National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

## Department of Computing

# [CS212]: [Object Oriented Programming]

## Class: [BSCS-5AB]

### Lab [2]: [Objects & Classes - Introduction]

## Date: [Feb 15, 2016]

## Time: [9-12, 2-5]

**Instructor: [Hirra Anwar]**

**Lab Engineer: [Iram Tariq]**

# Lab [2]: Objects & Classes - Introduction

**Introduction**

Uptill now, the traditional approach to write programs has been used i.e. procedural programs. You will be creating the same program using the procedural and object oriented approaches. This will help in understanding the need for the newer approach.

In addition to this, you will write your first C++ program and define classes using the standard syntax and define the objects of those classes.

**Objectives**

Solve the given problem using procedural approach as well as object oriented approach.

**Tools/Software Requirement**

You will need Visual Studio 2010/2012.

**Description**

## Structure of a Program

The following Hello World program contains all the fundamental components C++ programs have:

```
1 // my first program in C++
2 #include <iostream>
3
4 int main()
5 {
6   std::cout << "Hello World!";
7 }
```

```
Hello World!
```

The left panel above shows the C++ code for this program. The right panel shows the result when the program is executed by a computer. The grey numbers to the left of the panels are line numbers to make discussing programs and researching errors easier. They are not part of the program.

Let's examine this program line by line:

**Line 1: `// my first program in C++`**

Two slash signs indicate that the rest of the line is a comment inserted by the programmer but which has no effect on the behavior of the program. Programmers use them to include short explanations or observations concerning the code or program. In this case, it is a brief introductory description of the program.

**Line 2:** `#include <iostream>`

Lines beginning with a hash sign (`#`) are directives read and interpreted by what is known as the *preprocessor*. They are special lines interpreted before the compilation of the program itself begins. In this case, the directive `#include <iostream>`, instructs the preprocessor to include a section of standard C++ code, known as *header iostream*, that allows to perform standard input and output operations, such as writing the output of this program (`Hello World`) to the screen.

**Line 3: A blank line.**

Blank lines have no effect on a program. They simply improve readability of the code.

**Line 4:** `int main ()`

This line initiates the declaration of a function. Essentially, a function is a group of code statements which are given a name: in this case, this gives the name "main" to the group of code statements that follow. The function named `main` is a special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the `main` function, regardless of where the function is actually located within the code.

**Lines 5 and 7:** `{` **and** `}`

The open brace (`{`) at line 5 indicates the beginning of `main`'s function definition, and the closing brace (`}`) at line 7, indicates its end. Everything between these braces is the function's body that defines what happens when `main` is called. All functions use braces to indicate the beginning and end of their definitions.

**Line 6:** `std::cout << "Hello World!";`

This line is a C++ statement. A statement is an expression that can actually produce some effect. It is specifying its actual behavior. Statements are executed in the same order that they appear within a function's body.

This statement has three parts: First, `std::cout`, which identifies the **st**andar**d** **c**haracter **out**put device (usually, this is the computer screen). Second, the insertion operator (`<<`), which indicates that what follows is inserted into `std::cout`. Finally, a sentence within quotes ("Hello world!"), is the content inserted into the standard output.

Notice that the statement ends with a semicolon (`;`). All C++ statements must end with a semicolon character. One of the most common syntax errors in C++ is forgetting to end a statement with a semicolon.

You may have noticed that not all the lines of this program perform actions when the code is executed. There is a line containing a comment (beginning with `//`). There is a line with a directive for the preprocessor (beginning with `#`). There is a line that defines a function (in this case, the `main` function). And, finally, a line with a statements ending with a semicolon (the insertion into `cout`), which was within the block delimited by the braces (`{ }`) of the `main` function.

In C++, the separation between statements is specified with an ending semicolon (`;`), with the separation into different lines not mattering at all for this purpose. Many statements can be written in a single line, or each statement can be in its own line. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it, but has no effect on the actual behavior of the program.

Now, let's add an additional statement to our first program:

```
1 // my second program in C++
2 #include <iostream>
3
4 int main ()
5 {
6   std::cout << "Hello World! ";
7   std::cout << "I'm a C++ program";
8 }
```

```
Hello World! I'm a C++ program
```

In this case, the program performed two insertions into `std::cout` in two different statements. Once again, the separation in different lines of code simply gives greater readability to the program, since `main` could have been perfectly valid defined in this way:

```
int main () { std::cout << " Hello World! "; std::cout << " I'm a C++
program "; }
```

The source code could have also been divided into more code lines instead:

```
int main ()
{
  std::cout <<
    "Hello World!";
  std::cout
    << "I'm a C++ program";
}
```

And the result would again have been exactly the same as in the previous examples.

## Comments

As noted above, comments do not affect the operation of the program; however, they provide an important tool to document directly within the source code what the program does and how it operates.

C++ supports two ways of commenting code:

```
// line comment
/* block comment */
```

The first of them, known as *line comment*, discards everything from where the pair of slash signs (`//`) are found up to the end of that same line. The second one, known as *block comment*, discards everything between the `/*` characters and the first appearance of the `*/` characters, with the possibility of including multiple lines.

Let's add comments to our second program:

```
/* my second program in C++
   with more comments */

#include <iostream>

int main ()
{
  std::cout << "Hello World! ";     // prints Hello
World!
```

```
Hello World! I'm a C++ program
```

```
10   std::cout << "I'm a C++ program"; // prints I'm a
C++ program
}
```

If comments are included within the source code of a program without using the comment characters combinations `//`,`/*` or `*/`, the compiler takes them as if they were C++ expressions, most likely causing the compilation to fail with one, or several, error messages.

## Using namespace std

If you have seen C++ code before, you may have seen `cout` being used instead of `std::cout`. Both name the same object: the first one uses its *unqualified name* (`cout`), while the second qualifies it directly within the *namespace* `std` (as`std::cout`). `cout` is part of the standard library, and all the elements in the standard C++ library are declared within what is a called a *namespace*: the namespace `std`.

In order to refer to the elements in the `std` namespace a program shall either qualify each and every use of elements of the library (as we have done by prefixing `cout` with `std::`), or introduce visibility of its components. The most typical way to introduce visibility of these components is by means of *using declarations*:

```
using namespace std;
```

The above declaration allows all elements in the `std` namespace to be accessed in an *unqualified* manner (without the`std::` prefix).

With this in mind, the last example can be rewritten to make unqualified uses of `cout` as:

```
1  // my second program in C++
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    cout << "Hello World! ";
8    cout << "I'm a C++ program";
9  }
```

```
Hello World! I'm a C++ program
```

Both ways of accessing the elements of the `std` namespace (explicit qualification and *using* declarations) are valid in C++ and produce the exact same behavior.

**Class Structure:**

The basic structure of a class is:

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
}
```

Consult lecture slides for detailed description regarding Class structure & objects.

# Lab Tasks

**Task-1:**

Bicycle is a real world object. In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components.

From the real world understanding of an object bicycle, create a Class which acts as a template of bicycle object. Your class must have:

a. Data members
b. Member functions (implement functionality of applybrakes, increaseSpeed, decreaseSpeed and more)
    - Constructor
    - Relevant Functions

Create instances of the class and invoke functions accordingly and test various access specifiers in your code.

**Task-2:**

Write a program to implement the Date class which has three data members: day, month and year. Provide two constructors, one which takes arguments and another one which does not (you may set values of data members to 0). Write getter and setter functions (input and show functions) for the class. Write a member function pastDate() which tells whether the date is the past date or not.

In main program:

- Create two objects using two different constructors.
- Call setter and getter functions on those objects.
- Call the pastDate() function.

**Task-3:**

Develop a calculator application. The application should be able to do basic as well as some scientific calculations.

a. Implement using procedural approach
b. Implement using object oriented approach

**Task-4:**

You can convert temperature from degrees Celsius to degrees Fahrenheit by multiplying by 9/5 and adding 32. Write a program that allows the user to enter a floating-point number representing degrees Celsius, and then displays the corresponding degrees Fahrenheit and vice-versa.

a. Implement using procedural approach
b. Implement using object oriented approach

**Instructions**

You are encouraged to use good programming conventions by entering appropriate comments, using indentations, and using descriptive variable names in your programs

**Deliverables**

Upload the zip folder of .cpp files on LMS with all lab tasks within lab time. Show your work to the Lab Engineer and make sure it gets graded.