Object Oriented Programming
Lab 02
Classes and Objects
Version 1.0

In this particular lab session, as you are very new to the OOP, the very basics and fundamentals of OOP will be revealed towards you. With this session, you will be able to know various things about classes and objects.

A *class* is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions.

An *object* is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

Class declaration:

Classes are generally declared using the keyword `class`, with the following format:

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

Where `class_name` is a valid identifier for the class. The body of the declaration can contain members, that can be either data or function declarations, and optionally access specifiers.

All is very similar to the declaration on data structures, except that we can now include also functions and members, but also this new thing called *access specifier*. An access specifier is one of the following three keywords: `private`, `public` or `protected`. These specifiers modify the access rights that the members following them acquire:

1. `private` members of a class are accessible only from within other members of the same class or from their *friends*.

2. `protected` members are accessible from members of their same class and from their friends, but also from members of their derived classes.

3. Finally, `public` members are accessible from anywhere where the object is visible.

By default, all members of a class declared with the `class` keyword have private access for all its members. Therefore, any member that is declared before one other class specifier automatically has private access. For example:

```
class CRectangle {
int x,y;
public:
void set_values (int,int);
int area (void);
};
```

Declares a class (i.e., a type) called `CRectangle`. This class contains four members: two data members of type `int` (member `x` and member `y`) with private access (because private is the default access level) and two member functions with public access: `set_values()` and `area()`, of which for now we have only included their declaration, not their definition.

Member function definition (outside the class):

The following is the format for defining a member function:
ret-type class-name :: function-name(argument-list)
{
  Function body
}

Creating objects:

Once a class has been declared, we can create variables of that type by using the class name(like any other built-in type variable). The following is the format for creating an object:
class_name object_name;

Accessing class members:

As pointed out earlier, the private data of a class can be accessed only through the member functions of that class. The following is the format for calling a member function:

object_name.function_name(actual_argument);

Here dot(.) operator is used to access the public members of class.

Here is the complete example of class CRectangle:

```
// classes example
#include <iostream>
using namespace std;
class CRectangle {
int x, y;
public:
void set_values (int,int);
int area () {return (x*y);}
};
void CRectangle::set_values (int a, int
b) {
x = a;
y = b;
}
int main () {
CRectangle rect;
rect.set_values (3,4);
cout << "area: " << rect.area();
return 0;
}
```

```
// output
area: 12
```

The most important new thing in this code is the operator of scope (::, two colons) included in the definition of `set_values()`. It is used to define a member of a class from outside the class definition itself. You may notice that the definition of the member function `area()` has been included directly within the definition of the `CRectangle` class given its extreme simplicity, whereas `set_values()` has only its prototype declared within the class, but its definition is outside it. In this outside declaration, we must use the operator of scope (::) to specify that we are defining a function that is a member of the class `CRectangle` and not a regular global function.

The scope operator (::) specifies the class to which the member being declared belongs, granting exactly the same scope properties as if this function definition was directly included within the class definition. For example, in the function `set_values()` of the previous code, we have been able to use the variables `x` and `y`, which are private members of class `CRectangle`, which means they are only accessible from other members of their class.

The only difference between defining a class member function completely within its class or to include only the prototype and later its definition, is that in the first case

the function will automatically be considered an inline member function by the compiler, while in the second it will be a normal (not-inline) class member function, which in fact supposes no difference in behavior.

Members x and y have private access (remember that if nothing else is said, all members of a class defined with keyword class have private access). By declaring them private we deny access to them from anywhere outside the class. This makes sense, since we have already defined a member function to set values for those members within the object: the member function set_values(). Therefore, the rest of the program does not need to have direct access to them. Perhaps in a so simple example as this, it is difficult to see an utility in protecting those two variables, but in greater projects it may be very important that values cannot be modified in an unexpected way (unexpected from the point of view of the object).

One of the greater advantages of a class is that, as any other type, we can declare several objects of it. For example, following with the previous example of class CRectangle, we could have declared the object rectb in addition to the object rect:

```
// example: one class, two objects
#include <iostream>
using namespace std;
class CRectangle {
int x, y;
public:
void set_values (int,int);
int area () {return (x*y);}
};
void CRectangle::set_values (int a, int
b) {
x = a;
y = b;
}
int main () {
CRectangle rect, rectb;
rect.set_values (3,4);
rectb.set_values (5,6);
cout << "rect area: " << rect.area() <<
endl;
cout << "rectb area: " << rectb.area() <<
endl;
return 0;
}
```

```
// output
rect area: 12
rectb area: 30
```