Introduction to Classes, Structs, and Objects in C++

By Jeremie Bornais

Online Article

View the most up-to-date version of this presentation as an article at
https://blog.bornais.ca/posts/2023-03-21-cxx-classes/

Overview

If you're familiar with programming in C++, you've likely heard of classes, structs, and objects. These concepts are at the core of object-oriented programming and are essential for creating efficient, reusable, and modular code.

In this presentation, we'll take a closer look at what classes, structs, and objects are, how they work, and provide examples to illustrate their use in C++.

Classes and Structs

A class is a user-defined data type that encapsulates data and functions together into a single entity. It serves as a blueprint for creating objects that share common attributes and behaviors. In other words, a class defines the properties and methods that objects of that class will have.

A struct is similar to a class, except that the default access level for its members is public, whereas a class's default access level is private. Structs are typically used for simple data structures, whereas classes are used for more complex objects that have both data and behavior.

Here's an example of a class in C++:

```
class Rectangle {
  private:
    int width;
    int height;
  public:
    void set_values(int width, int height){
      this->width = width;
      this->height = height;
    }

    int area(){
      return width*height;
    }

    int perimeter(){
      return width+width+height+height;
    }
};
```

In this example, we define a class called "Rectangle" with two private member variables, "width" and "height", and three public member functions, "set_values", "area", and "perimeter". The "set_values" function is used to set the values of the width and height attributes, while the "area" and "perimeter" functions calculate the area and perimeter of the rectangle, respectively.

Objects

An object is an instance of a class or struct. It is created by allocating memory for it and initializing its member variables. Once an object is created, its member functions can be called to manipulate its data.

Here's an example of how to create an object of the "Rectangle" class we defined earlier:

```
Rectangle rect;
```

In this example, we create an object of the "Rectangle" class called "rect". By default, the width and height of the rectangle are uninitialized and have undefined values.

To initialize the values of the rectangle's width and height attributes, we can call the "set_values" function:

```
rect.set_values(5, 6);
```

In this example, we call the "set_values" function to set the width and height of the rectangle to 5 and 6, respectively.

We can also call the "area" and "perimeter" functions to calculate the area and perimeter of the rectangle:

```
int area = rect.area();
int perimeter = rect.perimeter();
```

In this example, we call the "area" and "perimeter" functions to calculate the area and perimeter of the rectangle and store the results in the "area" and "perimeter" variables, respectively.

Access Modifiers

Access modifiers are keywords used to define the visibility and accessibility of a class's member variables and functions. There are three access modifiers in C++: public, private, and protected.

Public members can be accessed from outside the class, while private members can only be accessed from within the class. Protected members are similar to private members but can be accessed by derived classes.

Here's an example of how access modifiers are used in a class:

```
class Circle {
  private:
    double radius;
  public:
    void set_radius(double r) {
      radius = r;
    }
    double get_radius() {
      return radius;
    }
  protected:
    double pi = 3.14159;
};
```

In this example, we define a class called "Circle" with a private member variable "radius", a public member function "set_radius" to set the value of "radius", and a public member function "get_radius" to retrieve the value of "radius". We also define a protected member variable "pi" that can be accessed by derived classes.

Inheritance

Inheritance is a powerful feature of object-oriented programming that allows us to create new classes based on existing classes. The new class, called the derived class, inherits the properties and methods of the existing class, called the base class, and can also add its own properties and methods.

Here's an example of how inheritance works in C++:

```
class Shape {
  protected:
    int x;
    int y;
  public:
    void set_position(int x_pos, int y_pos) {
      x = x_pos;
      y = y_pos;
    }
};

class Circle : public Shape {
  private:
    double radius;
  public:
    void set_radius(double r) {
      radius = r;
    }
    double area() {
      return 3.14159 * radius * radius;
    }
};
```

In this example, we define a base class called "Shape" with two protected member variables, "x" and "y", and a public member function "set_position" to set the position of the shape. We also define a derived class called "Circle" that inherits from "Shape" and adds its own private member variable, "radius", and two member functions, "set_radius" and "area", to set the radius of the circle and calculate its area, respectively.

Constructors

Constructors are special member functions that are called when an object of a class is created. They are used to initialize the object's member variables with default or user-defined values. In C++, constructors have the same name as the class and do not have a return type, not even void.

Here's an example of a class with a constructor:

```
class Rectangle {
  private:
    int width;
    int height;
  public:
    Rectangle(int w, int h) {
      width = w;
      height = h;
    }
    int area() {
      return width * height;
    }
};
```

In this example, we define a class called "Rectangle" with two private member variables, "width" and "height", and a constructor that takes two integer parameters and assigns them to the member variables. We also define a public member function "area" that calculates the area of the rectangle and returns it as an integer.

We can create an object of the "Rectangle" class and initialize it using the constructor as follows:

```
Rectangle r(3, 4);
cout << "Area of the rectangle: " << r.area() << endl;
```

This will output "Area of the rectangle: 12", which is the product of the width and height of the rectangle.

Constructors can also be overloaded, meaning we can define multiple constructors with different parameter lists. For example, we can define a default constructor with no parameters that initializes the member variables with default values:

```
class Rectangle {
  private:
    int width;
    int height;
  public:
    Rectangle() {
      width = 0;
      height = 0;
    }
    Rectangle(int w, int h) {
      width = w;
      height = h;
    }
    int area() {
      return width * height;
    }
};
```

Now we can create an object of the "Rectangle" class without providing any parameters, and the default constructor will be called:

```
Rectangle r1;
cout << "Area of the rectangle: " << r1.area() << endl;
```

This will output "Area of the rectangle: 0", since the width and height are initialized to zero by the default constructor.

In summary, constructors are important member functions of a class that are used to initialize objects with default or user-defined values. They can be overloaded with different parameter lists and have the same name as the class. Constructors do not have a return type, not even void.

Conclusion

Classes, structs, and objects are fundamental concepts in C++ and are essential for writing efficient, modular, and reusable code. By encapsulating data and behavior into a single entity, we can create objects that have well-defined attributes and methods. Access modifiers allow us to control the visibility and accessibility of our classes' member variables and functions, while inheritance allows us to create new classes based on existing ones. With these tools at our disposal, we can write powerful and flexible C++ programs.

I hope this article has provided a useful introduction to classes, structs, and objects in C++. Feel free to experiment with the code examples and explore other features of object-oriented programming in C++. Happy coding!

Appendix

Try it Yourself

Here is a code block which contains an overview of all the concepts we've gone over in this article. You can copy and paste this code into your favorite C++ compiler or online IDE, and run it to see the output.

The code defines a class called "Rectangle" with private member variables for width and height, and public member functions for setting and getting the width and height, calculating the area, and printing the rectangle's properties. It also defines a struct called "Point" with public member variables for x and y coordinates.

In the main function, we create two Rectangle objects using different constructors, and manipulate their properties using member functions. We also create a Point object and print its properties.

Now, let's take a look at the code:

```cpp
#include <iostream>
using namespace std;

class Rectangle{
private:
    int width;
    int height;

public:
    Rectangle(){
        width = 0;
        height = 0;
    }
    Rectangle(int w, int h){
        width = w;
        height = h;
    }
```

```
    void setWidth(int w){
        width = w;
    }
    void setHeight(int h){
        height = h;
    }
    int getWidth(){
        return width;
    }
    int getHeight(){
        return height;
    }
    int area(){
        return width * height;
    }
    void print(){
        cout << "Width: " << width << endl;
        cout << "Height: " << height << endl;
        cout << "Area: " << area() << endl;
    }
};
```

```
struct Point{
    int x;
    int y;
};

int main(){
    Rectangle r1;
    Rectangle r2(3, 4);
    r1.print();
    r2.print();
    r1.setWidth(5);
    r1.setHeight(6);
    cout << "Width of r1: " << r1.getWidth() << endl;
    cout << "Height of r1: " << r1.getHeight() << endl;
    cout << "Area of r1: " << r1.area() << endl;
    Point p = {2, 3};
    cout << "Point p: (" << p.x << ", " << p.y << ")" << endl;
    return 0;
}
```

When you run this code, you should see output that looks like this:

```
Width: 0
Height: 0
Area: 0
Width: 3
Height: 4
Area: 12
Width of r1: 5
Height of r1: 6
Area of r1: 30
Point p: (2, 3)
```

This output shows that we have successfully created two Rectangle objects, r1 and r2, using different constructors, and set and printed their properties using member functions. We have also created a Point object p and printed its properties.

You can experiment with this code by modifying the values passed to the constructors and member functions, adding new member functions, or creating additional objects and structs.

Exercise - Zoo Management System

Your task is to create a Zoo management system using C++ classes and objects. The system should allow the user to create a new zoo and add animals to it. Each animal should have a name, species, age, and type of food they eat. The system should also allow the user to view a list of all the animals in the zoo and their details.

Here's some starter code to get you going:

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Animal
{
public:
    string name;
    string species;
    int age;
    string foodType;

    Animal(string name, string species, int age, string foodType)
    {
        // animal constructor
        // fill this in!
    }

    void print()
    {
        // prints the animal to standard output
        // fill this in!
    }
};
```

```
class Zoo
{
public:
    vector<Animal> animals;
    void addAnimal(Animal animal)
    {
        // this function adds an animal to the "animals" vector
        // fill this in!
        // HINT: use push_back with the "animals" vector
    }

    void printAnimals()
    {
        // this function should print out all of the animals and their properties
        // fill this in!
    }
};
```

```
int main()
{
    Zoo myZoo;

    // making some animals
    Animal a1("Jeff", "Lion", 12, "Meat");
    Animal a2("Anne", "Owl", 3, "Meat");
    Animal a3("Zain", "Elk", 6, "Plants");

    // Add some animals to the zoo
    myZoo.addAnimal(a1);
    myZoo.addAnimal(a2);
    myZoo.addAnimal(a3);

    // Print out the animals in the zoo
    myZoo.printAnimals();

    return 0;
}
```

Your Tasks

- Create the constructor for the `Animal` class
- Complete the `print()` function in the `Animal` class to print out an animal
- Complete the `addAnimal()` function in the `Zoo` class to add an animal to the zoo
- Complete the `printAnimals()` function in the `Zoo` class to print out all of the animals. You should use each animals' `print()` function
- Add some sub-classes (for example `Mammal`, `Bird`, `Fish`, etc.) that inherit from the `Animal` class and have additional properties. Overload the `print()` function of each subclass to include the additional properties

Extra Challenges

Here are some extra challenges in case you've already learned this and this seems easy for you:

- Make the application interactive by using a loop and user input in the `main` function. Use your creativity here!
- Add a function `sortAnimals()` to your zoo, which sorts the `animals` vector by an animal's name
- Modify your `sortAnimals()` function to sort based on different properties (like species)
- Look into multiple inheritance and add it into the program. (For example, maybe make a `Flying` subclass of animal and a `Swimming` subclass of animal, and make a third subclass `Duck` which inherits from both `Flying` and `Swimming`)
- **This one's only for the real prodigies** (even I don't really know how to do this): Use an external library to create a GUI for your application
- Solve the infamous P versus NP Problem

Thanks for Coming!

I hope you all learned something new today.

As a reminder, you can find the most up to date version of this presentation here:
https://blog.bornais.ca/posts/2023-03-21-cxx-classes/