Florida Institute of Technology
Department of Electrical and Computer Engineering

ECE 2551 – Software/Hardware Design

Tutorial: Introduction to C++ Classes and Objects

Reference: Chapter 3

Overview:

In this tutorial, you will learn how to create user-defined data types using C++ classes and how to create and use objects of those classes. You will define data members of a class to maintain data for each object of the class. You will also define member functions that operate on that data. You will learn how to call an object's member functions to request the services that the object provides and you will also learn to pass data to those member functions as arguments. The difference between a local variable of a member function and a data member of a class will be discussed. You will also be able to show how to use a constructor to specify initial values for an object's data members (i.e., object initialization).

Tutorial Objectives:

- *To define a class and use it to create one or more objects*
- *To specify a class's services as member functions*
- *To specify a class's attributes by defining data member variables*
- *To call a member function of an object to perform a task*
- *The differences between data members of a class and local variables of a function*
- *To use a constructor to initialize an object's data when the object is created*
- *To engineer a class to separate its interface from its implementation and encourage reuse*
- *To use objects of class string*

Summary of Tasks:

In this tutorial, you will create a class called Employee that includes three pieces of information as data members—a first name (type `string`), a last name (type `string`) and a monthly salary (type `int`). [*Note:* In subsequent chapters, we'll use numbers that contain decimal points (e.g 2.75)–called floating-point values—to represent dollar amounts.] Your class should have a constructor that initializes the three data members and it should provide a *set* and *get* function for each data member. If the monthly salary is not positive, set it to 0. Write a test program that demonstrate class Employee's capabilities. Create two `Employee` objects and display each object's yearly salary. Then give each Employee a 10 percent raise and display each Employee's yearly salary again.

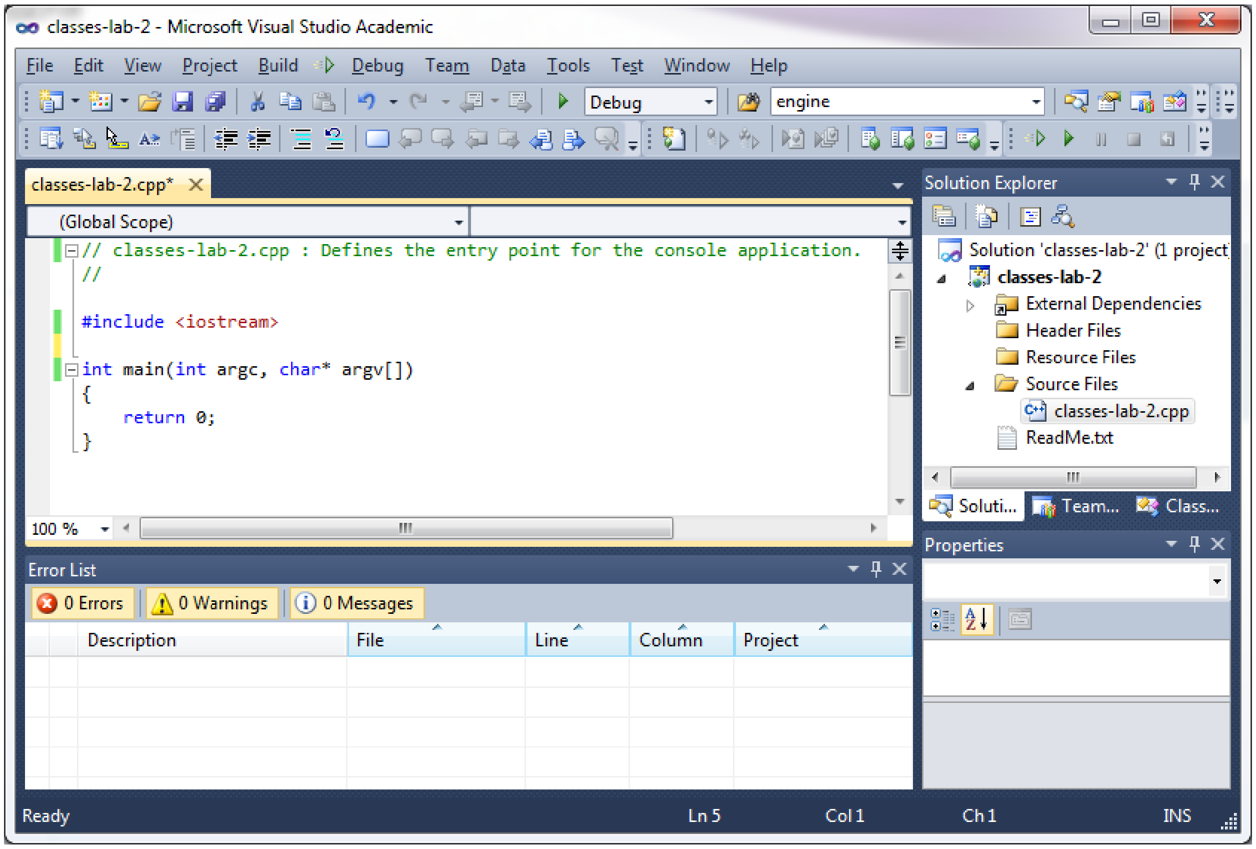
When you complete the program and the output is displayed to the screen, capture it using a "print-screen" and save it so that you can submit it with later on in your report.

Computer Software

This section describes the step-by-step instructions to create the `Employee` class and showcase its services using a test (driver) program. If you're brand-new to working with classes and objects, there will be some new things to learn, but we'll jump right in and explain things as we go. Also note that the examples here use a particular version of Microsoft Visual Studio; however, you can accomplish all of the steps used here in one way or another on different version of MS Visual Studio or other IDE's such as Eclipse, xCode, QT, etc.

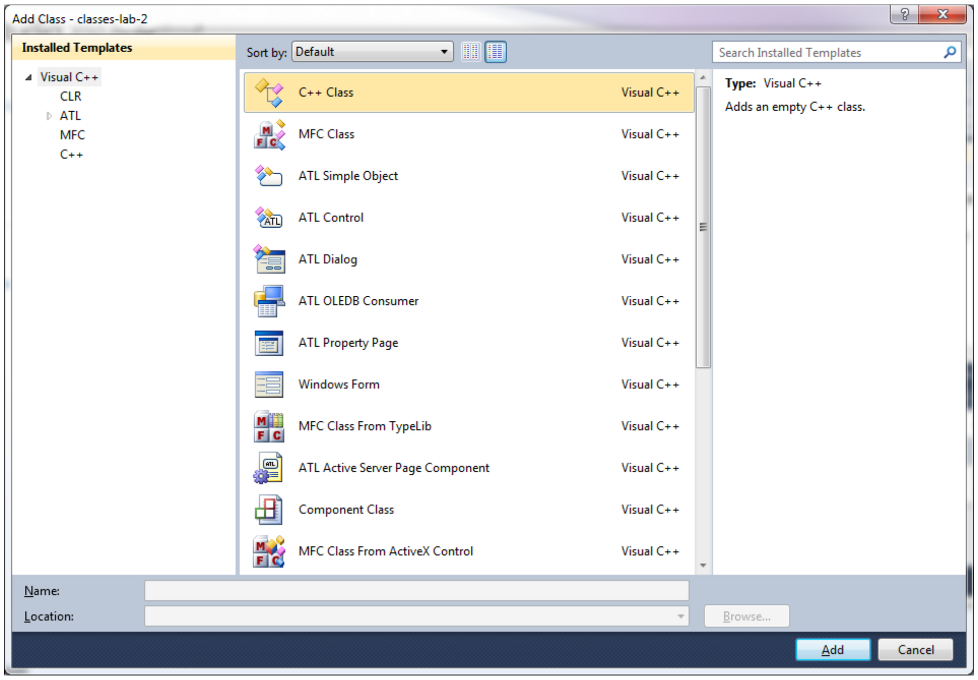
In C++, you have the ability to create your own data types—we call these user-defined types. (We use the term user-defined types to differentiate from native types, which are defined in the C++ standard library.) When working with user-defined types, there are two main "steps" that you need to follow. First, you need to create a "blueprint" for the data type. This is done using the `class` keyword and it is used to define the data and services provided by the user-defined type. Once a correct blueprint has been defined, the next step involves "building" one or more objects using the blueprint—this process is known as **instantiation**, which brings objects from a given class to life. Thus, one or more objects can be instantiated from a single class definition. Let's see how to define a class in C++.

Step 1 — *Starting the Microsoft Visual Studio Integrated Development Environment (IDE).* Open the MS VS Development Environment and create a console-based project (*we leave it to you to figure out how to create a console-based project, if you need to, feel free to ask the TA*). From this step on, it is assumed that a console-based project named **classes-lab-2** has been created.

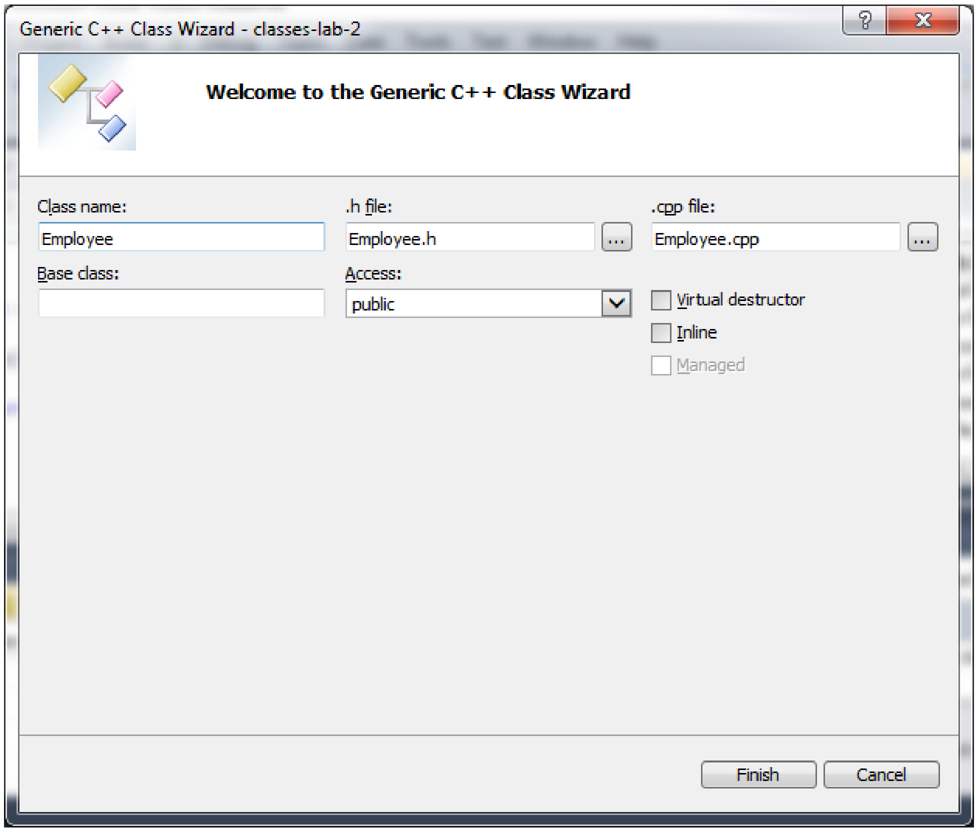

classes-lab-2 - Microsoft Visual Studio Academic
File Edit View Project Build Debug Team Data Tools Test Window Help
Debug
engine
classes-lab-2.cpp*
(Global Scope)
Solution Explorer
Solution 'classes-lab-2' (1 project
classes-lab-2
External Dependencies
Header Files
Resource Files
Source Files
classes-lab-2.cpp
ReadMe.txt
Soluti... Team... Class...
Properties
100 %
Error List
0 Errors
0 Warnings
0 Messages
Description File Line Column Project
Ready Ln 5 Col 1 Ch 1 INS

```
// classes-lab-2.cpp : Defines the entry point for the console application.
//

#include <iostream>

int main(int argc, char* argv[])
{
    return 0;
}
```

Step 2 *— Adding the Employee class.* In the solution explorer (upper-right window), right-click on the **classes-lab-2** project, and select the menu item Add->Class to open the Add Class window, as seen below.


Add Class - classes-lab-2
Installed Templates
Visual C++
CLR
ATL
MFC
C++
Sort by: Default
Search Installed Templates
C++ Class Visual C++
MFC Class Visual C++
ATL Simple Object Visual C++
ATL Control Visual C++
ATL Dialog Visual C++
ATL OLEDB Consumer Visual C++
ATL Property Page Visual C++
Windows Form Visual C++
MFC Class From TypeLib Visual C++
ATL Active Server Page Component Visual C++
Component Class Visual C++
MFC Class From ActiveX Control Visual C++
Type: Visual C++
Adds an empty C++ class.
Name:
Location:
Browse...
Add
Cancel

In the Add Class window, select the "C++ Class" option and click on the Add button to display the Generic C++ Class Wizard, as seen below.


Generic C++ Class Wizard - classes-lab-2

Welcome to the Generic C++ Class Wizard

Class name: Employee

.h file: Employee.h

.cpp file: Employee.cpp

Base class:

Access: public

Virtual destructor [ ]

Inline [ ]

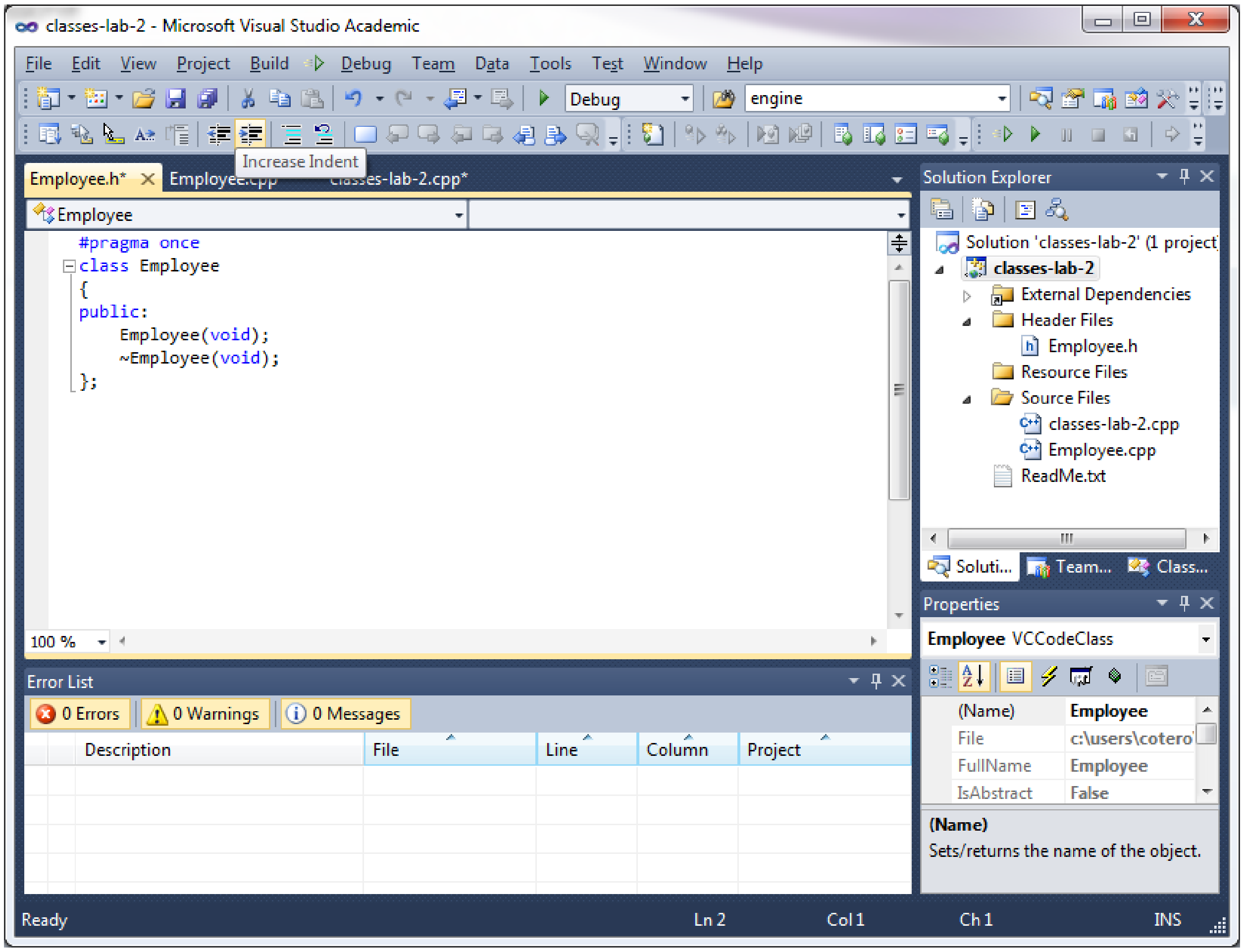
Managed [ ]

In the "Generic C++ Class Wizard" window, enter `Employee` in the Class Name textbox—this should fill out all other relevant textboxes—and click on the Finish button. Once you have done this, your Solution Explorer window will be updated with two files, Employee.h under the Header Files folder and Employee.cpp under the Source Files folder, as seen below. From now on, every class that we create will have both a *.h* and *.cpp* files. The *.h* file will be used to define the class, declare the class' functions, and define its data members. The .cpp file will be used to define the class' functions (i.e., write the code that goes inside the class' functions). We'll see examples of this shortly.

Let's take a closer look at the Employee.h file, where the class definition is almost empty, except for two special functions created for you by the Microsoft Visual Studio IDE, as seen below:

```
class Employee
{
public:
    Employee();
    ~Employee();
};
```

The first thing to notice about both functions is that they do not include a function body; these are function declarations, thus they do not specify what the functions will actually do. However, one thing that looks interesting is that these functions' names are both the same as the class name (i.e., `Employee`). As it turns out, both of these functions are special functions of the `Employee` class. Let's start with the Employee() function, which specifies no return value. Any class function that does not specify a return value and is named after the class name is called a constructor function, or simply **constructor**. Constructors are essential in C++ to define the code that executes every time an object of a given class is *instantiated*. This means that once a variable of the class Employee is created, this constructor function will be called “behind-the-scenes” automatically.



```
#pragma once
class Employee
{
public:
    Employee(void);
    ~Employee(void);
};
```

classes-lab-2 - Microsoft Visual Studio Academic
File Edit View Project Build Debug Team Data Tools Test Window Help
Debug
engine
Increase Indent
Employee.h* Employee.cpp classes-lab-2.cpp*
Employee
100 %
Solution Explorer
Solution 'classes-lab-2' (1 project
classes-lab-2
External Dependencies
Header Files
Employee.h
Resource Files
Source Files
classes-lab-2.cpp
Employee.cpp
ReadMe.txt
Soluti... Team... Class...
Properties
Employee VCCodeClass
(Name) Employee
File c:\users\cotero
FullName Employee
IsAbstract False
(Name)
Sets/returns the name of the object.
Error List
0 Errors 0 Warnings 0 Messages
Description File Line Column Project
Ready Ln 2 Col 1 Ch 1 INS

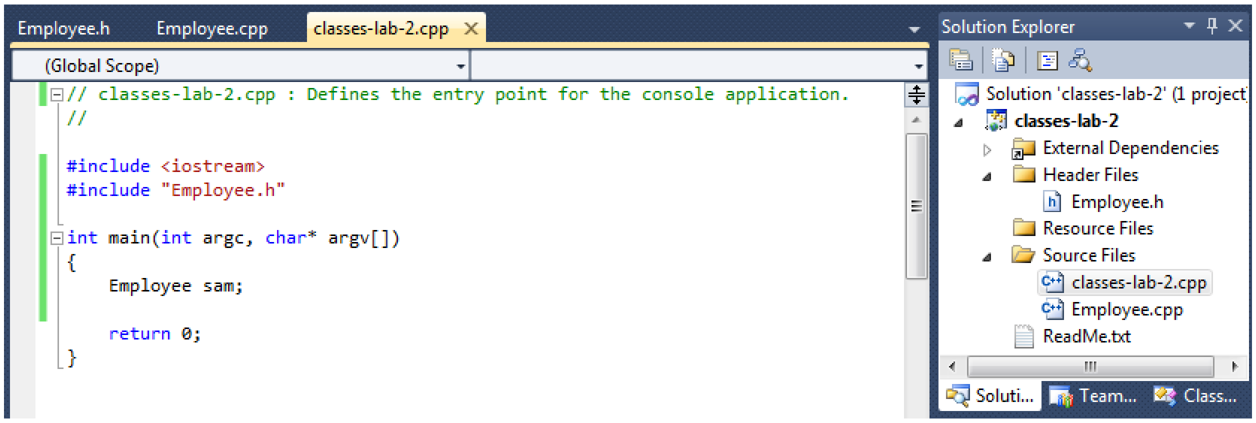
Let’s use the constructor to add some initialization code for the `Employee` class. To define the constructor, open the Employee.cpp file by double-clicking the file in the solution explorer window. You should see code similar to the one below.

```
#include "Employee.h"

Employee::Employee(void)
{
}

Employee::~Employee(void)
{
}
```

Modify the constructor function to display the following message to the console “*Employee object created!*” To do this, you will need to include the C++ standard library’s `cout` object, which is defined in the `iostream` library. (Notice that the standard c++ library's `cout` object belongs to the namespace `std`, so you will need to specify this as well, as seen below.)

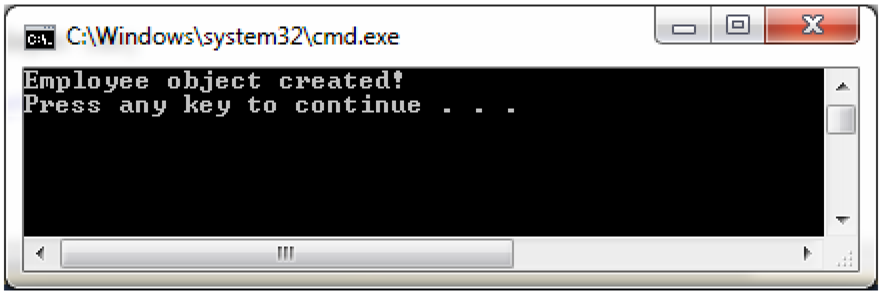
```
#include <iostream>
#include "Employee.h"

using namespace std;

Employee::Employee(void)
{
  cout<<"Employee object created!\n";
}

Employee::~Employee(void)
{
}
```

To test drive the newly created Employee class, (in the solution explorer) double-click on the file that contains your main function and instantiate an object of the `Employee` class—name this object sam, as seen below. Remember, for the compiler to understand how to work with objects of type `Employee`, you need to include the Employee's class definition. We do this by using the `#include "Employee.h"` line of code as seen below.



```
// classes-lab-2.cpp : Defines the entry point for the console application.
//

#include <iostream>
#include "Employee.h"

int main(int argc, char* argv[])
{
    Employee sam;

    return 0;
}
```

Once you compile and execute the code—you can click on Debug->Start Without Debugging to execute the program—you should see the console executing with similar output as below.

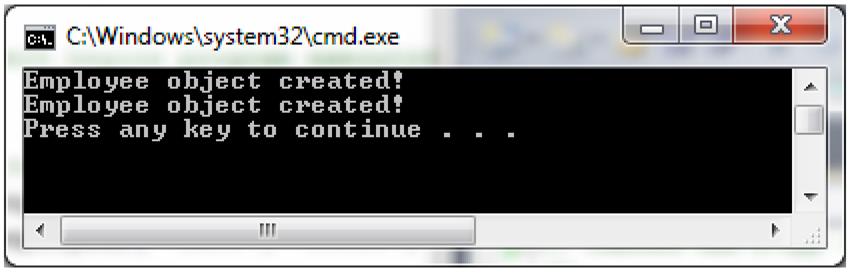


```
Employee object created!
Press any key to continue . . .
```

This program does very little but showcases a very important concept from C++ classes, which is their initialization. That is, *objects from a given class are initialized via constructors*. In this example, we only displayed a message to the console, but if the class definition had included data members, we could've used the constructor to initialize their values, as we will see later on. Let's instantiate another object of the class `Employee` to see how the output of the program changes. Modify the main function to add the following line; then compile and execute the program.

```
// Instantiate a second object of the class Employee
Employee alex;
```

The output of the program should now look like the one below.



```
Employee object created!
Employee object created!
Press any key to continue . . .
```

Since two objects are now being instantiated, two constructor calls are being made, one for each object coming to life. Before moving on, let's mention briefly the `~Employee()`, which is called a *destructor*. Destructors are opposite to constructors, that is, instead of being called upon object creation, they are called automatically "behind-the-scenes" once an object of a given class is destroyed. We'll talk more about destructors later on in the course.

Step 3 — *Specify the data members used by the class.* Data member definition occurs in the .h file, so open the Employee.h file by double-clicking on it in the solution explorer window.
In the class definition, edit the class to include the following three private data members, as seen below.

```
// Employee class definition.
#include <string> // program uses C++ standard string class

// Employee class definition
```

```
class Employee
{
private:
   std::string firstName; // Employee's first name
   std::string lastName; // Employee's last name
   int monthlySalary; // Employee's salary per month
}; // end class Employee
```

It is a good programming practice to define data member variables under the private section of the class. Anything defined under the private section of the class is **only** accessible within the class. In this example, this means that the data members' `firstName`, `lastName`, and `monthlySalary` can only be directly accessed by member functions of the class Employee (e.g., constructors or other user-defined member function). By preventing clients of your class from accessing directly the data members, you can control how data are stored in your objects. This principle is referred to as information hiding or **encapsulation**. Private data members allow class designers to hide implementation details to the user of a class.

Step 4 *— Specify the services provided by class via function declaration.*
Member function definition occurs in both the .h and .cpp files (*step 5*), so let's start by editing the Employee.h file by double-clicking on it in the solution explorer window. Once open, add a public section of code and add one parameterized constructor, and several get and set methods, as seen below.

```
// Employee class definition.
#include <string> // program uses C++ standard string class

// Employee class definition
class Employee
{
public:
   // constructor sets data members.
   Employee( std::string, std::string, int );
   void setFirstName( std::string ); // set first name
   std::string getFirstName(); // return first name
   void setLastName( std::string ); // set last name
   std::string getLastName(); // return last name
   void setMonthlySalary( int ); // set monthly salary
   int getMonthlySalary(); // return monthly salary
private:
   std::string firstName; // Employee's first name
   std::string lastName; // Employee's last name
   int monthlySalary; // Employee's salary per month
}; // end class Employee
```

Any code (i.e., data members or member functions) declared under the public section of the class is accessible by other member functions (within the class code) **and** by clients of objects instantiated from the class. For example, in this case, the function setFirstName(…) is accessible by code inside the main function (i.e., the client), as seen below.

```
void main() {
  // Instantiate the object sam using the parameterized constructor
  // of the Employee class.
  Employee sam("sem", "Malone", 60,000);
  sam.setFirstName("sam");
}
```

In this example, the function `main` is the client of the object `sam` and it is allowed to call on any of the object's public functions. Once the object is instantiated, the function setFirstName is used to set the value of the privately-defined `firstName` data member variable through the publicly-defined function. It is good programming practice to include public member functions to control access (get and set) the private data member.

***Step 5** — Define the functions.*
Once the .h file is edited to include the member function prototypes, we can now define the functions in the .cpp file, so open the Employee.cpp file by double-clicking on it in the solution explorer window. Each function definition in the .cpp file will have the following format:

```
<<return type>> <<class name>>::<<function name>>(<<optional parameter list>>)
{
  <<function body>>
}
```

As example, consider the following function definition, which specifies the function named setFirstName, which takes one parameter of type string, and belongs to the class Employee.

```
void Employee::setFirstName(string name) {
   // code here.
}
```

Note that the symbol `::` is known as the *scope operator*, hence the code `Employee::setFirstName` can be interpreted as the function `setFirstName` "scoped" by the `Employee` class, or in other words, the function `setFirstName` that belongs to the class `Employee`. There could be other functions in our code named `setFirstName`, but there can only be one with that specific signature that belongs to the `Employee` class. This is how the compiler knows how to use the correct function in our code.

Since constructors do not have a return value, constructors will have the following format:

```
<<class name>>::<<function name>>(<<optional parameter list>>)
{
  <<function body>>
}
```

Let's continue defining our class by including the code for each function identified in step 4, as seen below.

```
// Employee class member-function definitions.
#include <iostream>
#include "Employee.h" // Employee class definition
```

```
using namespace std;

// Employee constructor initializes the three data members
Employee::Employee( string first, string last, int salary )
{
   setFirstName( first ); // store first name
   setLastName( last ); // store last name
   setMonthlySalary( salary ); // validate and store monthly salary
} // end Employee constructor

// set first name
void Employee::setFirstName( string name )
{
   firstName = name; // no validation needed
} // end function setFirstName

// return first name
string Employee::getFirstName()
{
   return firstName;
} // end function getFirstName

// set last name
void Employee::setLastName( string name )
{
   lastName = name; // no validation needed
} // end function setLastName

// return last name
string Employee::getLastName()
{
   return lastName;
} // end function getLastName

// set monthly salary; if not positive, set to 0
void Employee::setMonthlySalary( int salary )
{
   if ( salary > 0 ) // if salary is positive
      monthlySalary = salary; // set monthlySalary to salary

   if ( salary <= 0 ) // if salary is not positive
      monthlySalary = 0; // set monthlySalary to 0
} // end function setMonthlySalary

// return monthly salary
int Employee::getMonthlySalary()
{
   return monthlySalary;
} // end function getMonthlySalary
```

***Step 6** — Create the driver (main) program.*
At this point, the class Employee has been defined and it is ready to use. In this step, we will create the “client” code to showcase the usage of Employee objects. Open the file that contains

the main function and create two instances of the Employee class, i.e., employee1 and employee2, as seen below. Since we have defined a parameterized constructor for the Employee class, we can use it to initialize the objects upon creation. Once the objects have been instantiated, we can use the dot operator (.) to access the public services (i.e., functions) offered by the class, which are the ones where the code resides in the .cpp file. Enter the following code in the main function.

```
// Create and manipulate two Employee objects.
#include <iostream>
#include "Employee.h" // include definition of class Employee
using namespace std;

// function main begins program execution
int main()
{
   // create two Employee objects
   Employee employee1( "Lisa", "Roberts", 4500 );
   Employee employee2( "Mark", "Stein", 4000 );

   // display each Employee's yearly salary
   cout << "Employees' yearly salaries: " << endl;

   // retrieve and display employee1's monthly salary multiplied by 12
   int monthlySalary1 = employee1.getMonthlySalary();
   cout << employee1.getFirstName() << " " << employee1.getLastName()
      << ": $" << monthlySalary1 * 12 << endl;

   // retrieve and display employee2's monthly salary multiplied by 12
   int monthlySalary2 = employee2.getMonthlySalary();
   cout << employee2.getFirstName() << " " << employee2.getLastName()
      << ": $" << monthlySalary2 * 12 << endl;

   // give each Employee a 10% raise
   employee1.setMonthlySalary( monthlySalary1 + monthlySalary1 / 10 );
   employee2.setMonthlySalary( monthlySalary2 + monthlySalary2 / 10 );

   // display each Employee's yearly salary again
   cout << "\nEmployees' yearly salaries after 10% raise: " << endl;

   // retrieve and display employee1's monthly salary multiplied by 12
   monthlySalary1 = employee1.getMonthlySalary();
   cout << employee1.getFirstName() << " " << employee1.getLastName()
      << ": $" << monthlySalary1 * 12 << endl;

   monthlySalary2 = employee2.getMonthlySalary();
   cout << employee2.getFirstName() << " " << employee2.getLastName()
      << ": $" << monthlySalary2 * 12 << endl;
} // end main
```

***Step 7** — Compile, build, and execute the program.*

Self-Assessment

The objective of this section is to provide additional exercises that would help improve students' understanding of this tutorial. Although students are not required to submit answers to these questions, they are required to understand the material covered in this tutorial. Future assessment may be based on this material.

1. Explain the use of the `::` operator?

2. Modify the `Employee` class attribute section to include the `age` data member of type `int`. Once this is done, edit your main driver program of step 6 to add the following capabilities:
   a. Initialize the object's age during instantiation (in addition to all other existing data).
   b. Change the object's age during your program execution, for example, by prompting the user to change the age.
   c. Display the object's age before terminating the program.

3. If the pre-processing directive `#include "Employee.h"` is not included in the main .cpp file, will the program compile and execute? Why or why not? What does `#include` does?

4. Edit the main driver program from step 6 to include the following:

   ```
   Employee employee3;
   ```

   Will the program compile and execute? Why or why not? If it doesn't compile, provide suggestions to fix the code.

5. Modify the constructor of the Employee class as seen below.

   ```
   // Employee constructor.
   Employee::Employee( string first, string last, int salary ) :
   firstName(first), lastName(last), monthlySalary(salary)

   {
     // Intentionally left blank.

   } // end Employee constructor
   ```

   Will the program compile and execute? Why or why not? Explain what you think is happening here.

6. If you modify the Employee class so that the .h, .cpp, and driver code looked like the one below, will the program compile and execute? What will be the output of this program? Explain and if necessary, show the code that will enable the program to compile and execute correctly.

.h File

```
 // the class
class Employee
```

```
{
private:
   std::string firstName; // Employee's first name

}; // end class Employee
```

.cpp File

```
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

// Employee constructor initializes the three data members
Employee::Employee( string first)
{
   setFirstName( first ); // store first name
} // end Employee constructor
```

Driver code

```
// the main function
#include <iostream>
#include "Employee.h"
using namespace std;

int main()
{
   Employee employee1("Roberts");
}
```

7. Explain in your own words the concept of classes. What are they? What are they used for? Are they really needed in C++? How useful do you think they are? Is there a difference between classes and objects? If so, what is the relationship between classes and objects? In your write-up, include your own thoughts on what you have learned so far on the topic of classes.