Classes and Objects

Matt Warner

Contents

1 Overview

A **class** is a collection of C++ statements that specify the state **(data members)** and implementations of behavior **(member functions)** that a particular type of object may have.

Note:-

A class is not an object, but a description of an object. When we define a class in C++, we are defining a new data type.

To define a new class is C++, we write a class definition. For example, here is a clas definition for a new class named `Student`, which represents information about a student

```
class Student
{
private:

  // Data members
  char name[31];
  double gpa;

public:

  // Constructors
  Student();
  Student(const char*, double);

  // Accessor and mutator member functions
  void set_name(const char*);
  const char* get_name() const;
  void set_gpa(double);
  double get_gpa() const;

  // Other member functions
  void print(); const;
};
```

A class definition starts with the keyword class, followed by the name of the new class data type (Student in this case). The body of the class is enclosed in braces.

The class definition contains variable declarations for the data members of the class and prototypes for the member functions of the class.

Definitions for the member functions of the class are normally placed outside of the class definition.

Note:-

C++ also has the `struct` keyword for defining new composite data types, mostly to maintain backward compatibility with the C programming language, in practice, there is very little difference between a class and a struct in C++, beyond the fact that everything in a class is private by default while everything in a struct is public by default.

2 Access Modifiers

Access modifiers are keywords that modify the accessibility of members of a class. C++ uses three acess modifiers.

- **private**: Accessible in member functions of the class, member functions of friend classes., or by friend functions. Default access for class members
- **Protected**: Accessible in member functions of the class, member functions of friend classes, or by friend functions. Also accessible in member functions of derived classes and friends of those derived classes.
- **public**: accessible in any function that has access to an object of the class. Default access for struct members.

In general, data members that are not constants should usually have private access. Constants and member function prototypes should usually have public access. A constant or member function that is only used within the class (say a sort member function that is only called by another member function of the class) may be made private instead

Note:-

The **protected** keyword is used with inheritance.

3 Constructors

A **constructor** is a special type of member function that is called to create a new object of the class. The usual job of a constructor is to initilize the data members of a new object to valid values.

The name of a constructor is always the same as the name of the class. A constructor has no return type and cannot be constant. The constructor may have 0 to many parameters passed to it.

Through function overloading, a C++ class may have multiple constructors as long as their parameters are different.

If you do not code any constructors for your class, the compiler will generate a "default" constructor without any parameters. The compiler supplied constructor does nothing (effectively it has an empty function body). If you write any constructor for your class, the compiler will not supply this default constructor.

```
Student::Student()
{
  strcpy(name, "None");
  gpa = 0.0;
}

Student::Student(const char* new_name, double new_gpa)
{
  set_name(new_name);
  set_gpa(new_gpa);
}
```

Note:-

If your constructor accepts parameters, do not give them the same names as data members of the class unless you are prepared to use the **this pointer**

4 Accessor and Mutator Member Functions

If the data members of a class are private, how can code outside the class access or modify them?

Since private members are accessible to member functions of the class, we can write a member function to *get* the value of the data member (an accessor member function) and / or to *set* a new value for the data member (a mutator member function). The mutator member function may incorperate some code to ensure that the data member can oly be set to a valid value. For example:

```
void Student::set_name(const char* new_name){
  strcpy(name, new_name);
}

const char* Student::get_name() const
{
  return name;
}

void Student::set_gpa(double new_gpa)
{
  if (new_gpa < 0.0)
    gpa = 0.0;
  else if (new_gpa > 4.0)
    gpa = 4.0;
  else
    gpa = new_gpa;
}

double Student::get_gpa() const
{
  return gpa;
}
```

Since accessor member functions do not modify any data members of the object that called the member function, they should be coded as **constant member functions**

If an accessor member function returns a reference or pointer to a data member, the return type should usually be coded as a reference or pointer to a constant variable. Failure to do so could allow the code that called the accessor method to use the reference or pointer that was returned to modify the data member directly, even though it is private.

Parameters passed to mutator member functions using a referenceor pointer should usually be coded as a reference or pointer to a constant variable. The mutator member function code has no need to change the original variable in the calling routine using the reference or pointer and should not be allowed to do so.

As the following code example illustrates, we cannot directly access private data members of a class, but we can get the value of a data member or modify the value of a data member by calling a public accessor or mutator member function.

```
Student s("Anna Gonzalez", 3.65);

s.gpa = 3.72;     // Syntax error - gpa is private.
cout << s.name << endl;     // Syntax error - name is private.

s.set_gpa(3.72);    // This is correct and will change the gpa data member to 3.72.
cout << s.get_name() << endl;     // This is correct and will print the name.
```

5 Other Member Functions

We can write other member functions for the class as needed. For example, we might write a function to print the data members of an object in a neatly formatted fashion.

If a member function does not modify any data members of the object that called it, the function should be coded as a **constant member function**

6 C++11 initilization Option for Data Members

C++11 added the ability to initilize the non-static data members of a class at the time you declare them using "brace-or-equal" syntax. This is very convenient, and can eliminate most or all of the code from your default constructor. Here are a few examples of the kind of initilizations you can do in a class definition.

```
class InitTest
{
private:

  int a = 0;
  int b{10};

  double c = 9.9;
  double d{23.76};

  bool e = false;
  bool f{true};

  char g = 'a';
  char h{'x'};

  char str1[21]{};    // initilized to an empty string
  char str2[11]{"Joe"};

  string str3 = "Hello";
  string str4{"goodbye"};

  int ar1[4]{2, 4, 6, 8};

  ...
};
```

Even if you initilize all of your data members in your class definition, you may still need to write a default constructor (although the body of the constructor may be entirely empty). If you write at least one constructor, the compiler will not generate a default constructor definition for your class. If you then try to create an object in a way that would call the default constructor, you will get a syntax error.

There are two solutions to this issue.

1. Write an empty default constructor
2. Code = default on a default constructor prototype to instruct the compiler to generate a default constructor definition even though you've written other constructors.