# Lecture 5: Object Orientation

Curtin FIRST Robotics Club (FRC) Pre-season Training

Scott Day

265815F@curtin.edu.au

December 1, 2016

Curtin University

# Table of contents

# Object Orientation

# Object Orientation

Core of Object Oriented Programming (OOP) is to create objects, in code, that have certain properties and methods.

While designing C++ modules, we try to see the whole world in the form of objects.

For example, a car is an object which has certain properties such as color, number of doors, and the like. It also has ccertain methods such as accelerate, brake, and so on.

# Overview

There are a few principle concepts that form the foundation of OOP:

**Object** The basic unit of OOP. Both data and function that operate on data are bundled as a unit called an **Object**.

**Class** The blueprint for an object.

**Abstraction** refers to, providing only essential information to the outside world and hiding their background details.

**Encapsulation** is placing data and functions that work on that data in the same place.

**Inheritance** is the process off forming a new class from an existing class that is from the existing class called a base class. The new class formed is called the derived class.

**Polymorphism** is the ability to use a function in different ways.

**Overloading** is also a branch of polymorphism. It allows you to specify more than one definition for a **function name**.

# Classes and Objects

# C++ Class Definitions

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. Then a semicolon or a list of declarations.

```
1  class Box
2  {
3      public:
4          double length;  // Length of a box
5          double breadth; // Breadth of a box
6          double height;  // Height of a box
7  }
```

The keyword **public** determines the access attributes of the members of the class that follow it. You can also specify it as either **private** or **protected**.

# Define C++ Objects

We declare objects of a class with exatly the same sort of declaration that we declare variables of basi types.

```
1 Box box1;
2 Box box2;
```

# Accessing Data Members

Public data members of objects of a class can be accessed using the direct member access operator (.).

```cpp
#include <iostream>

using namespace std;

class Box {
   public:
      double length;  // Length of a box
      double breadth; // Breadth of a box
      double height;  // Height of a box
};

int main() {
   Box box1;            // Declare Box1 of type Box
   Box box2;            // Declare Box2 of type Box
   double volume = 0.0; // Store the volume of a box here

   // box 1 specification
   box1.height  = 5.0;
   box1.length  = 6.0;
   box1.breadth = 7.0;

   // box 2 specification
   box2.height  = 10.0;
   box2.length  = 12.0;
   box2.breadth = 13.0;

   // volume of box 1
   volume = box1.height * box1.length * box1.breadth;
   cout << "Volume of Box1 : " << volume << endl;

   // volume of box 2
   volume = box2.height * box2.length * box2.breadth;
   cout << "Volume of Box2 : " << volume << endl;

   return 0;
}
```

# Classes & Objects in Detail I

So far, we have covered the very basics of C++ Classes and Objects. Here are some further concepts we will discuss at some point.

| | |
|---|---|
| **Class Member Functions** | Functions with it's definition/prototype within the class definition like any other variable. |
| **Class Access Modifiers** | Class members defined as public, private or protected. |
| **Constructor** | Special function in a class that's called when a new object of the class is created. |
| **Destructor** | Special function which is called when a created object is deleted. |
| **Copy Constructor** | Creates an object by initializing it with an object of the same class, which has been created previously. |

# Classes & Objects in Detail II

| | |
|---|---|
| **Friend Functions** | Function that is permitted full access to private and protected members of a class. |
| **Inline Functions** | The compiler tries to expand the code in the body of the function in place of a call to the function. |
| **this pointer in C++** | Every object has a special pointer **this** which points to the object itself. |
| **Pointer to C++ Classes** | A pointer to a class done in the same way a pointer to a structure is. |
| **Static Members of a Class** | Both data and function members of a class can be declared as static. |

# Inheritance

t

t

t

# Overloading

t

t

t

# Polymorphism

t

t

t

# Abstraction

t

t

t

# Encapsulation

t

t

t

# Interfaces

t

t

t

# References I