Universidad Carlos III de Madrid

Industrial Informatics I Lab Session

Bachelor in Industrial Electronics and Automation

2016-2017

Lab 2

Teaching staff:

Juan Carlos Gonzalez Victores

Mohamed Abderrahim

Lab 2 - Classes

1. Classes

1.1. Definition

The first thing you are going to see is the structure of the `main.cpp` file in C++.

```
/////////////////////////////////////////////////////
// main.cpp
/////////////////////////////////////////////////////

#include <iostream>
#include “point.h”

using namespace std;
using namespace space;

int main(int argc, char **argv) {
      Point P;
      cout << P.getX() << endl;
      cout << P.getY() << endl;

      P.set(10, 10);
      cout << P.getX() << endl;
      cout << P.getY() << endl;

      P.display();
}
```

In this program, the first thing you do is include the standard library of C++ and the header file `point.h`, which contains the declaration of the class that will be created later. Note that standard library is not included in the same manner as done so in C, for example, as in the case of `iostream.h`. In C++, most of the standard library files end without `.h` and sometimes starting with the letter `“c”.`

Some examples of C include files equivalent to that of C++ are:

```
#include <stdlib.h>    →    #include <cstdlib>
#include <assert.h>    →    #include <cassert>
```

Then, you can find the directives:

```
using namespace std;
using namespace space;
```

These tell the compiler to use namespaces `std` (standard) and `space` (created by the user below).

In C++, classes can be encapsulated into **namespaces**. This is useful for better organization in larger projects. The namespace `std` defines the standard namespace of C++. If not explicitly stated to be used, the compiler will give an error, for example while trying to use `cout`, since this is a class under the namespace `std`. If you do not specify the namespace to be used, and do not want to get an error during compilation, you need to explicitly mention the namespace every time a function under it is called, like `std::cout`.

In the `main` function, observe the following statement:

```
Point P;
```

This is telling the compiler to create a variable of type `Point`. That is, you are going to create a variable of class `Point`, or it could also be said as creating an object of class `Point`.

As already seen during the lecture, calls to methods of a class can be done using the operator "`.`" (Point). Therefore, the expression `P.getX()` refers to the call of member function `getX()` of object `P`.

Consider the following code:

```
Point P, P2;
cout << P.getX() << endl;
cout << P2.getX() << endl;
```

It creates two different objects, both belonging to class `Point`. `P.getX()` is a call to the member function of the object, which is different compared to the call `p2.getX()`. The difference, as will be seen later, is that `P.getX()` will use the values of the object `P`, whereas `P2.getX()` would use the values of object `P2`.

At this point, you will see how to define a class. A class is defined in two files, a `.h` file and a `.cpp` file. The `.h` file contains the *definition* or *declaration* of the class, while the `.cpp` file contains the class *implementation*.

Classes can be defined within a namespace "`namespace`". The namespaces starts with a *lower case letter* (**by convention**). The class name begins with a *capital letter* (**by convention**). The class `Point` represents a point in space. This class can handle a point through member functions or class methods.

Classes are defined using the keyword `class`. Variables and functions, within a class definition, can be either defined as *Public*, which means that they can be accessed from other classes and functions, or as *Private*, which means that they can be only be accessed from within member functions of the same class.

The following code snippet shows the file `point.h`.

```
/////////////////////////////////////////////////////
// point.h
/////////////////////////////////////////////////////

#ifndef _POINT_H_
#define _POINT_H_

#include <iostream>

using namespace std;

namespace space {
      /**\brief This class represents a point
      */
      class Point {
            public:
                  int getX();
                  int getY();
                  void set(int x, int y);
                  void display();
            private:
                  int _x, _y;
      };
}

#endif
```

Within the class we have defined two private variables and four public methods. The first and second methods are used to obtain the values of the coordinates of the point, the third method is used for setting the values of the point, and the fourth method is used to display the values of the coordinates of the point. Finally, under the private declaration, two integers variables, representing the values of coordinates of the point, are declares. Private variables are declared starting with *underscore* "_" (**by convention**).

To hide the implementation details of a class and to allow changes to its implementation without changing the rest of the code that uses it, C++ introduces **access specifiers**. These allow you to define which members of a class are accessible from outside of it and which are not. There are three access specifiers: `public`, `private` and `protected`. `Public`, means that the member can be accessed from outside the class definition, while `private y protected` members can only be accessed from within the functions of the class or classes entitled to do so. By default, all members of a class are **private**.

If you try to access an attribute or private method, the compiler will output an error and fails to create the executable. The following code can be tested in `main.cpp`, to see how it fails to compile.

```
int main() {
      Point P;
      P._x = 10; // impossible! ERROR!
      P.display();
}
```

Another very important aspect of programming is documentation. All source files that you create should be documented. The comments should have the **Doxygen** format, which is as shown below:

```
/**
*/
```

There exist tools that can use documentation of this format for creating HTML documentation, which can be accessed and navigated using any browser.

Now let's create the `.cpp` file.

```
////////////////////////////////////////////////////
// point.cpp
////////////////////////////////////////////////////

#include "point.h"

namespace space {
      //////////////////////////////////////
      //
      //////////////////////////////////////
      int Point::getX() {
            return _x;
      }

      //////////////////////////////////////
      //
      //////////////////////////////////////
      int Point::getY() {
            return _y;
      }

      //////////////////////////////////////
      //
      //////////////////////////////////////
      void Point::set(int x, int y) {
            _x = x;
            _y = y;
      }

      //////////////////////////////////////
      //
      //////////////////////////////////////
      void Point::display() {
            cout << _x << ", " << _y << endl;
      }
}
```

Finally, compile the code using QtCreator and execute the program.

1.2. Constructors y destructor

1.2.1. Constructors

Probably as you may have noticed, when an instance of a class is created, class attributes initially contain "junk" value. However, it is possible to alter this behaviour by adding class constructors. A constructor is a function that is automatically invoked on creation of an object of the class, but this has to be defined.

Empty Constructor

First create an empty constructor. Add the following code to the class definition, right under the keyword "`public`".

```

/** Empty constructor
*/
Point();
```

This is a constructor, which is a member function that returns nothing and has the **same name as the class**. Note that, not only does the constructor return nothing, but it does not even say `void`. Although a member function can have parameters, in this case the parameter list is empty. Constructors are always the first class functions to be declared, and they should be **public**. Now see the implementation of it in the `.cpp` file.

```

////////////////////////////
//
//////////////////////////
Point::Point() {
      _x = 0;
      _y = 0;
}
```

The above code initializes the class variables to 0, for example, instead of containing "junk" value on creation of the object. Now, recompile the modified code and execute the program. Observe that the output would now be 0.

Overloading constructors
In C++ you can create multiple member functions with the same name but with different arguments list, this is called **overloading**. The compiler will take care of deciding which function to call depending on the arguments used.

Let us test overloading by adding two typical constructors, which are: **Parameterized constructor** and **Copy constructor**. The first will be used for assigning values to variables on creation of an instance of the class, while the second will create an object that is a copy of another. Add the following code to the `.h` file.

```
/** Parameterized constructor
*/
Point(int x, int y);

/** Copy constructor
*/
Point(const Point & P);
```

And then the following to the `.cpp` file.

```
///////////////////////////
//
/////////////////////////
Point::Point(int x, int y) {
	_x = x;
	_y = y;
}

///////////////////////////
//
/////////////////////////
Point::Point(const Point & P) {
	_x = P._x;
	_y = P._y;
}
```

Note that the first constructor accepts as an argument and assigns integer values to variables. The second constructor requires a little more explanation. Copy constructor, when called, makes the calling (newly created) object a copy of the object that is passed as its parameter. The (only) argument is a reference to an object of the same class.

```
const Point & P
```

The keyword `const` is used to indicate that the passed element must not be modified within the function. Notice the symbol &. In C there are two basic ways of passing arguments to a function: **by value** and **by reference**. In the first case, a copy of the variable is passed as an argument, where as in the second case, the memory address of the variable is passed as an argument. By passing the memory address (reference) of a variable, modification to it can be made directly from within the called function.

The same idea of pass by value or pass by reference applies when you want to pass a class object as a function parameter in C++. However, pass by value is **not** recommended in this case. This is because, classes may contain a lot of variables, and making a copy of all this data could become very costly and inefficient. It is therefore preferable to pass an object by its **reference**. To do this, the `&` operator is defined in C++. When you include `&`, it indicates that you can change the value of the variables from inside the called function. The concept of pass by references will be further explained in the forthcoming lab sessions. Now you can see a simple example.

```
main() {
      int v = 10;
      modify(v);
      cout << v << endl;
}

void modify(int & v) {
      v = 0;
}
```

In the case of copy constructor, the object is passed by reference (So that a duplicate copy of all its data is not created), although it is passed as a constant. This is done, so that it is not possible to modify its values from within the called function. Therefore, the following code would be **incorrect**:

```
///////////////////////////
//
/////////////////////////
Point::Point(const Point & P) {
      P._x = 100;
      P._y = 101;
}
```

Since it is trying to modify the variables of the passed object `P`, which is assumed to be a constant within the function.

1.2.2. Destructors

Destructor is a function that is called when the object has to be destroyed. That is, it is a function that is called when the object would no longer be used, if defined in the `.h` file.

```
/** Destructor
*/
~Point();
```

Now you have something new `~Point()`. This will be the destructor of the class. An object needs to free its memory, which is reserved when created, once it becomes no longer needed. The good thing about a destructor is that it is a function that is called automatically by the compiler when it detects that the object is not going to be used any more. There is no need to call the destructor explicitly; the compiler does it for the user. In the case of the example of the `Point` class, the destructor will do nothing for the moment but when you see dynamic memory allocation in the future lab session, this will be very useful.

The code that needs to be appended in the `.cpp` file would be:

```
///////////////////////////
//
/////////////////////////
Point::~Point() {
}
```

1.2.3. Operators

Another advantage of classes is that operators can be defined to be used over classes (=, +, -, *, /). Let us define the assignment operator. To do so, add the following definition under the public part of `Point` class:

```
/** Assign operator
*/
Point & operator=(const Point & P);
```

And the following in the `.cpp` file:

```
///////////////////////////
//
/////////////////////////
Point & Point::operator=(const Point & P) {
      _x = P._x;
      _y = P._y;

      return *this;
}
```

Operators are member functions that allow you to define common operations. All operators return a copy of the current object in an order that resembles a chain.

The return type is `Point &`, i.e. a reference to a class object. Looking at the implementation, once the copy is done, it returns `*this`. The keyword `this`, is pointer to the **current object**. The `this` pointer, used inside a function of class `Point`, is a pointer to the object, so in this case it is a pointer of type `Point *`. If `this` is a `Point *,` then `*this` is the object itself, i.e. `Point`. So what is being returned is the object itself. See an example:

```
int main(int argc, char **argv) {
      Point P(10, 10); // parametrized constructor
      Point P2;
      cout << P2.getX() << endl;
      cout << P2.getY() << endl;
      P2 = P;
      P2.display();
}
```

In this example, `P` is allocated to `P2`. Therefore, `P2` will be a copy of `P`. You can now see that:

```
int main(int argc, char **argv) {
      Point P(10);
      Point P2, P3;
      P2.display();
      P3 = P2 = P;
      P3.display();
}
```

In this example, `P` is assigned to `P2`, which is in turn assigned to `P3`. When written as `P3 = P2 = P`, the compiler will first perform `P2 = P`. The result of this operation is `P2`, which will then be assigned to `P3`. That is, the compiler decomposes the operation as:

```
P2 = P;
P3 = P2;
```

This is possible because the result of the assignment is a `Point &`.

Exercise

- Implement the classes needed for creating a database of people. To do this, define:
    - A `Person` class in the file `Person.h`, and implementation of its member functions in a file `Person.cpp`.
        - It must contain two private attributes, `_name,` of type `string`, and `_age` of type integer.
        - It should contain the following member functions:
            - Empty constructor, parameterized constructor and copy constructor. Each constructor would perform the appropriate initialization.
            - Destructor of the class.
            - Methods `getName` and `setName`: for getting and putting the name of the person.
            - Methods `getAge` and `setAge`: for getting and putting the age of the person.
            - Assignment operator.
- Create the `main.cpp` file in which the `main` function must contain the following:
    - Instantiate an object of type `Person` through its parameterized constructor and display its data on the screen.
    - Instantiate another object of type `Person` through its copy constructor, taking the previously created object as a reference, and the display the data of both the objects on the screen.
    - Instantiate the final object of type `Person` through its empty constructor and call its functions `setAge` and `setName`. Then display its data on the screen.
    - Instantiate two more objects of type `Person` and then assign values to its variables same as the ones assigned for the previous two objects, but by using the assignment operator.
    - Finally, display the names of all the people whose name contain the character 's'. Display the names of all the people whose age is more than 18 years.

Note

1. There should at least be some result.
2. Students are expected to complete the exercise on their own if they are unable to finish it during lab session.