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

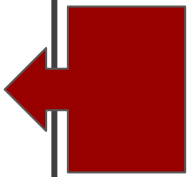
We can also use multiple instructions within the FOR loop.

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)
    {
        multiple = 5 * num;
        cout << "5 * " << num << " = " << multiple << endl;
    }
}
```



Counter Loop: Independent Iterations

However, In both of these 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.

For example, Write a C++ program that find sum of numbers between 1 and 100

Note: Do Not Use Summation Formula

Counter Loop: Dependent Iterations

For example, Write a C++ program that find sum of numbers between 1 and 100

Note: Do Not Use Summation Formula

We have a computational step that can add two numbers. Therefore, we can only add two numbers at one step.

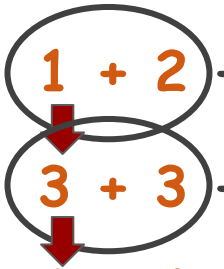
```
c = a + b;
```

Counter Loop: Dependent Iterations

For example, Write a C++ program that find sum of numbers between 1 and 100

Note: Do Not Use Summation Formula

At a step (n), add two numbers and the result shall be added into next number at next step (n+1).



1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + …. + 100

3 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + …. + 100

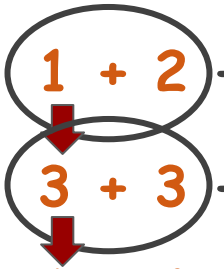
6 + 4 + 5 + 6 + 7 + 8 + 9 + …. + 100

……

Counter Loop: Dependent Iterations

At a step (n), add two numbers and the result shall be added into next number at next step (n+1).

Whenever such problem comes up, we first need to write it as a collection of repeating computational steps and then identify the repeating pattern.



1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + …. + 100

3 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + …. + 100

6 + 4 + 5 + 6 + 7 + 8 + 9 + …. + 100

……

Counter Loop: Repeating Sub Problems

We can see this problem as the repeating sub problems of adding two numbers.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

Counter Loop: Repeating Sub Problems

Here, each iteration is called sub problem.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

Counter Loop: Repeating Sub Problems

These sub problems are dependent because the result of each sub problem is used by next iteration.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

Counter Loop: How to Write Program

Before writing the program, First step is to identify the pattern within the sub problem such that this pattern can be written as a generic expression or set of generic expressions.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

……

Counter Loop: How to Write Program

One pattern is quite obvious that in each iteration number is repeating as sequence from 1 to 100.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

……

Counter Loop: How to Write Program

To generate this sequence at each iteration, we can use for Loop Statement.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

```
for (int x = 1; x <= 100; x = x + 1)
{

}
```

Counter Loop: Identifying the Pattern

Now, we need to find the pattern within the sub problems such that this pattern can be written as an or set of generic expressions.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

```
for (int x = 1; x <= 100; x = x + 1)
{

}
```

Counter Loop: Identifying the Pattern

Let's first do it for first iteration and then make it generic.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

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

Counter Loop: Identifying the Pattern

In second iteration, we need to add the result of first iteration into it. So, we can use sum variable because it shall have the result of previous variable.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

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

Counter Loop: Identifying the Pattern

In second iteration, we need to add the result of first iteration into it. So, we can use sum variable because it shall have the result of previous variable.

0 + 1 = 1

1 + 2 = 3

3 + 3 = 6

6 + 4 = 10

.....

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

Counter Loop: Code Snippet

The Complete Code is as follows:

```
#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: Output

The Output of the Code is as follows:

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

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

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

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 and 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:

2, 3, 5, 8, 13, 21, 34, 55, 89

Working Example 2: Fibonacci Series

Test Cases:

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

Solution: Fibonacci Series

```
#include <iostream>
using namespace std;
void fibonacciSeries(int n1, int n2, int n3)
{
    int next;
    cout << n1 << ", ";
    cout << n2;
    for(int x = 0; x < n3 - 2; 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 larger problems by decomposing it into smaller subproblems and combining their solution to achieve final results using the Counter Loop

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