V-Ex Tech

# V-EX TECH

## Web Development

Java / Node.js / PHP / .Net / Python

Certification Course

**Assured Placement Program**

**With International Certificate**

### About V-Ex Tech….

**V-Ex Tech** is an elevated education platform providing rigorous industry-relevant programs Designed and delivered on collaboration with industry professionals. It has been constantly Into process of creating an immersive learning experience binding latest technologies, pedagogy and services with enormous job placement opportunities too.

V-Ex Tech

# C++ LANGUAGE

## SYNTEX

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

int main() {
  cout << "Hello World!";
  return 0;
}
```

## What is C++?

C++ is a cross-platform language that can be used to create high-performance applications.

C++ gives programmers a high level of control over system resources and memory.

As C++ is close to C, C# and Java, it makes it easy for programmers to switch to C++ or vice versa.

**Line 1:** `#include <iostream>` is a **header file library** that lets us work with input and output objects, such as `cout` (used in line 5). Header files add functionality to C++ programs.

**Line 2:** `using namespace std` means that we can use names for objects and variables from the standard library.

Don't worry if you don't understand how `#include <iostream>` and `using namespace std` works. Just think of it as something that (almost) always appears in your program.

**Line 3:** A blank line. C++ ignores white space. But we use it to make the code more readable.

**Line 4:** Another thing that always appear in a C++ program, is `int main()`. This is called a **function**. Any code inside its curly brackets `{}` will be executed.

**Line 5:** `cout` (pronounced "see-out") is an **object** used together with the *insertion operator* (`<<`) to output/print text. In our example it will output "Hello World!".

**Note:** Every C++ statement ends with a semicolon `;`.

**Note:** The body of `int main()` could also been written as:

```
int main () { cout << "Hello World! "; return 0; }
```

**Remember:** The compiler ignores white spaces. However, multiple lines makes the code more readable.

**Line 6:** `return 0` ends the main function.

**Line 7:** Do not forget to add the closing curly bracket `}` to actually end the main function.

# Omitting Namespace

You might see some C++ programs that runs without the standard namespace library. The `using namespace std` line can be omitted and replaced with the `std` keyword, followed by the `::` operator for some objects:

```
#include <iostream>

int main() {
  std::cout << "Hello World!";
  return 0;
}
```

You can add as many `cout` objects as you want. However, note that it does not insert a new line at the end of the output:

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

int main() {
  cout << "Hello World!";
  cout << "I am learning C++";
  return 0;
}
```

# New Lines

To insert a new line, you can use the `\n` character:

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

int main() {
  cout << "Hello World! \n";
  cout << "I am learning C++";
  return 0;
}
```

Another way to insert a new line, is with the `endl` manipulator:

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

int main() {
  cout << "Hello World!" << endl;
  cout << "I am learning C++";
  return 0;
}
```

# C++ Variables

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

int main() {
  int myNum = 15;
  cout << myNum;
  return 0;
}
```

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

int main() {
  int myNum;
  myNum = 15;
  cout << myNum;
  return 0;
}
```

## Other Types

```
int myNum = 5;               // Integer (whole number without decimals)
double myFloatNum = 5.99;    // Floating point number (with decimals)
char myLetter = 'D';         // Character
string myText = "Hello";     // String (text)
bool myBoolean = true;
```

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

int main() {
  int myAge = 35;
  cout << "I am " << myAge << " years old.";
  return 0;
}
```

# C++ User Input

`cin` is a predefined variable that reads data from the keyboard with the extraction operator (`>>`).

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

int main() {
  int x;
  cout << "Type a number: "; // Type a number and press enter
  cin >> x; // Get user input from the keyboard
  cout << "Your number is: " << x;
  return 0;
}
```

## Create simple calculator

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

int main() {
  int x, y;
  int sum;
  cout << "Type a number: ";
  cin >> x;
  cout << "Type another number: ";
  cin >> y;
  sum = x + y;
  cout << "Sum is: " << sum;
  return 0;
}
```

# C++ Operators

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

int main() {
  int x = 100 + 50;
  cout << x;
  return 0;
}
```

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

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

int main() {
  int x = 5;
  int y = 3;
  cout << (x > y); // returns 1 (true) because 5 is greater than 3
  return 0;
}
```

# C++ Strings

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

int main() {
  string greeting = "Hello";
  cout << greeting;
  return 0;
}
```

## String Concatenation

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

int main () {
  string firstName = "John ";
  string lastName = "Doe";
  string fullName = firstName + lastName;
  cout << fullName;
  return 0;
}
```

## Adding Numbers and Strings

# C++ String Length

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

int main() {
  string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  cout << "The length of the txt string is: " << txt.length();
  return 0;
}
```

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

int main() {
  string myString = "Hello";
  cout << myString[0];
  return 0;
}
```

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

int main() {
  string txt = "We are the so-called \"Vikings\" from the north.";
  cout << txt;
  return 0;}
```

# User Input Strings

```
string firstName;
cout << "Type your first name: ";
cin >> firstName; // get user input from the keyboard
cout << "Your name is: " << firstName;

// Type your first name: John
// Your name is: John
```

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

int main() {
  string fullName;
  cout << "Type your full name: ";
  getline (cin, fullName);
  cout << "Your name is: " << fullName;
  return 0;
}
```

# C++ Math

```
cout << max(5, 10);
```

```
cout << min(5, 10);
```

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

int main() {
  cout << sqrt(64) << "\n";
  cout << round(2.6) << "\n";
  cout << log(2) << "\n";
  return 0;
}
```

# C++ Conditions and If Statements

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

int main() {
  int x = 20;
  int y = 18;
  if (x > y) {
    cout << "x is greater than y";
  }
  return 0;
}
```

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

int main() {
  int time = 22;
  if (time < 10) {
    cout << "Good morning.";
  } else if (time < 20) {
    cout << "Good day.";
  } else {
    cout << "Good evening.";
```

```
  }
  return 0;
}
```

# Short Hand If...Else (Ternary Operator)

*variable* = (*condition*) ? *expressionTrue* : *expressionFalse*;

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

int main() {
  int time = 20;
  string result = (time < 18) ? "Good day." : "Good evening.";
  cout << result;
  return 0;
}
```

# C++ For Loop

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

int main() {
  for (int i = 0; i < 5; i++) {
    cout << i << "\n";
  }
  return 0;
}
```

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

int main() {
  // Outer loop
  for (int i = 1; i <= 2; ++i) {
    cout << "Outer: " << i << "\n";  // Executes 2 times

    // Inner loop
    for (int j = 1; j <= 3; ++j) {
      cout << " Inner: " << j << "\n";  // Executes 6 times (2 * 3)
    }
  }
  return 0;
}
```

# Break and continue

# C++ Arrays

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

int main() {
  string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
  cout << cars[0];
  return 0;
}
```

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

int main() {
  string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
  cars[0] = "Opel";
  cout << cars[0];
  return 0;
}
```

# Arrays and Loops

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

int main() {
  string cars[5] = {"Volvo", "BMW", "Ford", "Mazda", "Tesla"};
  for (int i = 0; i < 5; i++) {
    cout << cars[i] << "\n";
  }
  return 0;
}
```

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

int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
  }
  return 0;
}
```

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

int main() {
  string cars[5];
  cars[0] = "Volvo";
  cars[1] = "BMW";
  cars[2] = "Ford";
  cars[3] = "Mazda";
```

```
  cars[4] = "Tesla";
  for(int i = 0; i < 5; i++) {
    cout << cars[i] << "\n";
  }
  return 0;
}
```

## Get the Size of an Array

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

int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  cout << sizeof(myNumbers);
  return 0;
}
```

# Multi-Dimensional Arrays

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

int main() {
  string letters[2][4] = {
    { "A", "B", "C", "D" },
    { "E", "F", "G", "H" }
  };

  cout << letters[0][2];
  return 0;
}
```

## Create a Structure

```
struct {              // Structure declaration
  int myNum;          // Member (int variable)
  string myString;    // Member (string variable)
} myStructure;        // Structure variable
```

## One Structure in Multiple Variables

```
struct {
  int myNum;
  string myString;
} myStruct1, myStruct2, myStruct3; // Multiple structure variables
separated with commas
```

```
struct {
  string brand;
  string model;
  int year;
} myCar1, myCar2; // We can add variables by separating them with a comma
here

// Put data into the first structure
myCar1.brand = "BMW";
myCar1.model = "X5";
myCar1.year = 1999;

// Put data into the second structure
myCar2.brand = "Ford";
myCar2.model = "Mustang";
myCar2.year = 1969;

// Print the structure members
cout << myCar1.brand << " " << myCar1.model << " " << myCar1.year << "\n";
cout << myCar2.brand << " " << myCar2.model << " " << myCar2.year << "\n";
```

# C++ Functions

## Create a Function

```
void myFunction() {
  // code to be executed
}
```

- `myFunction()` is the name of the function
- `void` means that the function does not have a return value. You will learn more about return values later in the next chapter

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

   void myFunction() {
     cout << "I just got executed!";
    }

   int main() {
      myFunction();
      return 0;
    }
```

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

void myFunction() {
  cout << "I just got executed!\n";
}

int main() {
  myFunction();
  myFunction();
  myFunction();
  return 0;
}
```

## Parameters and Arguments

```cpp
void functionName(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

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

void myFunction(string fname) {
  cout << fname << " Refsnes\n";
}

int main() {
  myFunction("Liam");
  myFunction("Jenny");
  myFunction("Anja");
  return 0;
}
```

## Default Parameter Value

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

void myFunction(string country = "Norway") {
  cout << country << "\n";
}

int main() {
  myFunction("Sweden");
  myFunction("India");
  myFunction();
  myFunction("USA");
  return 0;
}
```

# C++ Multiple Parameters

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

void myFunction(string fname, int age) {
  cout << fname << " Refsnes. " << age << " years old. \n";
}

int main() {
  myFunction("Liam", 3);
  myFunction("Jenny", 14);
  myFunction("Anja", 30);
  return 0;
}
```

## Return Values

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

int myFunction(int x) {
  return 5 + x;
}

int main() {
  cout << myFunction(3);
  return 0;
}
```

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

int myFunction(int x, int y) {
  return x + y;
}

int main() {
  cout << myFunction(5, 3);
  return 0;
}
```

## Pass By Reference

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

void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}

int main() {
  int firstNum = 10;
  int secondNum = 20;

  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  swapNums(firstNum, secondNum);

  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  return 0;
}
```

## Pass Arrays as Function Parameters

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

void myFunction(int myNumbers[5]) {
  for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
  }
}

int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  myFunction(myNumbers);
  return 0;
}
```

# C++ Function Overloading

```cpp
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

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

int plusFuncInt(int x, int y) {
  return x + y;
}

double plusFuncDouble(double x, double y) {
  return x + y;
}

int main() {
  int myNum1 = plusFuncInt(8, 5);
  double myNum2 = plusFuncDouble(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}
```

# C++ Recursion

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

int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }
}

int main() {
  int result = sum(10);
  cout << result;
  return 0;}
```

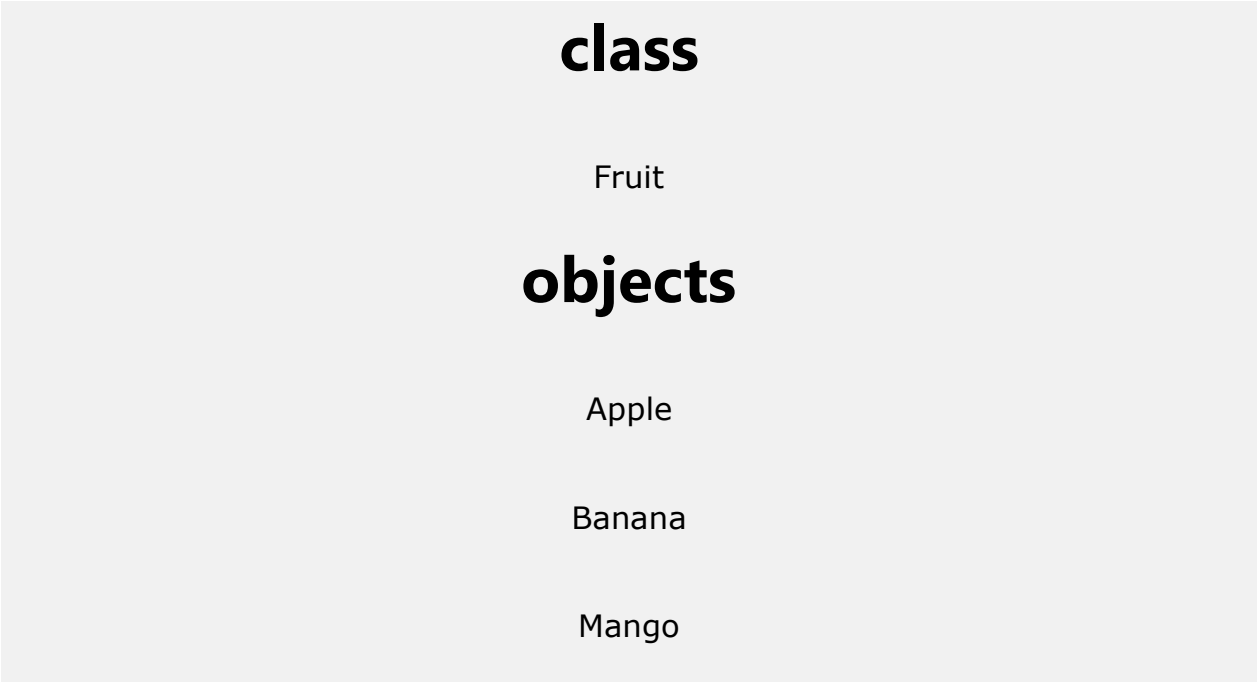
# C++ What is OOP?

OOP stands for Object-Oriented Programming.

# C++ What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:



# Create a Class

```
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};
```

# Create an Object

In C++, an object is created from a class. We have already created the class named `MyClass`, so now we can use this to create objects.

To create an object of `MyClass`, specify the class name, followed by the object name.

To access the class attributes (`myNum` and `myString`), use the dot syntax (`.`) on the object:

## Example

Create an object called "`myObj`" and access the attributes:

```
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";
```

```
  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

# Multiple Objects

You can create multiple objects of one class:

## Example

```
// Create a Car class with some attributes
class Car {
  public:
    string brand;
    string model;
    int year;
};

int main() {
  // Create an object of Car
  Car carObj1;
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  // Create another object of Car
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  // Print attribute values
```

```
  cout << carObj1.brand << " " << carObj1.model << " " <<
carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " <<
carObj2.year << "\n";
  return 0;
}
```

# C++ Class Methods

```
class MyClass {        // The class
  public:              // Access specifier
    void myMethod() {  // Method/function defined inside the class
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

```
class MyClass {        // The class
  public:              // Access specifier
    void myMethod();   // Method/function declaration
};

// Method/function definition outside the class
void MyClass::myMethod() {
  cout << "Hello World!";
}
```

```
int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

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

class Car {
  public:
    int speed(int maxSpeed);
};

int Car::speed(int maxSpeed) {
  return maxSpeed;
}

int main() {
  Car myObj; // Create an object of Car
  cout << myObj.speed(200); // Call the method with an argument
  return 0;
}
```

# C++ Constructors

A constructor in C++ is a **special method** that is automatically called when an object of a class is created.

```
class MyClass {      // The class
  public:            // Access specifier
    MyClass() {      // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass (this will call the constructor)
  return 0;
}
```

```
class Car {         // The class
  public:           // Access specifier
    string brand;  // Attribute
    string model;  // Attribute
    int year;      // Attribute
    Car(string x, string y, int z) { // Constructor with parameters
      brand = x;
      model = y;
      year = z;
    }
};

int main() {
  // Create Car objects and call the constructor with different values
  Car carObj1("BMW", "X5", 1999);
  Car carObj2("Ford", "Mustang", 1969);

  // Print values
  cout << carObj1.brand << " " << carObj1.model << " " <<
carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " <<
carObj2.year << "\n";
  return 0;
}
```

# pattern in c++

```cpp
// C++ code to demonstrate star pattern
#include <iostream>
using namespace std;

// Function to demonstrate printing pattern
void pypart(int n)
{
    // Outer loop to handle number of rows
    // n in this case
    for (int i = 0; i < n; i++) {

        // Inner loop to handle number of columns
        // values changing acc. to outer loop
        for (int j = 0; j <= i; j++) {

            // Printing stars
            cout << "* ";
        }

        // Ending line after each row
        cout << endl;
    }
}

// Driver Function
int main()
{
    int n = 5;
    pypart(n);
    return 0;
}
```

```
*
* *
* * *
* * * *
* * * * *
```

```cpp
// C++ code to demonstrate star pattern
#include <iostream>
using namespace std;

// Driver Code
int main()
{
    int n = 5;

    // looping rows
    for (int i = n; i > 0; i--) {
        for (int j = 1; j <= n; j++) // looping columns
        {
            if (j >= i) {
                cout << "* ";
            }
            else {
                cout << "  ";
            }
        }
        cout << endl;
    }
    return 0;
}
```

```
        *
      * *
    * * *
  * * * *
* * * * *
```

```cpp
// C++ code to demonstrate star pattern
#include <iostream>
using namespace std;

// Function to demonstrate printing pattern
void pypart(int n)
{
    // Outer loop to handle number of rows
    // n in this case
    for (int i = n; i > 0; i--) {

        // Inner loop to handle number of columns
        // values changing acc. to outer loop
        for (int j = 0; j < i; j++) {

            // Printing stars
            cout << "* ";
        }

        // Ending line after each row
        cout << endl;
    }
}

// Driver Function
int main()
{
    int n = 5;
```

```
    pypart(n);
    return 0;
}
```

```
        *
      * *
    * * *
  * * * *
* * * * *
```

```
// C++ code to demonstrate star pattern
#include <iostream>
using namespace std;

// Function to demonstrate printing pattern
void pypart2(int n)
{
        // Number of spaces
        int i, j, k = n;

        // Outer loop to handle number of rows
        // n in this case
        for (i = 1; i <= n; i++) {
```

```
            // Inner loop for columns
            for (j = 1; j <= n; j++) {

                  // Condition to print star pattern
                  if (j >= k)
                        cout << "* ";
                  else
                        cout << " ";
            }
            k--;
            cout << "\n";
      }
}

// Driver Code
int main()
{
      int n = 5;
      // Function Call
      pypart2(n);
      return 0;
}
```

# C++ Files

| | |
|---|---|
| `ofstream` | Creates and writes to files |
| `ifstream` | Reads from files |
| `fstream` | A combination of ofstream and ifstream: creates, reads, and writes to files |

## Create and Write To a File

To create a file, use either the `ofstream` or `fstream` class, and specify the name of the file.

To write to the file, use the insertion operator (`<<`).

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

int main() {
  // Create and open a text file
  ofstream MyFile("filename.txt");

  // Write to the file
  MyFile << "Files can be tricky, but it is fun enough!";

  // Close the file
  MyFile.close();
}
```

# Read a File

To read from a file, use either the `ifstream` or `fstream` class, and the name of the file.

Note that we also use a `while` loop together with the `getline()` function (which belongs to the `ifstream` class) to read the file line by line, and to print the content of the file:

```
#include <iostream>

#include <fstream>

#include <string>

using namespace std;



int main () {

 // Create a text file
```

```
 ofstream MyWriteFile("filename.txt");

 // Write to the file
 MyWriteFile << "Files can be tricky, but it is fun enough!";

 // Close the file
 MyWriteFile.close();

 // Create a text string, which is used to output the text file
 string myText;

 // Read from the text file
 ifstream MyReadFile("filename.txt");

 // Use a while loop together with the getline() function to read the file line by line
 while (getline (MyReadFile, myText)) {
   // Output the text from the file
   cout << myText;
 }
```

```
  // Close the file
  MyReadFile.close();
}
```