
        multiple = 5 * num;
        cout << "5 * " << num << " = " << multiple <<
endl;
    }
}
```

Counter Loop: Independent Iterations

Independent Iteration means the **result** of previous iteration **are not required** for next iterations.

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

main(){

 for(int x = 0; x < 5; x = x + 1)
 {
    cout << "Welcome to UET!!" << endl;
 }
}
```

```
#include <iostream>
using namespace std;
main()
{
    int multiple;
    for (int num = 1; num <= 10; num = num + 1)
    {
        multiple = 5 * num;
        cout << "5 * " << num << " = " << multiple <<
endl;
    }
}
```

Counter Loop: Dependent Iterations

However, there are set of problems that require the result from previous iterations.

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

main(){
    int sum = 0;
    for (int x = 1; x <= 100; x = x + 1)
    {
        sum = sum + x;
    }
    cout << "The sum of Numbers from 1 to 100 is: ";
    cout << sum;
}
```

Counter Loop: Dependent Iterations

Dependent Iteration means the result of previous iteration are required for next iterations.

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

main(){
    int sum = 0;
    for (int x = 1; x <= 100; x = x + 1)
    {
        sum = sum + x;
    }
    cout << "The sum of Numbers from 1 to 100 is: ";
    cout << sum;
}
```

Working Example 2: Fibonacci Series

The Fibonacci numbers are the numbers in the following integer sequence.

Fibonacci Number Series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811...

Working Example 2: Fibonacci Series

You are given the first 2 fibonacci numbers.

```
int n1 = 0;
int n2 = 1;
```

Fibonacci Number Series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811...

Display the next 5 fibonacci numbers.

Solution: Fibonacci Series

```
#include <iostream>
using namespace std;
main()
{
    int n1 = 0;
    int n2 = 1;
    int next;
    for(int x = 0; x < 5; x = x + 1)
    {
        next = n1 + n2;
        cout << next << ", ";
        n1 = n2;
        n2 = next;
    }
}
```

Working Example 3: Fibonacci Series

Input: Take three variables, n1, n2, n3 from user

Make a function named

`void fibonacciSeries(int n1, int n2, int n3)`

Output: Starting from n1 + n2 print Fibonacci series up to n3.

Hint: Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with n1=2, n2=3, and n3=9, terms will be:

5, 8, 13, 21, 34, 55, 89

Working Example 2: Fibonacci Series

Test Cases:

Input	Output
n1: 2 n2: 3 n3: 9	5, 8, 13, 21, 34, 55, 89, 144, 233
n1: 0 n2: 1 n3: 20	1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946,

Solution: Fibonacci Series

```
#include <iostream>
using namespace std;
void fibonacciSeries(int n1, int n2, int n3)
{
    int next;
    for(int x = 0; x < n3; x = x + 1){
        next = n1 + n2;
        cout << next << ", ";
        n1 = n2;
        n2 = next;
    }
}
```

Learning Objective

In this lecture, we learnt how to write a C++ Program that solves dependent Problems using the Counter Loops

Conclusion

- Within the body of loops one can write multiple instructions.
  In simple problems the individual iterations of the loops are independent of each other.
- However, complex problems need to use the result from the previous iterations.
  Each iteration in these problems is called a dependent sub problem.

Self Assessment

1. Write a factorial function that multiplies all whole numbers from our chosen number down to 1 and returns the result.
   Examples:

   factorial(4) = 4 × 3 × 2 × 1 = 24
   factorial(7) = 7 × 6 × 5 × 4 × 3 × 2 × 1 = 5040
   factorial(1) = 1