C++ Chapter 7: OOP

Contents

How Does Typing in a Code Tutorial Help with Learning?

Typing in a code tutorial can significantly enhance learning in several ways:

- **Active Engagement**: Typing the code yourself forces you to actively engage with the material, rather than passively reading or watching. This active participation helps reinforce the concepts being taught.
- **Muscle Memory**: Repeatedly typing code helps build muscle memory, making it easier to recall syntax and structure when you write code independently.
- **Error Handling**: When you type code, you're likely to make mistakes. Debugging these errors helps you understand common pitfalls and how to resolve them, which is a crucial skill for any programmer.
- **Understanding**: Typing out code allows you to see how different parts of the code interact. This deeper understanding can help you apply similar concepts to different problems.
- **Retention**: Studies have shown that actively doing something helps with retention. By typing out the code, you're more likely to remember the concepts and techniques.

Time required: 120 minutes

Online Tutorial

Go through the following tutorials before going through the tutorial assignments.

- C++ What is OOP?
- C++ Classes and Objects
- C++ Class Methods
- C++ Constructors
- C++ Access Specifiers
- C++ Encapsulation

Why OOP?

A car is assembled from parts and components, such as chassis, doors, engine, wheels, brake, and transmission. The components are reusable, e.g., a wheel can be used in many cars (of the same specifications).

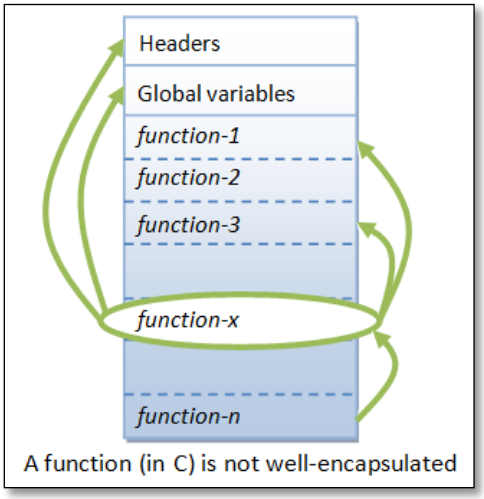
Hardware, such as computers and cars, are assembled from parts, which are reusable components.

How about software? Can you "assemble" a software application by picking a routine here, a routine there, and expect the program to run? The answer is obviously no! Unlike hardware, it is very difficult to "assemble" an application from software components. Since the advent of computer 60 years ago, we have written tons and tons of programs. However, for each new application, we re-invent the wheels and write the program from scratch.

Why re-invent the wheel?

Traditional Procedural-Oriented Languages

Can we do this in traditional procedural-oriented programming language such as C, Fortran, Cobol, or Pascal?



A function (in C) is not well-encapsulated

Traditional procedural-oriented languages (such as C and Pascal) suffer some notable drawbacks in creating reusable software components:

The programs are made up of functions. Functions are often not reusable. It is very difficult to copy a function from one program and reuse in another program because the function is likely to reference the headers, global variables and other functions. In other words, functions are not well-encapsulated as a self-contained reusable unit.

The procedural languages are not suitable of high-level abstraction for solving real life problems. For example, C programs uses constructs such as if-else, for-loop, array, function, pointer, which are low-level and hard to abstract real problems such as a Customer Relationship Management (CRM) system or a computer soccer game. (Imagine using assembly codes, which is a very low-level code, to write a computer soccer game. C is better but not much better.)

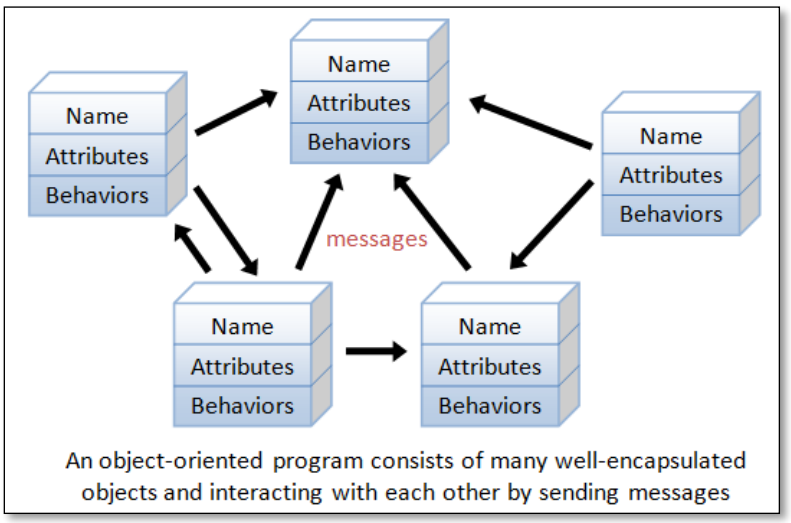
In brief, the traditional procedural-languages separate the data structures and algorithms of the software entities.

Object-Oriented Programming

Object-oriented programming (OOP) languages are designed to overcome these problems.

The basic unit of OOP is a class, which encapsulates both the static attributes and dynamic behaviors within a "box" and specifies the public interface for using these boxes. Since the class is well-encapsulated (compared with the function), it is easier to reuse these classes. In other words, OOP combines the data structures and algorithms of a software entity inside the same box.

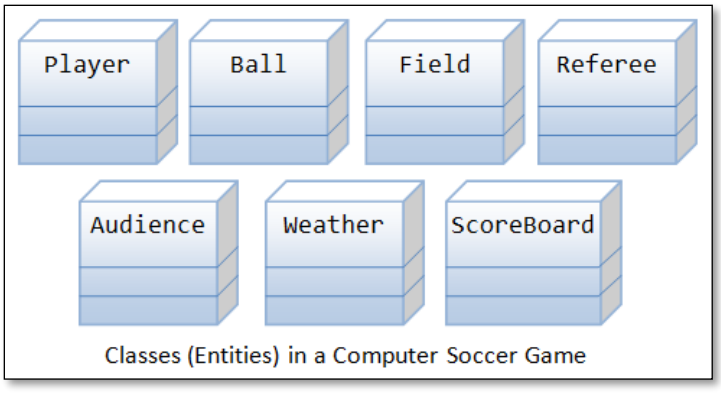
OOP languages permit higher level of abstraction for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++, C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.



As an example, suppose you wish to write a computer soccer game (which I consider as a complex application). It is quite difficult to model the game in procedural-oriented languages. But using OOP languages, you can easily model the program accordingly to the "real things" appear in the soccer games.

Player: attributes include name, number, location in the field, and etc; operations include run, jump, kick-the-ball, and etc.

- Ball:
- Referee:
- Field:
- Audience:
- Weather:



Classes (Entities) in a Computer Soccer Game

Some of these classes (such as Ball and Audience) can be reused in another application, e.g., computer basketball game, with little or no modification.

Benefits of OOP

The procedural-oriented languages focus on procedures, with function as the basic unit. You need to first figure out all the functions and then think about how to represent data.

The object-oriented languages focus on components that the user perceives, with objects as the basic unit. You figure out all the objects by putting all the data and operations that describe the user's interaction with the data.

Object-Oriented technology has many benefits:

- Ease in software design as you could think in the problem space rather than the machine's bits and bytes. You are dealing with high-level concepts and abstractions. Ease in design leads to more productive software development.
- Ease in software maintenance: object-oriented software is easier to understand, therefore easier to test, debug, and maintain.
- Reusable software: you don't need to keep re-inventing the wheels and re-write the same functions for different situations. The fastest and safest way of developing a new application is to reuse existing code - fully tested and proven code.

OOP Basics

Classes and Instances

Class: A class is a definition of an object of the same kind. A class is a blueprint, template, or prototype that defines and describes the static attributes and dynamic behaviors common to all objects of the same kind.

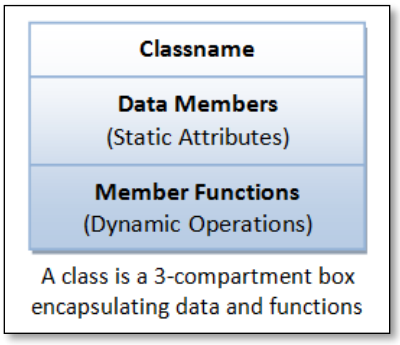
Instance: An instance is a realization of a particular item of a class. An instance is an instantiation of a class. All the instances of a class have similar properties, as described in the class definition. For example, you can define a class called "Student" and create three instances of the class "Student" for "Peter", "Paul" and "Pauline".

The term "object" refers to instance.

What is a Class?

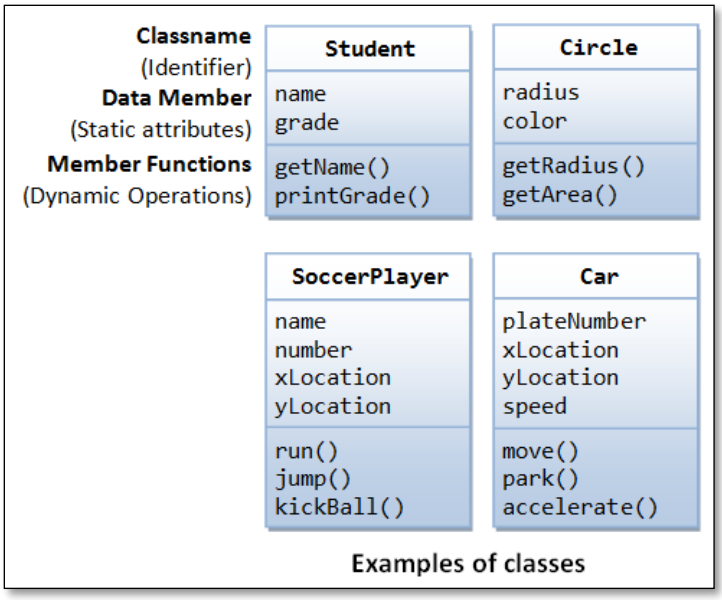
A class can be visualized as a three-compartment box encapsulating Data and Methods.

1. **Classname** (or identifier): identifies the class.
2. **Data Members or Variables** (or attributes, states, fields): contains the static attributes of the class.
3. **Member Functions** (or methods, behaviors, operations): contains the dynamic operations of the class.

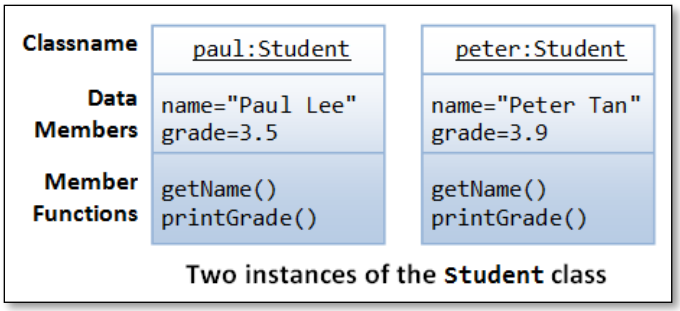


A class is a 3-compartment box encapsulating data and functions

A class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.



An example of two instances of the class student.



Unified Modeling Language (UML) Class and Instance Diagrams: The above class diagrams are drawn according to the UML notations. A class is represented as a 3-compartment box, containing name, data members (variables), and member functions, respectively. classname is shown in bold and centralized. An instance (object) is also represented as a 3-compartment box, with instance name shown as `instanceName:Classname` and underlined.

Summary of OOP

- A class is a programmer-defined, abstract, self-contained, reusable software entity that mimics a real-world thing.
- A class is a 3-compartment box containing the name, data members (variables) and the member functions.

- A class encapsulates the data structures (in data members) and algorithms (member functions). The values of the data members constitute its state. The member functions constitute its behaviors.
- An instance is an instantiation (or realization) of a particular item of a class.

C++ Class

A class is a blueprint for an object.

We can think of a class as a sketch (prototype) or blueprint of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Create a Class

A class is defined in C++ using keyword **class** followed by the name of the class.

The body of the class is defined inside the curly brackets and terminated by a semicolon at the end.

```
class ClassName {
   // some data
   // some functions
};
```

Here is an example of a Room class.

```
class Room {
    private:
        double length;
        double width;
        double height;
    public:
        double calculateArea(){
            return length * width;
        }
        double calculateVolume(){
            return length * width * height;
        }
};
```

Another example of a SoccerPlayer.

```
class SoccerPlayer {   // classname
private:
   int number;         // Private data members (fields, variables)
   string name;
   int x, y;
public:
   void run();         // Public member functions (methods)
   void kickBall();
};
```

A Circle class.

```
class Circle {         // classname
private:
   double radius;      // Data members (variables)
   string color;
public:
   double getRadius(); // Member functions
   double getArea();
};
```

Naming Conventions

A **data member** (variable) has a name (or identifier) and a data type. It holds a value of that data type.

Data Member Naming Convention: A data member name is a noun or a noun phrase made up of several words. The first word is in lowercase and the rest of the words are initial-capitalized (camel-case), e.g., fontSize, roomNumber, xMax, yMin and xTopLeft. Take note that variable name begins with a lowercase, while classname begins with an uppercase.

A **member function** (method):

1. receives parameters from the caller
2. performs the operations defined in the function body
3. returns a piece of result (or void) to the caller

Member Function Naming Convention: A function name is a verb, or a verb phrase made up of several words. The first word is in lowercase and the rest of the words are initial-capitalized (camel-case). For example, getRadius(), getParameterValues().

A data member name is a noun (denoting a static attribute), while a function name is a verb (denoting an action). They have the same naming convention. You can easily distinguish them from the context. Functions take arguments in parentheses (possibly zero argument with empty parentheses), but variables do not.

"public" vs "private" Access Control

An access control modifier can be used to control the visibility of a data member or a member function within a class. We begin with the following two access control modifiers:

- **public:** The member (data or function) is accessible and available to all parts of the program.
- **private:** The member (data or function) is accessible and available within this class only.

Keyword "this"

You can use keyword "this" to refer to this instance inside a class definition.

One of the main usages of keyword this is to resolve ambiguity between the names of data member and function parameter. For example,

```
class Circle {
private:
   double radius;                 // Member variable called "radius"
   ......
public:
   void setRadius(double radius) { // the argument also called "radius"
      this->radius = radius;
         // "this->radius" refers to this instance's member variable
         // "radius" resolved to the function's argument
   }
   ......
}
```

In the above code, there are two identifiers called radius - a data member and the function parameter. This causes a naming conflict. To resolve the naming conflict, you could name the function parameter rds instead of radius. However, radius is more approximate and meaningful in this context. You can use keyword this to resolve this naming conflict. "this->radius" refers to the data member; while "radius" resolves to the function parameter.

"this" is a pointer to this object. A pointer is an address that points to another address. It is somewhat like a reference.

You could use a prefix (such as m_) or suffix (such as _) to name the data members to avoid name crashes. For example,

```
class Circle {
private:
   double m_radius;  // or radius_
   ......
public:
   void setRadius(double radius) {
      m_radius = radius;  // or radius_ = radius
   }
   ......
}
```

The C++ Compiler internally names the data members beginning with a leading underscore (e.g., _xxx) and local variables with 2 leading underscores (e.g., __xxx). Avoid names beginning with underscore in your program.

Tutorial 1: Spaceship

The following tutorial program uses a Spaceship class to calculate the area and volume of a square spaceship. Create a C++ program named **spaceship.cpp** Add the following code.

```cpp
/**
 * Filename: spaceship.cpp
 * Written by:
 * Written on:
 * Create a spaceship object, calculate the area and volume
 * This is a box shaped spaceship designed for interplanetary travel
 * This is the initial program for building a Borg Cube
 */

#include <iostream>

/*****************************************************************
 * SPACESHIP CLASS
 *****************************************************************/
class Spaceship
{
private:
    /************** PRIVATE DATA MEMBERS ************************/
    double m_length;
    double m_width;
    double m_height;

public:
    /************** PUBLIC MEMBER FUNCTIONS *********************/
    double calculateArea()
    {
        return m_length * m_width;
    }

    double calculateVolume()
    {
        return m_length * m_width * m_height;
    }

    void insertData(double length, double width, double height)
    {
        m_length = length;
        m_width = width;
        m_height = height;
    }
};
```

```
43  int main()
44  {
45      // Create object of Spaceship class
46      Spaceship spaceship_1;
47
48      // Assign values to data members
49      spaceship_1.insertData(42.5, 30.8, 19.2);
50
51      // Calculate and display the area and volume of the room
52      std::cout << "  Area of Spaceship: "
53                << spaceship_1.calculateArea() << std::endl;
54      std::cout << "Volume of Spaceship: "
55                << spaceship_1.calculateVolume() << std::endl;
56
57      return 0;
58  }
```

Example run:

```
Area of Spaceship:   1309
Volume of Spaceship: 25132.8
```

Constructors

A constructor is a special member function that has the function name the same as the classname. A constructor is called automatically wen an object is created.

In the Circle class, we define a constructor as follows:

```
// A constructor has the same name as the class
public:
    // Default constructor
    Circle(){};
    // Parameterized constructor
    Circle(double r, string c) {
        radius = r;
        color = c;
};
```

A constructor with no parameters is known as a **default constructor**. In the example above, Circle() is a default constructor. A default constructor can be empty or have code.

In C++, a constructor with parameters is known as a **parameterized constructor**. This is the preferred method to initialize member data. A parameterized constructor is used to construct and initialize all the data members.

If you want to create an object without any parameters, an empty default constructor is also needed.

If we have not defined a constructor in our class, then the C++ compiler will automatically create a default constructor with an empty code and no parameters.

To create a new instance of a class, declare the name of the instance and invoke the constructor. For example,

```
// Parameterized constructor creates an object
// that passes parameters to the data members
Circle c1(1.2, "blue");
Circle c2(3.5, "red");
// Uses the default constructor to create an object
Circle c3;
```

A constructor function is different from an ordinary function in the following aspects:

- The name of the constructor is the same as the classname.
- Constructor has no return type (or implicitly returns void). Hence, no return statement is allowed inside the constructor's body.
- Constructor can only be invoked once to initialize the instance constructed. You cannot call the constructor afterwards in your program.

Tutorial 2: Spaceship with Constructors and Overloaded Methods

Open **Spaceship.cpp** Save the file as **SpaceshipConstructor.cpp** Add the following changed code.

```cpp
/**
 * Filename: spaceship_constructor.cpp
 * Written by:
 * Written on:
 * Create a Spaceship object with parameterized constructor
 */

#include <iostream>

/*****************************************************
 * SPACESHIP CLASS
 *****************************************************/
class Spaceship
{
private:
    /************** PRIVATE DATA MEMBERS **************/
    double m_length;
    double m_width;
    double m_height;

public:
    /************** PUBLIC MEMBER FUNCTIONS ************/
    // Default constructor
    Spaceship(){};

    // Parameterized constructor
    Spaceship(double length, double width, double height)
    {
        m_length = length;
        m_width = width;
        m_height = height;
    }

    // Method without parameters
    double calculateArea()
    {
        return m_length * m_width;
    }

    // Method overloading with parameters
    // C++ looks at the method signature to determine which
    // method to run
    double calculateArea(double length, double width)
    {
        return length * width;
    }
```

```
48      double calculateVolume()
49      {
50          return m_length * m_width * m_height;
51      }
52
53      void insertData(double length, double width, double height)
54      {
55          m_height = height;
56          m_width = width;
57          m_length = length;
58      }
59  };
60
61  int main()
62  {
63      // Create object of Room class
64      Spaceship spaceship_1;
65
66      // Assign values to data members
67      spaceship_1.insertData(42.5, 30.8, 19.2);
68
69      // Calculate and display the area and volume of the room
70      std::cout << "  Area of Spaceship 1: "
71                << spaceship_1.calculateArea() << std::endl;
72      std::cout << "  Area of Spaceship 1: "
73                << spaceship_1.calculateArea(12, 12) << std::endl;
74      std::cout << "Volume of Spaceship 2: "
75                << spaceship_1.calculateVolume() << std::endl;
76
77      // Create object of Room class with parameterized constructor
78      Spaceship spaceship_2(98.3, 20.1, 15.7);
79
80      // Calculate and display the area and volume of the room
81      std::cout << "\n  Area of Spaceship 2: "
82                << spaceship_2.calculateArea() << std::endl;
83      std::cout << "Volume of Spaceship 2: "
84                << spaceship_2.calculateVolume() << std::endl;
85
86      return 0;
87  }
```

Example run:

```
  Area of Spaceship 1: 1309
  Area of Spaceship 1: 144
Volume of Spaceship 2: 25132.8

  Area of Spaceship 2: 1975.83
Volume of Spaceship 2: 31020.5
```

Information Hiding and Encapsulation

A class encapsulates the static attributes and the dynamic behaviors into a "3-compartment box". Once a class is defined, you can seal up the "box" and put the "box" on the shelf for others to use and reuse. Anyone can pick up the "box" and use it in their application. This cannot be done in the traditional procedural-oriented language like C, as the static attributes (or variables) are scattered over the entire program and header files. You cannot "cut" out a portion of C program, plug into another program and expect the program to run without extensive changes.

Data member of a class are typically hidden from the outside world, with private access control modifier. Access to the private data members are provided via public assessor functions, e.g., getRadius() and getColor().

This follows the principle of information hiding. That is, objects communicate with each other using well-defined interfaces (public functions). Objects are not allowed to know the implementation details of others. The implementation details are hidden or encapsulated within the class. Information hiding facilitates reuse of the class.

Rule of Thumb: Do not make any data member public, unless you have a good reason.

Getters and Setters

To allow other part of the program to read the value of a private data member like xxx, you shall provide a get function (or getter or accessor function) called getXxx(). A getter need not expose the data in raw format. It can process the data and limit the view of the data others will see. Getters shall not modify the data member.

To allow other classes to modify the value of a private data member says xxx, you shall provide a set function (or setter or mutator function) called setXxx(). A setter could provide data validation (such as range checking) and transform the raw data into the internal representation.

In our Circle class, the data members radius and color are declared private. They are only available within the Circle class and not visible outside the Circle class - including main().

You cannot access the private data members radius and color from the main() directly - via say c1.radius or c1.color. The Circle class provides two public accessor functions, namely, getRadius() and getColor(). These functions are declared public. The main() can invoke these public accessor functions to retrieve the radius and color of a Circle object, via c1.getRadius() and c1.getColor().

There is no way you can change the radius or color of a Circle object, after it is constructed in main(). You cannot issue statements such as c1.radius = 5.0 to change the radius of instance c1, as radius is declared as private in the Circle class and is not visible to other including main().

If the designer of the Circle class permits the change the radius and color after a Circle object is constructed, he must provide the appropriate setter, e.g.,

```
// Setter for color
void setColor(string c) {
   color = c;
}

// Getter for radius
double getRadius() const {
   return r;
}
```

With proper implementation of information hiding, the designer of a class has full control of what the user of the class can and cannot do.

"const" Member Functions

A const member function, identified by a const keyword at the end of the member function's header, cannot modify any data member of this object. For example,

```
double getRadius() const {  // const member function
   radius = 0;
      // error: assignment of data-member 'Circle::radius' in read-only
structure
   return radius;
}
```

Tutorial 3: Spaceship Finale with Getters and Setters

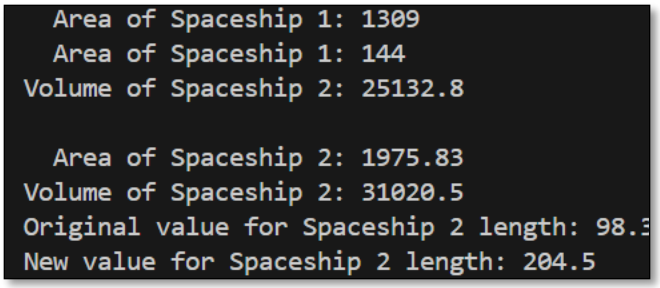
The version of the program brings everything together for a final liftoff.

```cpp
/**
 * Filename: spaceship_finale.cpp
 * Written by:
 * Written on:
 * Create a Spaceship object with parameterized constructor
 */

#include <iostream>

/*****************************************************
 * SPACESHIP CLASS
 *****************************************************/
class Spaceship
{
private:
    /************** PRIVATE DATA MEMBERS **************/
    double m_length;
    double m_width;
    double m_height;

public:
    /************** PUBLIC MEMBER FUNCTIONS ************/
    // Default constructor
    Spaceship() {};

    // Parameterized constructor
    Spaceship(double length, double width, double height)
    {
        m_length = length;
        m_width = width;
        m_height = height;
    }

    // Example getter and setter properties for length
    // Typically you would have getters and setters for all data members
    // Set the length
    double length()
    {
        return m_length;
    }

    // Get the length
    void length(double length)
    {
        m_length = length;
    }
```

```
48      // Method without parameters
49      double calculateArea()
50      {
51          return m_length * m_width;
52      }
53
54      // Method overloading with parameters
55      // C++ looks at the method signature to determine which
56      // method to run
57      double calculateArea(double length, double width)
58      {
59          return length * width;
60      }
61
62      double calculateVolume()
63      {
64          return m_length * m_width * m_height;
65      }
66
67      void insertData(double length, double width, double height)
68      {
69          m_height = height;
70          m_width = width;
71          m_length = length;
72      }
73  };
```

```cpp
int main()
{
    // Create object of Room class
    Spaceship spaceship_1;

    // Assign values to data members
    spaceship_1.insertData(42.5, 30.8, 19.2);

    // Calculate and display the area and volume of the room
    std::cout << "  Area of Spaceship 1: "
              << spaceship_1.calculateArea() << std::endl;
    std::cout << "  Area of Spaceship 1: "
              << spaceship_1.calculateArea(12, 12) << std::endl;
    std::cout << "Volume of Spaceship 2: "
              << spaceship_1.calculateVolume() << std::endl;

    // Create object of Room class with parameterized constructor
    Spaceship spaceship_2(98.3, 20.1, 15.7);

    // Calculate and display the area and volume of the room
    std::cout << "\n  Area of Spaceship 2: "
              << spaceship_2.calculateArea() << std::endl;
    std::cout << "Volume of Spaceship 2: "
              << spaceship_2.calculateVolume() << std::endl;

    // Getter to retrieve the original value
    std::cout << "Original value for Spaceship 2 length: "
              << spaceship_2.length() << std::endl;

    // Setter to assign new value to property
    spaceship_2.length(204.5);

    // Display the new value
    std::cout << "New value for Spaceship 2 length: "
              << spaceship_2.length() << std::endl;

    return 0;
}
```

Example run:



Assignment: Vehicle

Choose your own vehicle. Substitute your own vehicle for tractor. Be creative!

1. Create a C++ program file named **tractor.cpp**
2. Create a class named **Tractor**
3. Create 3 data members
4. Create 3 member methods
    a. One method inserts data.
    b. Overload one method.
5. Create a default and a 3-parameter constructor.
6. Create two Tractor objects. Use each constructor.

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.