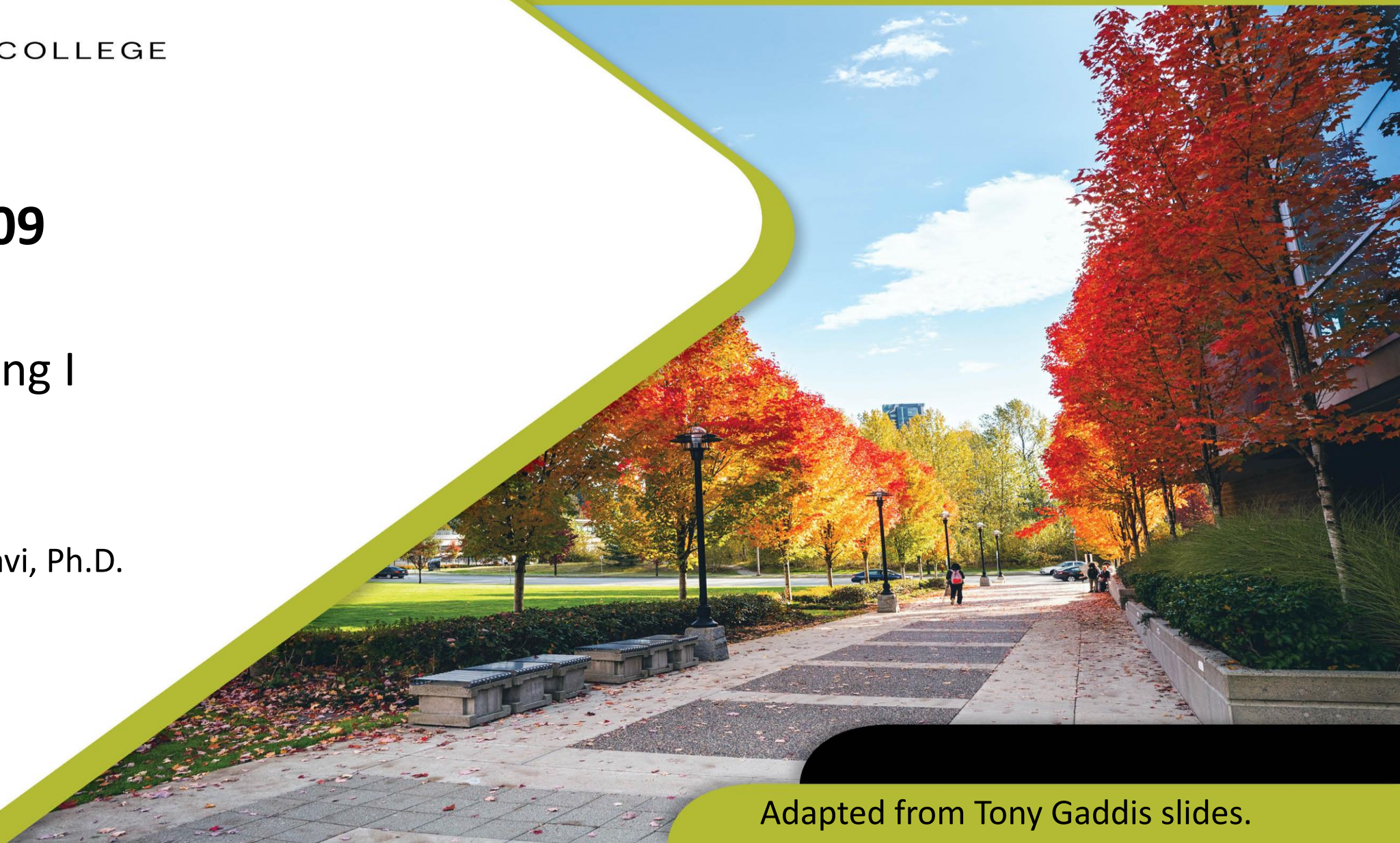
DOUGLAS COLLEGE

CMPT 1109

Programming I

Shahriar Khosravi, Ph.D.

Lecture 4



Adapted from Tony Gaddis slides.

Plan for Today

- The Increment and Decrement Operators
- Introduction to Loops: The **`while`** Loop
- Using the **`while`** Loop for Input Validation
- Counters
- The **`do-while`** Loop
- The **`for`** Loop
- Keeping a Running Total
- Sentinels
- Nested Loops
- Breaking (**`break`**) and Continuing (**`continue`**) a Loop
- Using Files for Data Storage

DOUGLAS COLLEGE

The Increment and Decrement Operators

The Increment and Decrement Operators

- **++** is the **increment operator**. It adds one to a variable.
- **`val++;`** is the same as **`val = val + 1;`** or **`val += 1;`**
- **++** can be used before (**prefix**) or after (**postfix**) a variable: **`++val;`** **`val++;`**
- **`--`** is the **decrement operator**. It subtracts one from a variable.
- **`val--;`** is the same as **`val = val - 1;`** or **`val -= 1;`**
- **`--`** can be also used before (**prefix**) or after (**postfix**) a variable: **`--val;`** **`val--;`**
- **++** 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.

Example

```
// This program demonstrates the ++ and -- operators.
#include <iostream>
using namespace std;

int main()
{
    int num = 4;   // num starts out with 4.

    // Display the value in num.
    cout << "The variable num is " << num << endl;
    cout << "I will now increment num.\n\n";

    // Use postfix ++ to increment num.
    num++;
    cout << "Now the variable num is " << num << endl;
    cout << "I will increment num again.\n\n";

    // Use prefix ++ to increment num.
    ++num;
    cout << "Now the variable num is " << num << endl;
    cout << "I will now decrement num.\n\n";

    // Use postfix -- to decrement num.
    num--;
    cout << "Now the variable num is " << num << endl;
    cout << "I will decrement num again.\n\n";

    // Use prefix -- to increment num.
    --num;
    cout << "Now the variable num is " << num << endl;
    return 0;
}
```

Prefix vs. Postfix Example

```
// This program demonstrates the ++ and -- operators.
#include <iostream>
using namespace std;

int main()
{
    int num, val = 12;
    cout << val++ << endl; // displays 12,
                           // val is now 13;
    cout << ++val << endl; // 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
    return 0;
}
```

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)++;
```

The operand of the increment and decrement operators must be an lvalue.
An lvalue identifies a place in memory whose contents may be changed.

- Can be used in relational expressions:

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

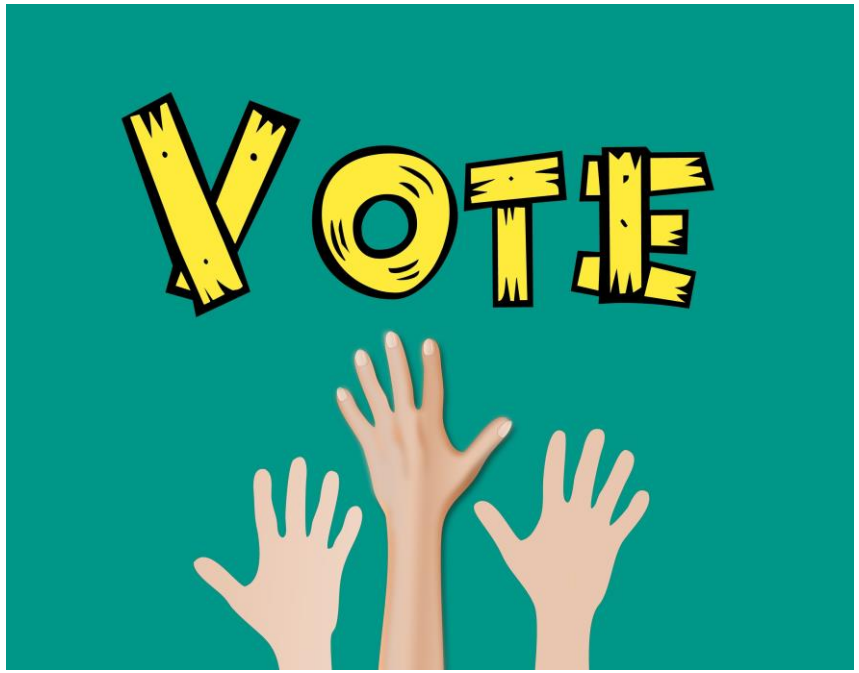
- Pre- and post-operations will cause different comparisons.

Poll 1 (Extra Credit)

What will the following code snippet display on the console screen?

```
int x = 0;
if (++x)
    cout << "True!\n";
else cout << "False!\n";
```

a) True!
b) False!



Please use the "Poll" window to participate for extra credit! One answer only please!

DOUGLAS COLLEGE

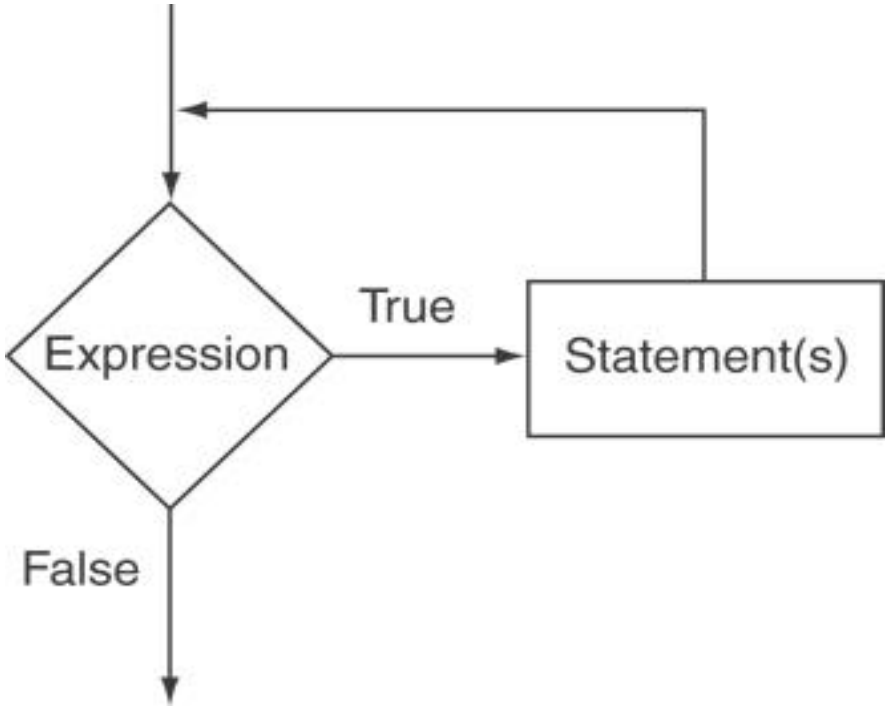
The `while` Loop

The `while` Loop

- **Loop**: a control structure that causes a statement or statements to repeat.
- General syntax of the **`while`** loop:

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

- **`statement;`** can also be a block of statements enclosed in curly brackets **`{ }`**
- How it works:
  - **`expression`** is evaluated first (**pretest loop**).
  - If it returns **`true`**, then statement is executed, and **`expression`** is evaluated again.
  - if **`false`**, then the loop is finished and statements following **`statement;`** execute

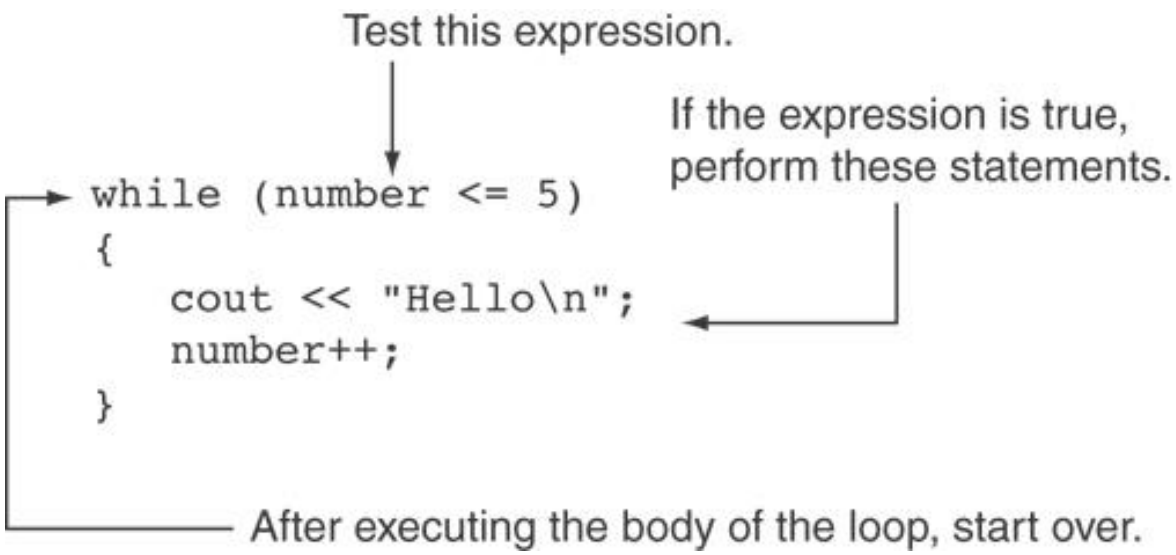


Example

```
// This program demonstrates a simple while loop.
#include <iostream>
using namespace std;

int main()
{
    int number = 0;

    while (number < 5)
    {
        cout << "Hello\n";
        number++;
    }
    cout << "That's all!\n";
    return 0;
}
```

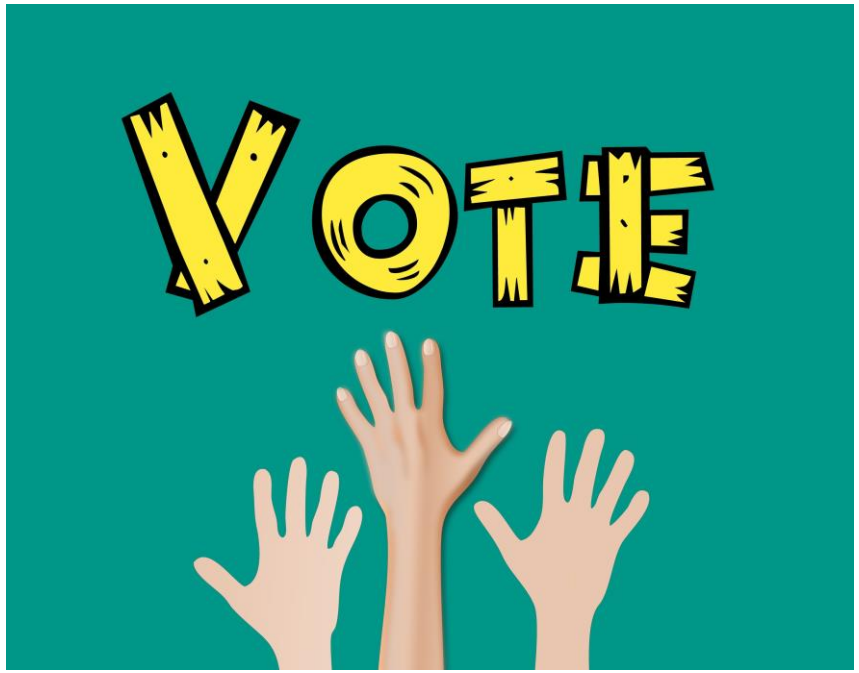


Poll 2 (Extra Credit)

What will the following loop ever execute?

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

a) Yes!
b) No!



Please use the "Poll" window to participate for extra credit! One answer only please!

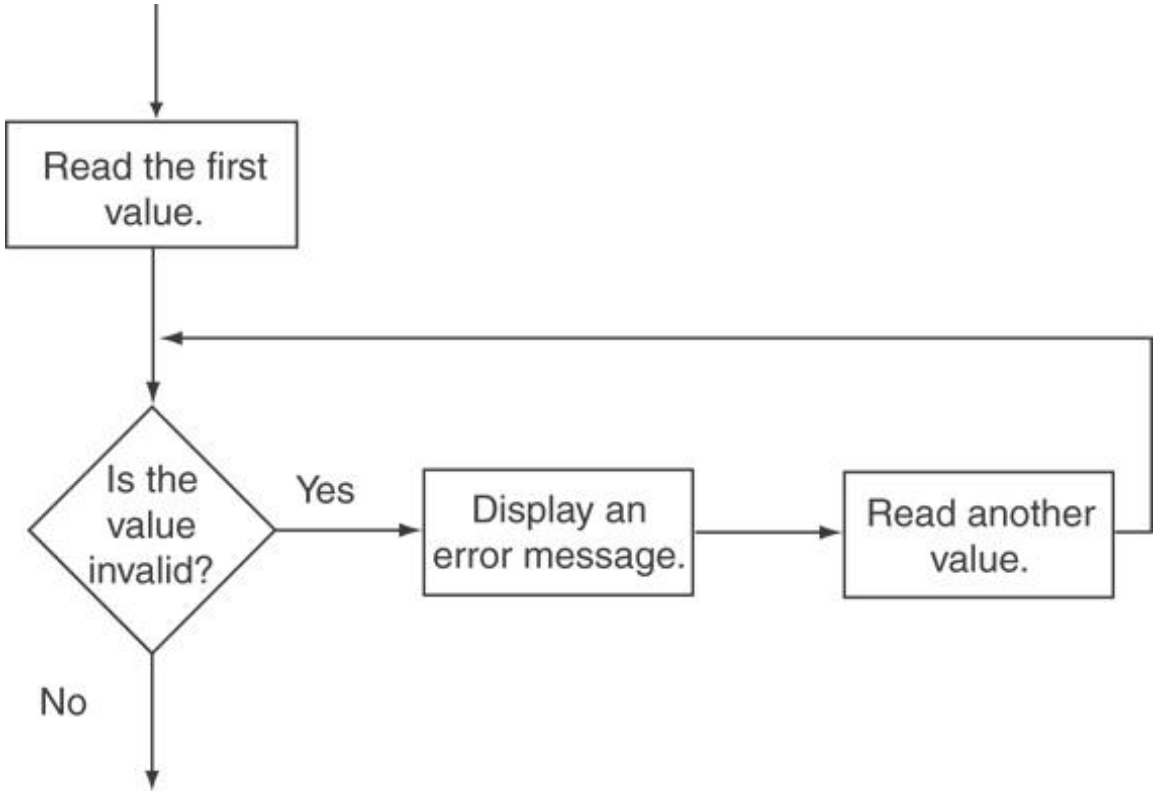
Infinite Loops

- The loop must contain code to make expression become **`false`**.
- Otherwise, the loop will have no way of stopping!
- Such a loop is called an **infinite loop**, because it will repeat an infinite number of times (or until the CPU crashes or interrupted)!
- Avoid writing infinite loops by double-checking your loop logic.

```
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.




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

Counters

- **Counter**: a variable that is incremented or decremented each time a loop repeats.
- Can be used to control execution of the loop (also known as the **loop control variable**).
- Must be initialized before entering loop.

```
Microsoft Visual Studio Debug Console
Number Number Squared
-------------------------
1               1
2               4
3               9
4               16
5               25
6               36
7               49
8               64
9               81
10              100
```

```
// This program displays a list of numbers and
// their squares.
#include <iostream>
using namespace std;

int main()
{
    const int MIN_NUMBER = 1,   // Starting number to square
        MAX_NUMBER = 10;  // Maximum number to square

    int num = MIN_NUMBER;       // Counter

    cout << "Number Number Squared\n";
    cout << "-------------------------\n";
    while (num <= MAX_NUMBER)
    {
        cout << num << "\t\t" << (num * num) << endl;
        num++; //Increment the counter.
    }
    return 0;
}
```

DOUGLAS COLLEGE

The `do-while` Loop

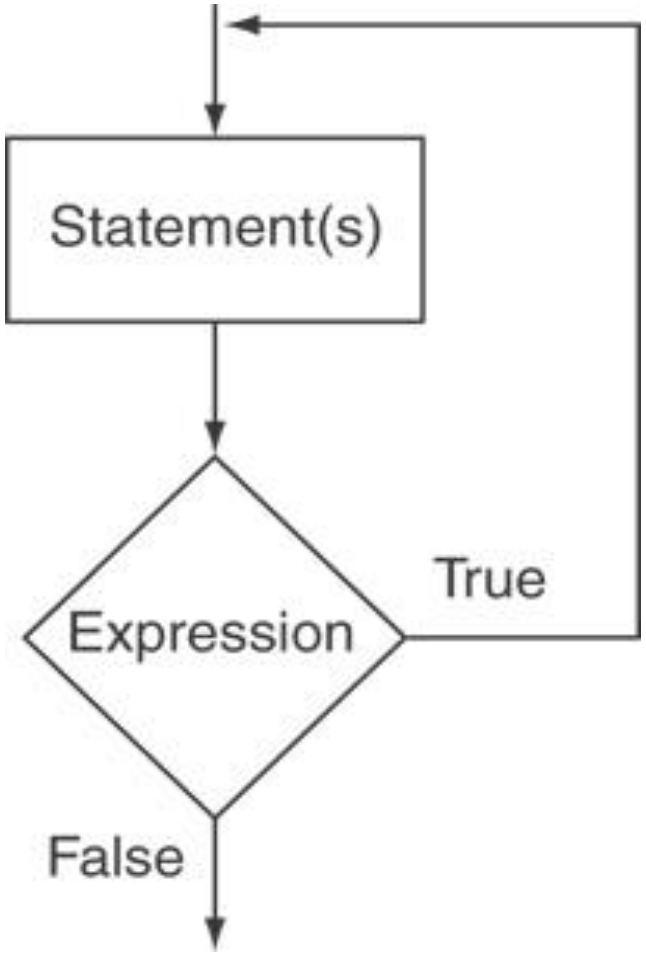
The do-while Loop

- **do-while**: a posttest loop – execute the loop, then test the expression.
- General syntax:

```
do
        statement;  // or block in { }
while (expression);
```

- Note that a semicolon is required after **(expression)**.
- Example:

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



The do-while Loop

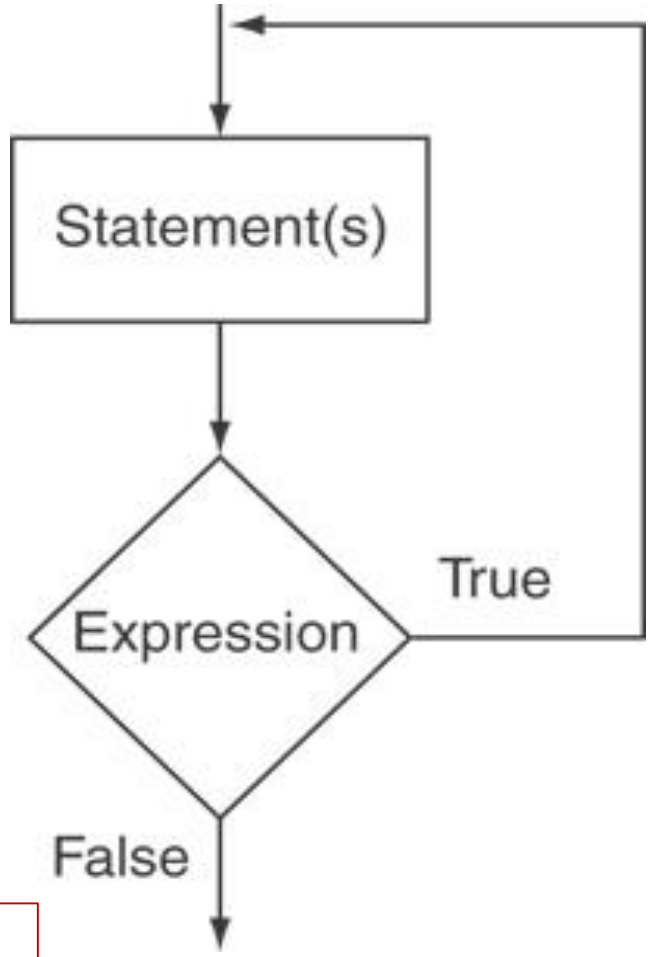
- **do-while**: a posttest loop – execute the loop, then test the expression.
- General syntax:

```
do
        statement;  // or block in { }
while (expression);
```

- Note that a semicolon is required after **(expression)**.
- Example:

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

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



Example

```
// This program averages 3 test scores. It repeats as
// many times as the user wishes.
#include <iostream>
using namespace std;

int main()
{
    int score1, score2, score3; // Three scores
    double average;             // Average score
    char again;                 // To hold Y or N input

    do
    {
        // Get three scores.
        cout << "Enter 3 scores and I will average them: ";
        cin >> score1 >> score2 >> score3;

        // Calculate and display the average.
        average = (score1 + score2 + score3) / 3.0;
        cout << "The average is " << average << ".\n";

        // Does the user want to average another set?
        cout << "Do you want to average another set? (Y/N) ";
        cin >> again;
    } while (again == 'Y' || again == 'y');
    return 0;
}
```

DOUGLAS COLLEGE

The `for` Loop

The for Loop

- Sometimes we know the exact number of iterations that a loop must perform.
- A loop that repeats a specific number of times is known as a **count-controlled loop**.
- The **for** loop is specifically designed to initialize, test, and update a counter variable.
- General syntax:

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

- Note there is no semicolon after the update expression or after the left bracket.
- How it works:
  1. Perform **initialization**
  2. Evaluate **test** expression
  3. If **true**, execute **statement** or block
  4. If **false**, terminate loop execution
  5. Execute update, then re-evaluate test expression

The for Loop – Example

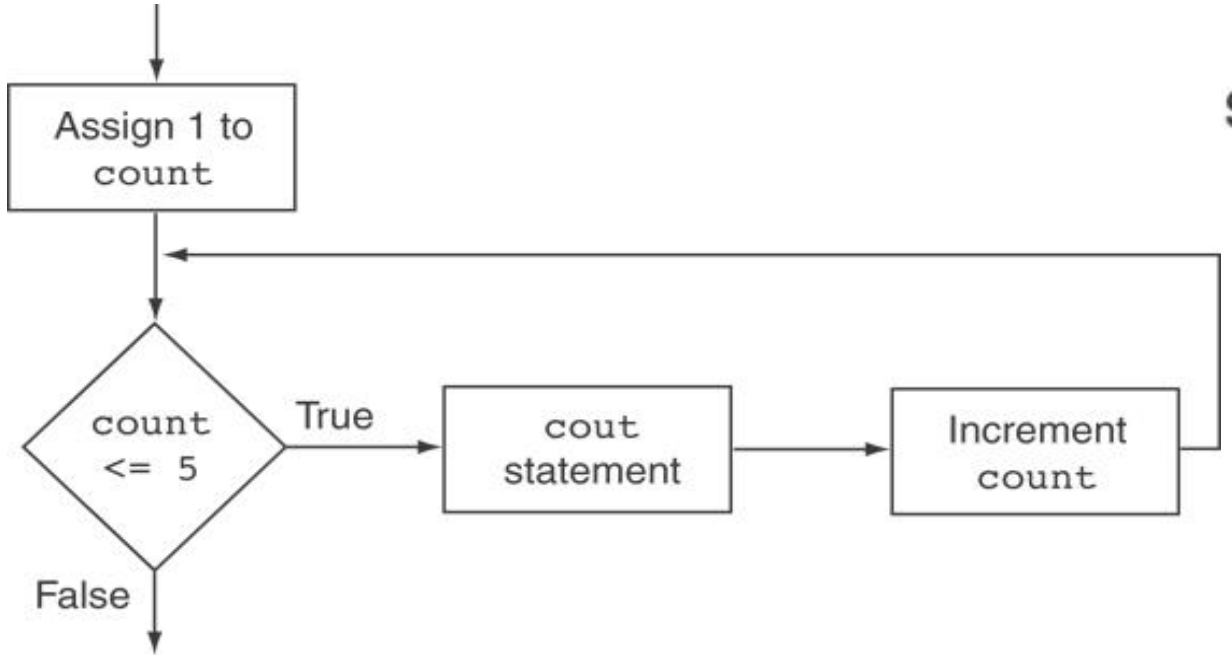
Step 1: Perform the initialization expression.

Step 2: Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.

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

Step 3: Execute the body of the loop.

Step 4: Perform the update expression, then go back to Step 2.



Example

```
// This program displays the numbers 1 through 10 and
// their squares.
#include <iostream>
using namespace std;

int main()
{
    const int MIN_NUMBER = 1,   // Starting value
        MAX_NUMBER = 10;  // Ending value
    int num;

    cout << "Number Number Squared\n";
    cout << "-------------------------\n";

    for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
        cout << num << "\t\t" << (num * num) << endl;

    return 0;
}
```

Step 1: Perform the initialization expression.

Step 2: Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.

Step 4: Perform the update expression, then go back to Step 2.

```
for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
   cout << num << "\t\t" << (num * num) << endl;
```

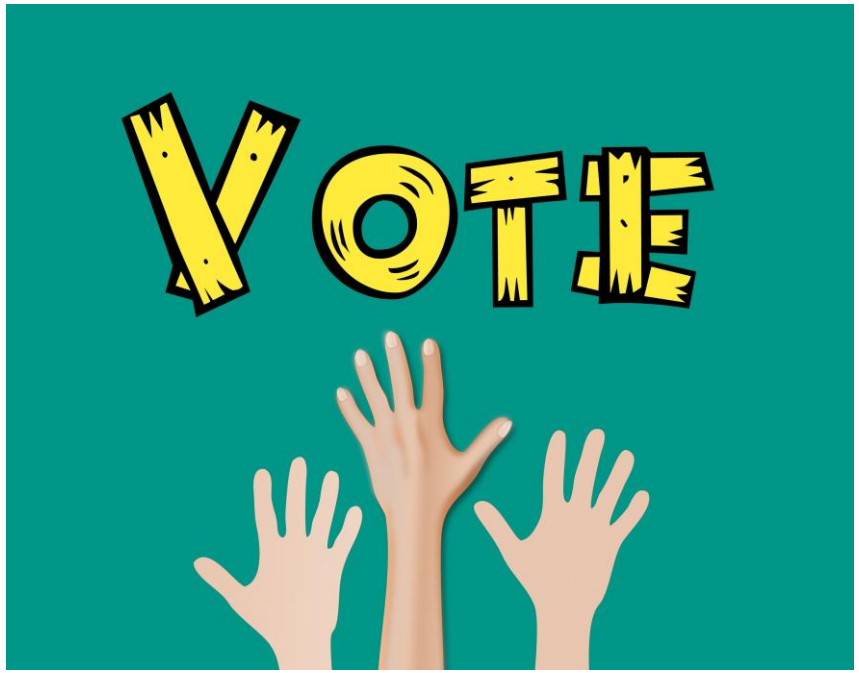
Step 3: Execute the body of the loop.

Poll 3 (Extra Credit)

What will the following loop ever execute?

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

a) Yes!
b) No!



Please use the "Poll" window to participate for extra credit! One answer only please!

for Loop Modifications

- We can have multiple statements in the initialization expression.
- Just 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

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

Semicolon still required after the opening bracket!

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

for Loop Modifications

- We can declare one or more variables in the initialization expression.
- Just separate the statements with a comma:

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

Keeping a Running Total

- **Running total**: accumulated sum of numbers from each repetition of loop.
- **Accumulator**: variable that holds running total.

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

int main()
{
    int sum = 0, num = 1; // sum is the
    while (num <= 10) // accumulator
    {
        sum += num;
        num++;
    }
    cout << "Sum of numbers 1 - 10 is "
        << sum << endl;
    return 0;
}
```

Sentinels

- **Sentinel**: a value in a list of values that indicates end of data.
- It is a special value that cannot be confused with a valid value, e.g., **-999** for a test score.
- Sentinels are usually used to terminate input when user may not know how many values will be entered.

```
// This program calculates the total number of points a
// soccer team has earned over a series of games. The user
// enters a series of point values, then -1 when finished.
#include <iostream>
using namespace std;

int main()
{
    int game = 1,   // Game counter
        points,     // To hold a number of points
        total = 0;  // Accumulator

    cout << "Enter the number of points your team has earned\n";
    cout << "so far in the season, then enter -1 when finished.\n\n";
    cout << "Enter the points for game " << game << ": ";
    cin >> points;

    while (points != -1)
    {
        total += points;
        game++;
        cout << "Enter the points for game " << game << ": ";
        cin >> points;
    }
    cout << "\nThe total points are " << total << endl;
    return 0;
}
```

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:

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

- 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.

Example

```
// This program averages test scores. It asks the user for the
// number of students and the number of test scores per student.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int numStudents,  // Number of students
        numTests;     // Number of tests per student
    double total,     // Accumulator for total scores
        average;   // Average test score

 // Set up numeric output formatting.
    cout << fixed << showpoint << setprecision(1);

    // Get the number of students.
    cout << "This program averages test scores.\n";
    cout << "For how many students do you have scores? ";
    cin >> numStudents;

    // Get the number of test scores per student.
    cout << "How many test scores does each student have? ";
    cin >> numTests;

    // Determine each student's average score.
    for (int student = 1; student <= numStudents; student++)
    {
        total = 0;      // Initialize the accumulator.
        for (int test = 1; test <= numTests; test++)
        {
            double score;
            cout << "Enter score " << test << " for ";
            cout << "student " << student << ": ";
            cin >> score;
            total += score;
        }
        average = total / numTests;
        cout << "The average score for student " << student;
        cout << " is " << average << ".\n\n";
    }
    return 0;
}
```

DOUGLAS COLLEGE

Breaking and Continuing A Loop

Breaking Out of a Loop

- Sometimes it's necessary to stop a loop before it goes through all its iterations.
- The **break** statement causes a loop to terminate early.
  - **Avoid using it – makes code harder to understand and debug.**
  - When used in an inner loop, terminates that loop only and goes back to outer loop.

```
// This program raises the user's number to the powers
// of 0 through 10.
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    int value;
    char choice;

    cout << "Enter a number: ";
    cin >> value;
    cout << "This program will raise " << value;
    cout << " to the powers of 0 through 10.\n";
    for (int count = 0; count <= 10; count++)
    {
        cout << value << " raised to the power of ";
        cout << count << " is " << pow(value, count);
        cout << "\nEnter Q to quit or any other key ";
        cout << "to continue. ";
        cin >> choice;
        if (choice == 'Q' || choice == 'q')
            break;
    }
    return 0;
}
```

The `continue` Statement

- The **`continue`** statement causes the current iteration of a loop to end immediately.
- When **`continue`** is encountered, all the statements in the body of the loop that appear after it are ignored, and the loop prepares for the next iteration.
  - In a **`while`** loop, this means the program jumps to the test expression at the top of the loop.
  - In a **`do-while`** loop, the program jumps to the test expression at the bottom of the loop, which determines whether the next iteration will begin.
  - In a **`for`** loop, **`continue`** causes the update expression to be executed, then the test expression to be evaluated.
  - **Avoid using it – makes code harder to understand and debug.**

Example

```
// This program calculates the charges for DVD rentals.
// Every third DVD is free.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int dvdCount = 1;     // DVD counter
    int numDVDs;          // Number of DVDs rented
    double total = 0.0;   // Accumulator
    char current;         // Current release, Y or N

    // Get the number of DVDs.
    cout << "How many DVDs are being rented? ";
    cin >> numDVDs;

    // Determine the charges.
    do
    {
        if ((dvdCount % 3) == 0)
        {
            cout << "DVD #" << dvdCount << " is free!\n";
            continue;  // Immediately start the next iteration
        }
        cout << "Is DVD #" << dvdCount;
        cout << " a current release? (Y/N) ";
        cin >> current;
        if (current == 'Y' || current == 'y')
            total += 3.50;
        else
            total += 2.50;
    } while (dvdCount++ < numDVDs);

    // Display the total.
    cout << fixed << showpoint << setprecision(2);
    cout << "The total is $" << total << endl;
    return 0;
}
```

DOUGLAS COLLEGE

Using Files for Data Storage

Introduction to File Input and Output

- For a program to retain data between the times it is run, we must save the data to disk.
  - Data is usually saved to a file, typically on a computer disk.
  - Saved data can be retrieved and used at a later time.
- In general, there are two types of files
  - **Text files**: contains data that has been encoded as human-readable text.
  - **Binary files**: contains data that has not been converted to human-readable text.
- There are two ways to access data stored in a file:
  - **Sequential access**: file read sequentially from beginning to end, cannot skip ahead.
  - **Direct access**: can jump directly to any piece of data in the file.

Using Files for Data Storage

- The **`<fstream>`** header file defines the data types **`ofstream`** and **`ifstream`**:
  - **`ifstream`** for input from a file.
  - **`ofstream`** for output to a file.

- Define file stream objects and use the **`open()`** member function to open a file for input or output:

```
ifstream inputFile;
inputFile.open("Customers.txt");

ofstream outfile;
outputFile.open("Employees.txt");
```

Opening Files

- When we call an **`ofstream`** object's **`open()`** member function, the specified file will be created.
  - If the specified file already exists, it will be deleted, and a new file with the same name will be created.
- For an **`ifstream`** object, Input file must already exist for **`open()`** to work.
- Sometimes, we need to specify the path to a file path as well as its name.
  - For example, on a Windows system the following statement opens the file C:\data\inventory.txt:

```
inputFile.open("C:\\data\\inventory.txt")
```

  - Recall that two backslashes are needed to represent one backslash in a string literal.
- It is also possible to define a file stream object and open a file in one statement.:

```
ifstream inputFile("Customers.txt");
ofstream outputFile("Employees.txt");
```

Testing for File Open Errors

- We can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");
if (!infile)
{
        cout << "File open failure!";
}
```

- We can also use the fail() member function: **`if (infile.fail())`**
- The **`fail()`** member function returns true when an attempted file operation is unsuccessful.
- When using file I/O, we should always test the file stream object to make sure the file was opened successfully.
- If the file could not be opened, the user should be informed and appropriate action taken by the program.

Writing Data to A File

- We already know how to use the stream insertion operator (**<<**) with the **cout** object to write data to the screen.
- This operator can also be used with ofstream objects to write data to a file!
- Assuming **outputFile** is an **ofstream** object, the following statement demonstrates using the **<<** operator to write a string literal to a file:

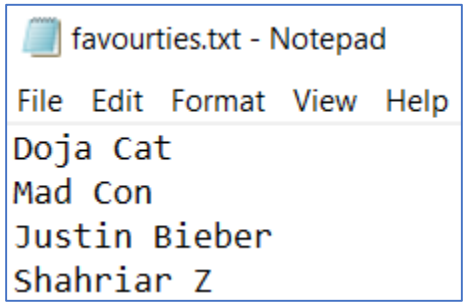
```
outputFile << "I love CMPT-1109!\n";
```

- Here is a statement that writes both a string literal and the contents of a variable to a file:

```
outputFile << "Price: " << price << endl;
```

In-Class Exercise

Write a C++ program that creates and opens a file named "favourites.txt" in the current project directory and writes the following names to the file:



Do not forget to use the `close()` member function to close the file at the end of your program.

In-Class Exercise

```
// This program writes data to a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ofstream outputFile;
    outputFile.open("favourties.txt");

    cout << "Now writing data to the file.\n";

    // Write four names to the file.
    outputFile << "Doja Cat\n";
    outputFile << "Mad Con\n";
    outputFile << "Justin Bieber\n";
    outputFile << "Shahriar Z\n";

    // Close the file
    outputFile.close();
    cout << "Done.\n";
    return 0;
}
```

Reading Data from A File

- Assuming **`inputFile`** is an if **`stream`** object, the following statement shows the **>>** operator reading data from the file into the variable name:

  inputFile >> name;

- When a file has been opened for input, the file stream object internally maintains a special value known as a **read position**.
- A file's read position marks the location of the next byte that will be read from the file.
- When an input file is opened, its read position is initially set to the first byte in the file.
- As data is read from the file, the read position moves forward, toward the end of the file.
- When the **>>** operator extracts data from a file, it expects to read pieces of data that are separated by whitespace characters (spaces, tabs, or newlines).

Example

```
// This program reads data from a file.
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
ifstream inputFile;
string name;

inputFile.open("Friends.txt");
cout << "Reading data from the file.\n";

inputFile >> name;      // Read name 1 from the file
cout << name << endl;   // Display name 1

inputFile >> name;      // Read name 2 from the file
cout << name << endl;   // Display name 2

inputFile >> name;      // Read name 3 from the file
cout << name << endl;   // Display name 3

inputFile.close();      // Close the file
return 0;
}
```

Reading Numeric Data

- When data is stored in a text file, it is encoded as text, using a scheme such as ASCII or Unicode.
  - Even if the file contains numbers, those numbers are stored in the file as a series of characters (e.g., “37”).
- We can still use the >> operator to read data such as this from a text file into a numeric variable, and the >> operator will automatically convert the data to a numeric data type.

```
// This program reads numbers from a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream inFile;
    int value1, value2, value3, sum;

    // Open the file.
    inFile.open("NumericData.txt");

    // Read the three numbers from the file.
    inFile >> value1;
    inFile >> value2;
    inFile >> value3;

    // Close the file.
    inFile.close();

    // Calculate the sum of the numbers.
    sum = value1 + value2 + value3;

    // Display the three numbers.
    cout << "Here are the numbers:\n"
        << value1 << " " << value2
        << " " << value3 << endl;

    // Display the sum of the numbers.
    cout << "Their sum is: " << sum << endl;
    return 0;
}
```

Using Loops to Process Files

- Suppose we need to write a program that displays all of the items in a file, but we do not know how many items the file contains.
- We can open the file and use a loop to repeatedly read an item from the file.
- The **>>** operator not only reads data from a file, but also returns a **`true`** or **`false`** value indicating whether the data was successfully read or not.

```
while (inputFile >> number)
```

  - If the operator returns **`true`**, then a value was successfully read.
  - If the operator returns **`false`**, it means that no value was read from the file.

Example

```
// This program reads data from a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream inputFile;
    int number;

    // Open the file.
    inputFile.open("ListOfNumbers.txt");

    // Read the numbers from the file and
    // display them.
    while (inputFile >> number)
    {
        cout << number << endl;
    }

    // Close the file.
    inputFile.close();
    return 0;
}
```

Thank you.

DOUGLAS COLLEGE

DOUGLAS COLLEGE