C++ Language

C++ is a statically typed, compiled, case-sensitive programming language that supports procedural, object-oriented, and generic programming. C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983. C++ is a superset of C, i.e. virtually any legal C program is a legal C++ program.

1) Preliminaries

You need the following two software tools available on your computer :

(a) Text Editor (examples of few a editors include Windows Notepad, OS Edit command, EMACS, and vim or vi) and

(b) The C++ Compiler (g++)

- check whether g++ is installed on your system by entering the following command from the command line : `$ g++ -v`
- If the g++ is missing, you need to install it. In **Windows**, you'll need a Unix-like environment such as Cygwin, and install the packages "Devel>gcc/g++: C++ Compiler" and "Devel>gdb: The GNU Debugger". In **Linux**(Debian/Ubuntu distribution), run the command :

sudo apt-get install build-essential.

If you use your personal machine, G++ is not installed, and you don't want to install it, try the exercises using the on-line compiler available at :

- **CodingGround** (https://www.tutorialspoint.com/codingground.htm)
- Scroll down until « Online IDEs »
- Choose C++ among the options
- You can then write your C++ code, compile and execute it. The result is displayed in the terminal zone, at the bottom of the page.
- If you use CodingGround, please save your exercices in .c files so that the professor can validate your solutions.

2) Hello Word in C++

Compile and Execute C Program

- Open a text editor and add the following code.

```
#include <iostream>
using namespace std;
int main() {
   // prints Hello World
   cout << "Hello World";
   return 0;
}
```

- Save the file as *hello.cpp*
- Open a command prompt and go to the directory where you have saved the file.
- Type *g++ hello.cpp* and press enter to compile your code.
- If there are no errors in your code, the command prompt will take you to the next line and would generate *a.exe* executable file.

- Now, if you're in Linux, type *./a* to execute your program. If you're in Windows, type *a.exe*
- You will see the output *"Hello World"* printed on the screen.

3) Using the NetBeans IDE

In this Review, we will use the NetBeans IDE. Normally the machines in the university are already equipped with NetBeans. If you want to install in your personal machine, you'll need :

- to download it in https://netbeans.org/downloads/
- choose the Download button under "All" column.

Creating the C++ Project for the exercises

- In NetBeans, create a new C++ Project in File>New Project
- Set it as follows :
  - Category: C/C++, Project: C/C++ Application
  - Next button
  - Project Name: Review_Cpp
  - Project Location: A folder in the network that you'll create for the C++ Review
- A project is created. Add your source code in the folder "Source Files".
- For instance, past the "Hello Word" example above inside the "main.cpp" file.
- To execute it, Run->Run Project

For the following exercises, you have several options: either you always use the name "main.cpp" file above, and you comment each exercise once it is completed; or you create a function for each exercise (inside the "main.cpp" file) and you call it in the "main" function.

4) C++ – the basics

- The source files for C++ programs are typically named with the extension « .cpp »
- C++ fully supports object-oriented programming
- In a C++ program, each individual **statement** must be ended with a **semicolon**.
- Like C, one-line comments begin with //, and multi-line comments should appear between the delimiters /* and */.

C++ Program Structure

Let us look at a simple code that would print the words *Hello World*.

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.
- The line **using namespace std;** tells the compiler to use the std namespace.
- The next line '**//prints Hello World**' is a single-line comment.
- The line **int main()** is the main function where program execution begins.
- The next line **cout << "Hello World.";** causes the message to be displayed on the screen.
- The next line **return 0;** terminates main( ) function and causes it to return the value 0 to the calling process.

Namespaces in C++ and the Scope Resolution Operator ::

Namespaces provide a method for preventing name conflicts in large projects. Symbols declared inside a namespace block are placed in a named scope that prevents them from being mistaken for identically-named symbols in other scopes.

In each scope, a name can only represent one entity. So, there cannot be two variables with the same name in the same scope. Using namespaces, we can create two variables or member functions having the same name.

A namespace definition begins with the keyword **namespace** followed by the namespace name. The **scope resolution operator** :: (also called "to prepend the namespace") is used to identify and disambiguate identifiers used in different scopes. Example :

```
#include <iostream>
using namespace std;
// first name space
namespace first_space {
   void func() {
      cout << "Inside first_space" << endl;
   }
}
// second name space
namespace second_space {
   void func() {
      cout << "Inside second_space" << endl;
   }
}
int main () {
   // Calls function from first name space.
   first_space::func();
   // Calls function from second name space.
   second_space::func();
   return 0;
}
```

The output will be :

```
Inside first_space
Inside second_space
```

You can also avoid prepending of namespaces with the **using namespace** directive. This directive tells the compiler that the subsequent code is making use of names in the specified namespace:

```
#include <iostream>
using namespace std;
// first name space
namespace first_space {
   void func() {
      cout << "Inside first_space" << endl;
   }}
// second name space
namespace second_space {
   void func() {
      cout << "Inside second_space" << endl;
   }}
using namespace first_space;
int main () {
   // This calls function from first name space.
   func();
   return 0;}
```

A scope resolution operator without a scope qualifier refers to the global namespace. Example :

```
namespace NamespaceA{
    int x;
}
int x;
int main() {
    int x;
    // the x in main()
    x = 0;
    // The x in the global namespace
    ::x = 1;
    // The x in the A namespace
    NamespaceA::x = 2;
}
```

5) C++ Classes and Objects

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming. A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package.

A **class definition** starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows

```
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

The keyword **public** determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. We declare **objects** of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box −

```
Box Box1;          // Declare Box1 of type Box
Box Box2;          // Declare Box2 of type Box
```

The public data members of objects of a class can be **accessed** using the direct member access operator (.). Let us try the following example to make the things clear −

```
#include <iostream>
using namespace std;
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};

int main() {
   Box Box1;        // Declare Box1 of type Box
   Box Box2;        // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;

   // box 2 specification
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;

   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.height * Box2.length * Box2.breadth;
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

It is important to note that private and protected members can not be accessed directly using direct member access operator (.). You need to create public functions to access private data of the class.

A **class constructor** is a special member function of a class that is executed whenever we create new objects of that class. A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables. A default constructor does not have any parameter, but if you need, a constructor can have **parameters**. This helps you to assign initial value to an object at the time of its creation.

A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Following example explains the concepts of constructor and destructor −

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

class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line(double len);  // This is the constructor declaration
      ~Line();  // This is the destructor declaration
   private:
      double length;
};

// Member functions definitions including constructor
Line::Line( double len) {
   cout << "Object is being created, length = " << len << endl;
   length = len;
}
Line::~Line(void) {
   cout << "Object is being deleted" << endl;
}
void Line::setLength( double len ) {
   length = len;
}
double Line::getLength( void ) {
   return length;
}

// Main function for the program
int main() {
   Line line(10.0);

   cout << "Length of line : " << line.getLength() <<endl;

   // set line length again
   line.setLength(6.0);
   cout << "Length of line : " << line.getLength() <<endl;

   return 0;
}
```

A **friend function** of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions. To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend** as follows :

```
class Box {
   double width;

   public:
      double length;
      friend void printWidth( Box box );
      void setWidth( double wid );
};
```

Consider the following program :

```
#include <iostream>
using namespace std;
class Box {
   double width;
   public:
      friend void printWidth( Box box );
      void setWidth( double wid );
};

// Member function definition
void Box::setWidth( double wid ) {
   width = wid;
}

// Note: printWidth() is not a member function of any class.
void printWidth( Box box ) {
   // Because printWidth() is a friend of Box, it can directly access any member of this
class
   cout << "Width of box : " << box.width <<endl;
}

// Main function for the program
int main() {
   Box box;
   // set box width without member function
   box.setWidth(10.0);
   // Use friend function to print the wdith.
   printWidth( box );
   return 0;
}
```

When the above code is compiled and executed, it produces the following result :

```
Width of box : 10
```

For the following exercises, you can refer to the **tutorial** available in https://www.tutorialspoint.com/cplusplus/index.htm , or any other tutorial available on-line.

6) Exercises

Writing – Write a C++ program to declare a variable *age,* initializes it, and print the following lines. To display the text on the screen you can use cout<<

You are 10 years old.

You are too young to play the game.

Reading – Write a C++ program to prompt the user to input her/his name and print this name on the screen, as shown below. The text from keyboard can be read by using cin>>. There is no native type for strings in C++. You may use an array of characters.

Math – Given the following pseudo code, write a program that executes it.

a. read x
b. read y
c. compute p=x*y
d. compute s=x+y
e. total=s2+p*(s-x)*(p+y)
f. print total

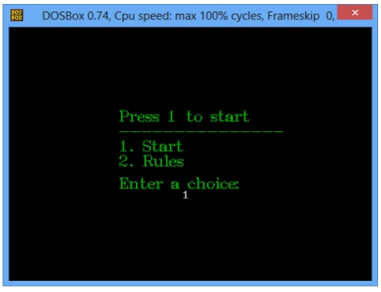
Calculator – Write a C++ program that will display the calculator menu. The program will prompt the user to choose the operation choice (from 1 to 5). Then it asks the user to input two integer vales for the calculation. **Use Functions**. See the sample below.

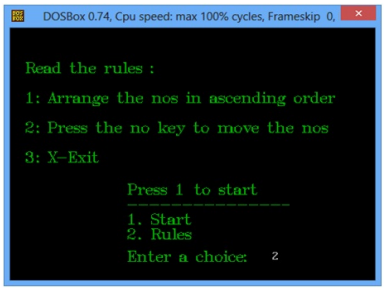
MENU
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus

Enter your choice: 1
Enter your two numbers: 12 15
Result: 27
Continue? y

Printing – Print (on the screen) 0 - 100 and then 100-0 using one function without loops. (cout<<"0 1 2...100 100...0" doesn't count !).

Shuffle Game – Shuffle Game is a program to shuffle the numbers. The numbers predefined from 1-8. Initially all the numbers are shuffled and displayed in random order. Arrange the numbers in ascending order from left to right. The gamer can input any number which is either left, right, top or down to the empty button to move that number.





C++ Classes and Objects

Before doing these exercises, read the section "5 - C++ Classes and Objects", and try out the examples.

Rectangle – Write a program that defines a rectangle class whose constructor takes 2 parameters, width and height, and offers the following functions :

Calculate the Perimeter
Calculate the Surface
Displays the rectangle using '*'. Example for a 6x2 rectangle :

******
******

Counter Class – We want to implement a class representing an integer counter. Such an object is characterized by:

- An integer value, initially 0, which can be positive or zero.
- It can only vary in steps of 1 (increment or decrement). Decrementation of a zero counter has no effect.

Create a Counter class to render the requested service. Write a small test program which:
1. create a counter and display its value;
2. increment it 10 times, then display its value again;
3. Decrement it 20 times, then display its value three times

The output of this program must give (something like) "0 10 0"

Point class with friend functions – Create a Point class to represent points in a plane characterized by two coordinates, x and y. Define the following **member functions**:

- contructor
- display: display the Cartesian coordinates of the point.

Define the following **friend functions**:

- compare: checks if two points are identical.
- distance: calculates the distance between two points.
- middle: Returns the middle point of the two-point segment.