Programming Essentials

(UOMU022023)

اساسيات البرمجة
2025-2024

Loops

by
Dr Murtada Dohan
murtada.dohan@uomus.edu.iq


AL MUSTAQBAL UNIVERSITY
جامعة المستقبل
أسست عام
2010


كلية الهندسة والتقنيات
الهندسية
COLLEGE OF ENGINEERING &
TECHNOLOGY
AL MUSTAQBAL UNIVERSITY
جامعة المستقبل
2010

Recap

Syntax of if-else Statement

```
if (condition) {
   // code block if condition is true
} else {
   // code block if condition is false
}
```

1. Boolean variable

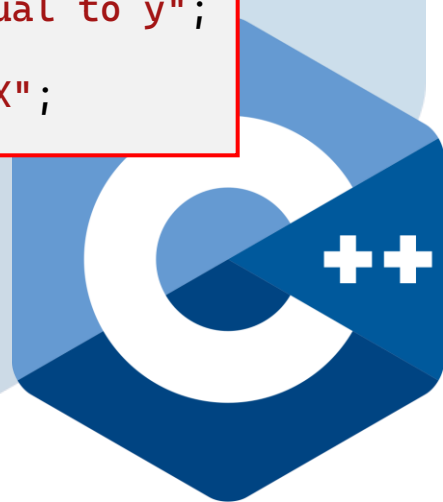
Example:

```
bool x = true;
if (x)
     cout << "Here we go!";
else
     cout << "We cannot make it";
```

2. Comparison

Example:

```
int x = 10, y = 56;
if (x>=y)
     cout << "X is greater or equal to y";
else
     cout << "Y is greater than X";
```



Syntax of switch Statement

```
switch (expression) {
case value1:
     // code block
     break;
case value2:
     // code block
     break;
     ...
default:
}  // code if no case matches
```

- Key Points:
  - expression must evaluate to an integer or character
  - break exits the switch (if omitted, execution falls through)
  - default is optional (executes if no case matches)

Today, Agenda

- The Increment and Decrement Operators
- for Loop
- while Loop
- do .. while Loop

The Increment and Decrement Operators

- ++ is the increment operator.

It adds one to a variable.

```
int val = 2;
val++;
```

is the same as `val = val + 1;`

- ++ can be used before (prefix) or after (postfix) a variable:

```
int val = 2;
++val;
val++;
```

The Increment and Decrement Operators

- `--` is the decrement operator.

It subtracts one to a variable.

```
int val = 2;
Val--;
```

is the same as `val = val - 1;`

- `--` can be used before (prefix) or after (postfix) a variable:

```
int val = 2;
--val;
Val--;
```

Prefix vs. Postfix

- ++ and -- operators can be used in complex statements and expressions
- In prefix mode (++val, --val) the operator increments or decrements, then returns the value of the variable
- In postfix mode (val++, val--) the operator returns the value of the variable, then increments or decrements

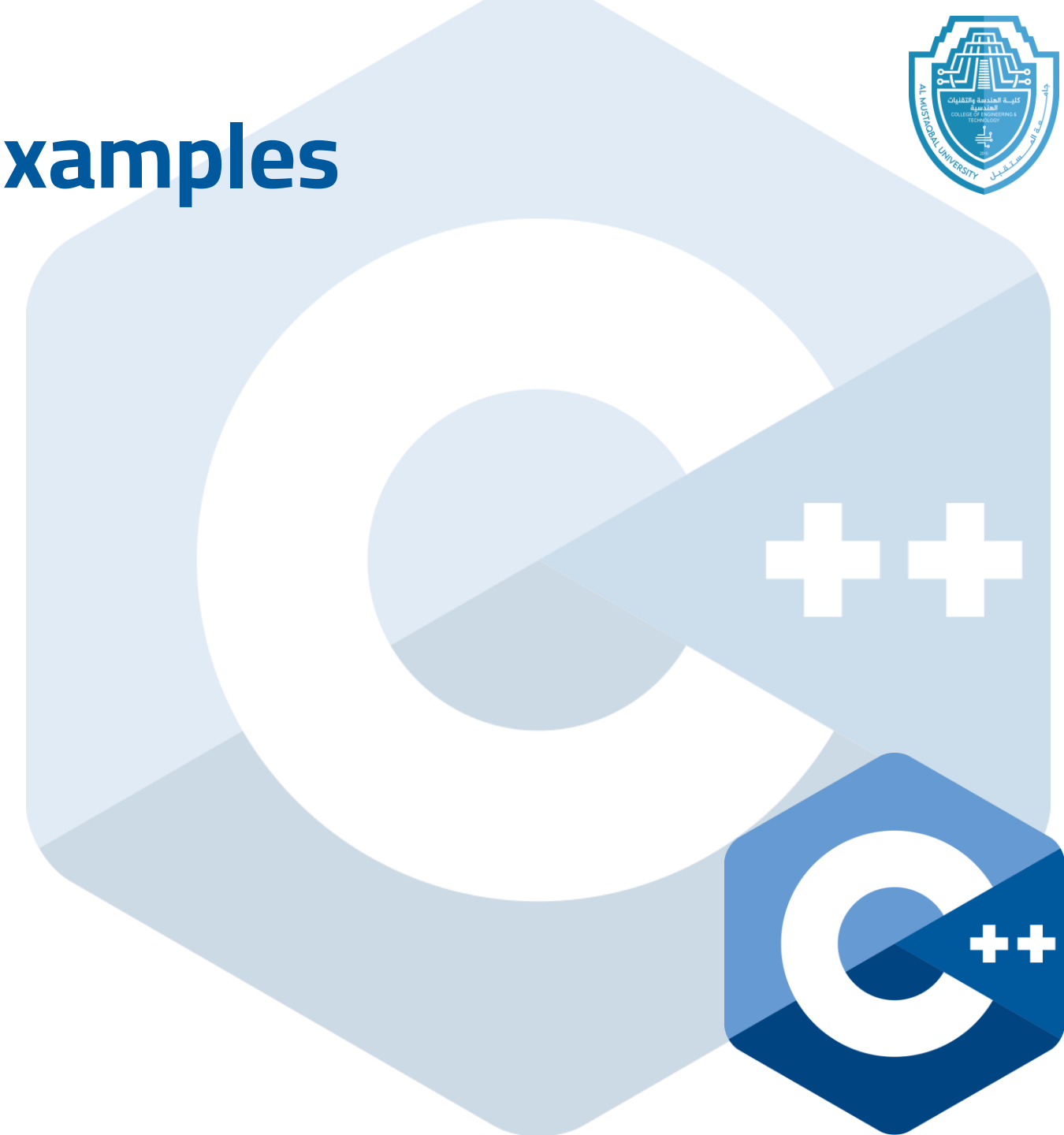
Prefix vs. Postfix - Examples

```
int num, val = 12;
cout << val++;

cout << ++val;

num = --val;

num = val--;
```



Prefix vs. Postfix - Examples

```
int num, val = 12;
cout << val++; // displays 12,
               // val is now 13;
cout << ++val; // sets val to 14,
               // then displays it
num = --val; // sets val to 13,
             // stores 13 in num
num = val--; // stores 13 in num,
           // sets val to 12
```

Notes on Increment and Decrement

- Can be used in expressions:

```
result = num1++ + --num2;
```

- Must be applied to something that has a location in memory. Cannot have:

```
result = (num1 + num2)++;
```

- Can be used in relational expressions:

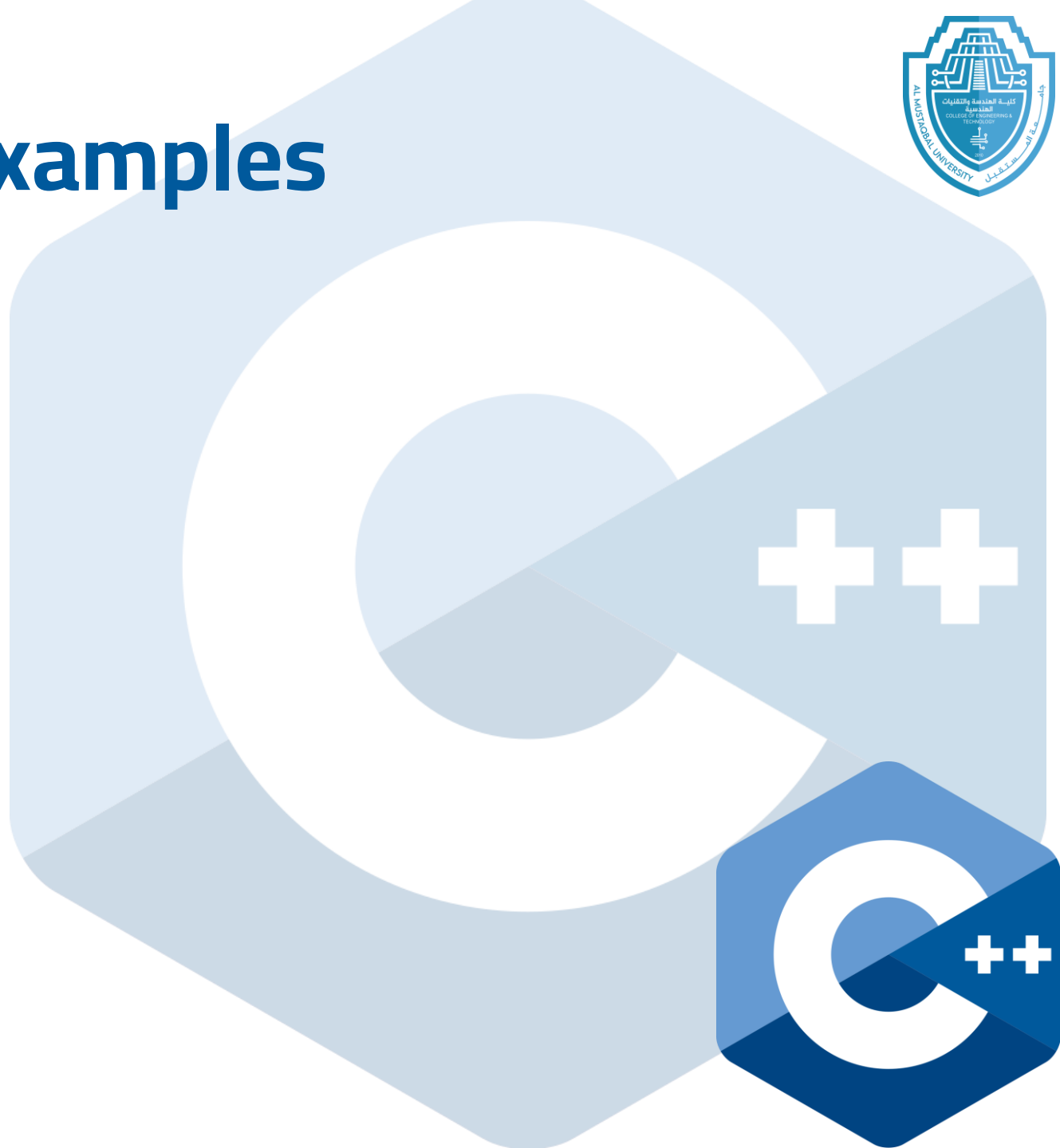
```
if (++num > limit)
```

- pre- and post-operations will cause different comparisons

Prefix vs. Postfix - Examples

```
int a = 4;
int b = 7;
b =  ++++a - ++++b;
cout << "a:" << a << endl;
cout << "b:" << b << endl;
```

- What are the values of a and b?



Prefix vs. Postfix - Examples

```
int a = 4;
int b = 7;
b =  --++a - ++++a;
cout << "a:" << a << endl;
cout << "b:" << b << endl;
```

- What are the values of a and b?

Introduction to Loops

The `while` Loop

- Loop: a control structure that causes a statement or statements to repeat
- General format of the while loop:

```
while (expression)
      statement;
```

- `statement;` can also be a block of
- statements enclosed in { }

The while Loop – How It Works

```
while (condition)
 {
   statement;
 }
```

- `condition` is evaluated
- – if `true`, then statement is executed, and expression is evaluated again
- – if `false`, then the loop is finished and program statements following statement execute

How the while Loop in Program Works

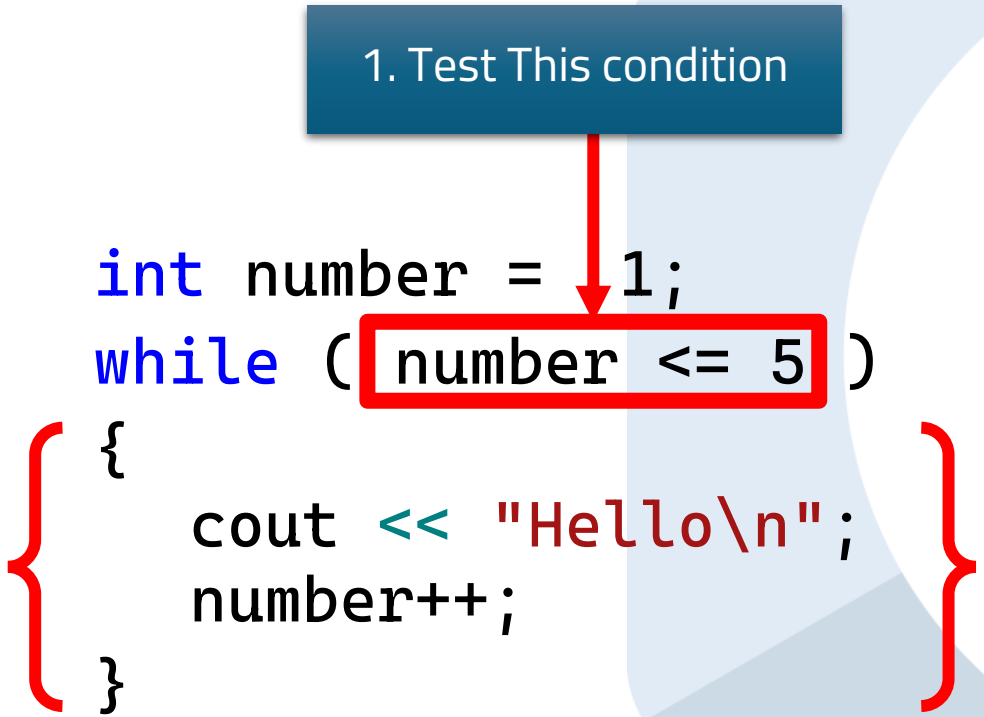
```
int number =  1;
while ( number <= 5 )
{
   cout << "Hello\n";
   number++;
}
```

How the while Loop in Program Works

1. Test This condition

```
int number = 1;
while ( number <= 5 )
{
   cout << "Hello\n";
   number++;
}
```

How the while Loop in Program Works



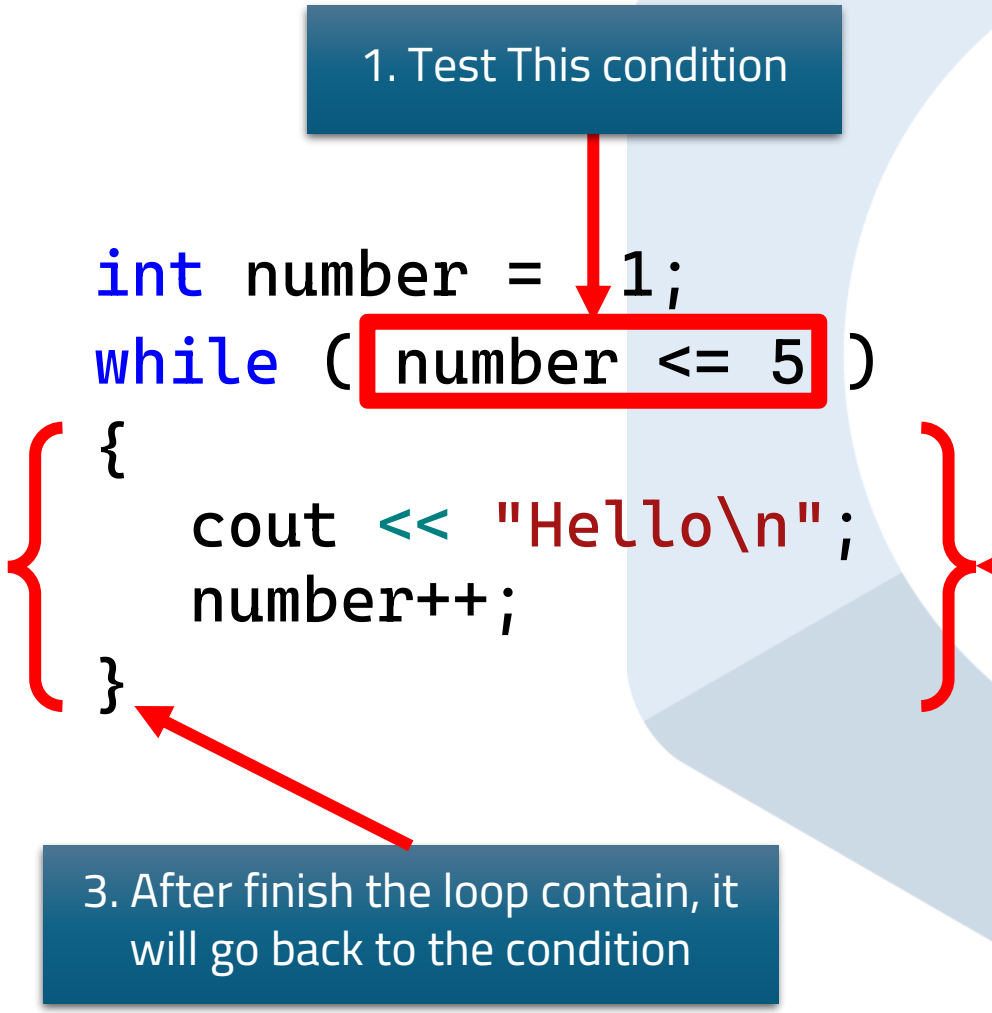
How the while Loop in Program Works

1. Test This condition

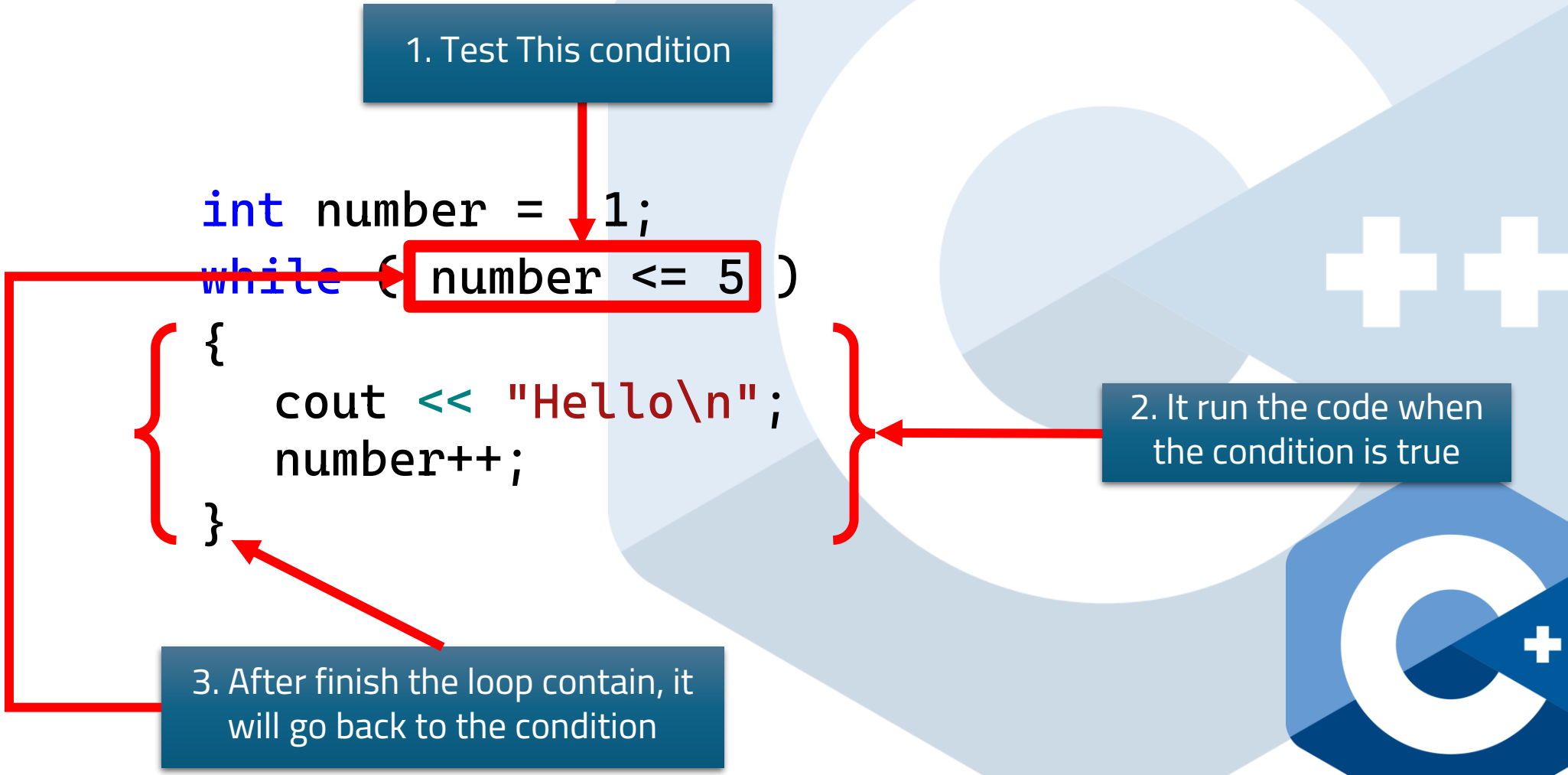
```
int number = 1;
while ( number <= 5 )
{
    cout << "Hello\n";
    number++;
}
```

2. It run the code when the condition is true

How the while Loop in Program Works



How the while Loop in Program Works



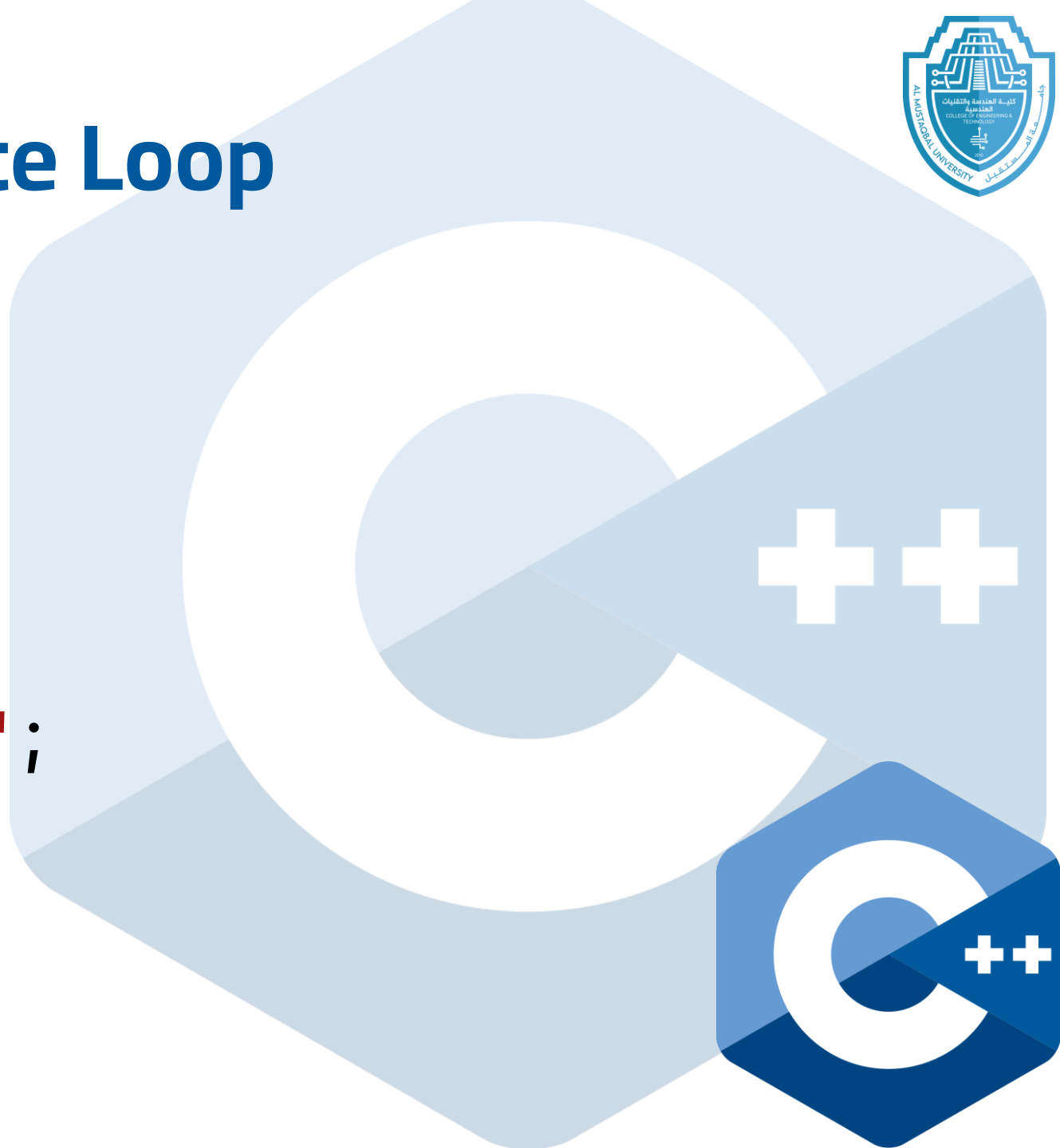
The while Loop is a Pretest Loop

- The condition is evaluated before the loop executes. The following loop will never execute:

```
int number = 6;
while (number <= 5)
{
    cout << "Hello\n";
    number++;
}
```

Example of an Infinite Loop

```
int number = 1;
while (number <= 5)
{
    cout << "Hello\n";
}
```



Using the while Loop for Input Validation

- Input validation is the process of inspecting data that is given to the program as input and determining whether it is valid.
- The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.

Using the while Loop for Input Validation

```
int number;
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
    cout << "Invalid Entry!"
        << "Enter a number less than 10: ";
    cin >> number;
}
```

Example: while Loop for Input Validation

```
int number;
cout << "Enter a Positive Number";
cin >> number;
while (number < 0)
{
    cout << "Invalid Entry!"
        << "Enter a number greater than or equal 0: ";
    cin >> number;
}
```

Example: sum all numbers between 1 and 10

```
int sum = 0;
int index = 1;
while (index <= 10)
{
   sum = sum + index;
   index++;
}
cout << "Results:" << sum;
```

Example: sum all even numbers between 1 and 100

```
int sum = 0;
int index = 1;
while (index <= 100)
{
  if(index % 2==0)
    sum = sum + index;
  index++;
}
cout << "Results:" << sum;
```

Example: sum number of numbers based on user input

```
int x;// input number from the user
int counter = 1;//control while loop
int results = 0;// sum all the input numbers

int Number_of_time;
cout << "Enter the count of the number:";
cin >> Number_of_time;


while (counter <= Number_of_time)
{
cout << "Enter Number " << counter << " :";
cin >> x;//input the number
results = results + x;// add number to the results
counter++;// increament counter to avoid inifinit loop
}
cout << "Results:" << results;// print the results of the sum
```

The do-while Loop

- `do-while`: a posttest loop – execute the loop, then test the ***condition***
  - General Format:

```
do
   statement;                    // or block in { }

while (condition);
```

  - Note that a semicolon is required after (***condition***)

An Example do-while Loop

```
int x = 1;
do
{
    cout << x << endl;
} while(x < 0);
```

Although the test expression is false, this loop will execute one time because `do-while` is a posttest loop.

An Example do-while Loop

```
int sum = 0;
int index = 1;
do
{
   sum = sum + index;
   index++;
} while (index <= 5);
cout << "Results:" << sum;
```

- Sum all numbers between 1 and 5

Example: write code to read an positive number for the user

```
int x;
do {
   cout << "Enter positive value X:";
   cin >> x;
} while (x < 0);
```

The `for` Loop

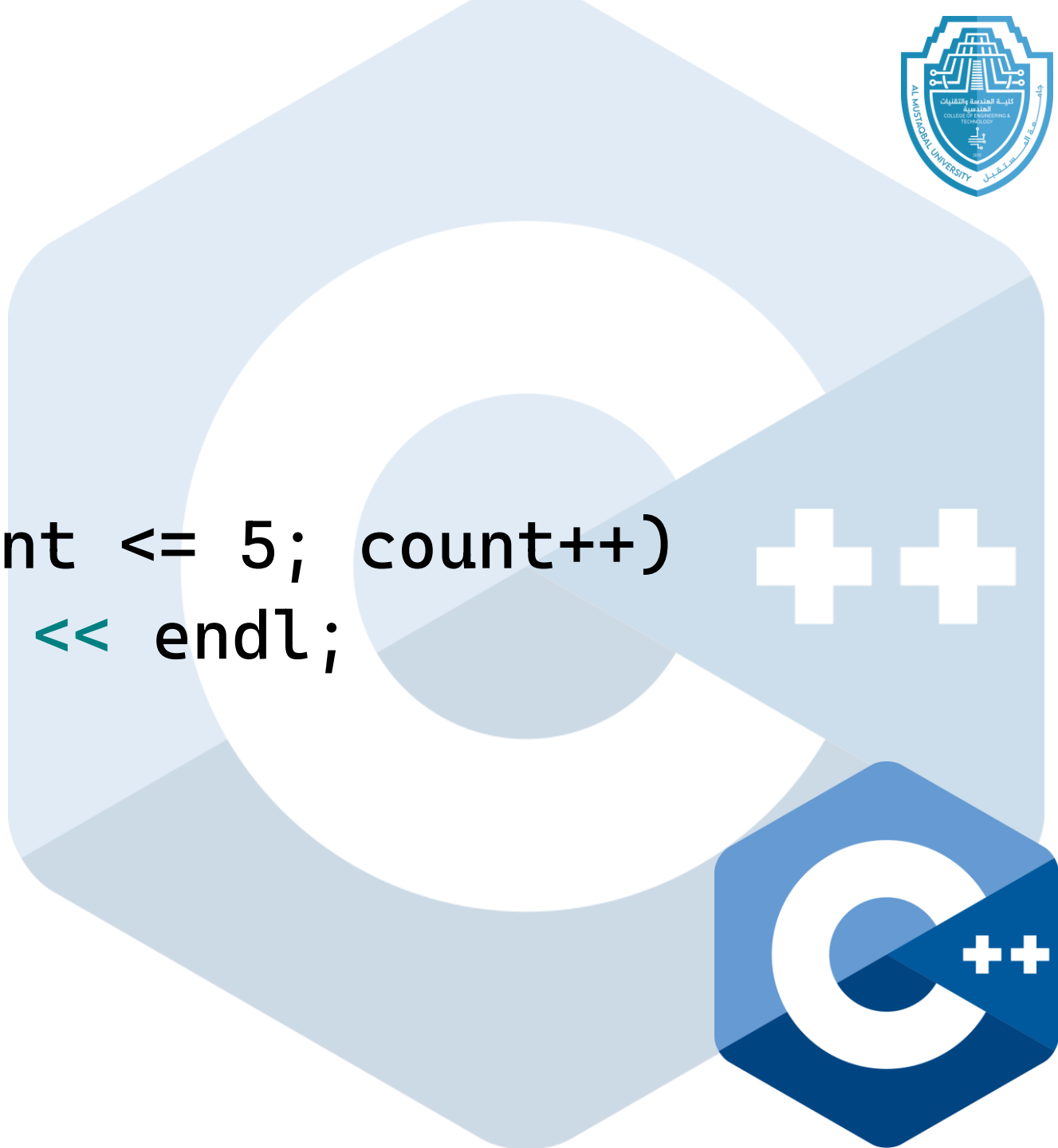
- Useful for counter-controlled loop
- General Format:

```
for(initialization; test; update)
  statement; // or block in { }
```

- No semicolon after the `update` expression or after the)

for Loop - Example

```
int count;
for (count = 1; count <= 5; count++)
    cout << "Hello" << endl;
```



The `for` Loop is a Pretest Loop

- The for loop tests its test expression before each iteration, so it is a pretest loop.
- The following loop will never iterate:

```
for (int count = 11; count <= 10; count++)
     cout << "Hello" << endl;
```

for Loop - Modifications

- You can have multiple statements in the *initialization* expression. Separate the statements with a comma:

Initialization Expression

```
int x, y;
for (x = 1, y = 1; x <= 5; x++)
{
cout << x << " plus " << y<< " equals " << (x + y)<< endl;
}
```

`for` Loop - Modifications

- You can also have multiple statements in the *update* expression. Separate the statements with a comma:

Update Expression

```
int x, y;
for (x = 1, y = 1; x <= 5; x++, y++)
{
     cout << x << " plus " << y<< " equals " << (x + y)<< endl;
}
```

for Loop - Modifications

- You can omit the *initialization* expression if it has already been done:

```
int sum = 0, num = 1;
for ( ; num <= 10; num++)
      sum += num;
```

`for` Loop - Modifications

- You can declare variables in the *`initialization`* expression:

```
int sum = 0;
for (int num = 0; num <= 10; num++)
     sum += num;
```

- The scope of the variable `num` is the `for` loop.

Deciding Which Loop to Use

- The `while` loop is a conditional pretest loop
  - Iterates as long as a certain condition exits
  - Validating input
  - Reading lists of data terminated by a sentinel
- The `do-while` loop is a conditional posttest loop
  - Always iterates at least once
  - Repeating a menu
- The `for` loop is a pretest loop
  - Built-in expressions for initializing, testing, and updating
  - Situations where the exact number of iterations is known

Nested Loops

- A nested loop is a loop inside the body of another loop
- Inner (inside), outer (outside) loops:

```
int row, col;
for (row = 1; row <= 3; row++) //outer
   for (col=1; col<=3; col++)//inner
      cout << row * col << endl;
```

Nested `for` Loop in Program

```
26    // Determine each student's average score.
27    for (int student = 1; student <= numStudents; student++)
28    {
29       total = 0;       // Initialize the accumulator.
30       for (int test = 1; test <= numTests; test++)
31       {
32          double score;
33          cout << "Enter score " << test << " for ";
34          cout << "student " << student << ": ";
35          cin >> score;
36          total += score;
37       }
38       average = total / numTests;
39       cout << "The average score for student " << student;
40       cout << " is " << average << ".\n\n";
41    }
```

Inner Loop

Outer Loop

Nested Loops - Notes

- Inner loop goes through all repetitions for each repetition of outer loop
- Inner loop repetitions complete sooner than outer loop
- Total number of repetitions for inner loop is product of number of repetitions of the two loops.

Breaking Out of a Loop

- Can use **`break`** to terminate execution of a loop
- Use sparingly if at all – makes code harder to understand and debug
- When used in an inner loop, terminates that loop only and goes back to outer loop

The `continue` Statement

- Can use **`continue`** to go to end of loop and prepare for next repetition
  - **`while`**, **`do-while`** loops: go to test, repeat loop if test passes
  - **`for`** loop: perform update step, then test, then repeat loop if test passes
- Use sparingly – like **`break`**, can make program logic hard to follow

Example : for-loop

- Print even numbers between 1 and 100

```
#include <iostream>
using namespace std;
int main()
{
for (int i = 0; i <= 100; i = i + 2) {
    cout << i << "\n";
}
}
```

Let’s try C++

Install Visual Studio and familiarise yourself with its interface.