Chapter 20 - Inheritance

Concepts:

Inheritance - a form of software reuse. A new class is created using an existing class's capabilities and then adds to those capabilities.

- The existing class is called the **base class** (or parent class or "superclass").
- The new class is called the **derived class** (the child class or "subclass").
- Inheritance represents the "is-a" relationship. For example, a Car (derived class) is a Vehicle (base class).
- private members of a class are only accessible in the member functions of the class where they are defined.
- private members of a base class are inherited, but not directly accessible in a derived class.
- public members of a class are accessible in any function.
- protected members of a class can be accessed within the body of the class, by friends of the class, and by members and friends of any classes derived from that class.
- C++ allows a class to inherit from more than one base class.
- With public inheritance an object of a derived class can be treated as an object of its base class.

Types of Inheritance

C++ has three types of inheritance: public, protected, and private.

The most commonly used type of inheritance is public.

To specify that a new class is to be publicly-derived from a base class:

```
class DerivedClassName : public BaseClassName {

   // derived class member definitions

   ...

};
```

All data members and member functions of the base class are inherited by the derived class. They may not be directly accessible (if they're declared private), but they are inherited.

Calling a Base Class Constructor

To specify that a constructor of a derived class is supposed to call the constructor of the base class:

```
// in the constructor definition (not the prototype)

DerivedClassName::DerivedClassName ( parameter list )

  : BaseClassName( parameter names to be passed to the base class
constructor )
{
     // body of derived class constructor function
     ...

}
```

The base class constructor is called before the body of the derived class constructor begins execution.

Here's an example. The base class is called Employee and the derived class is SalariedEmployee. In the SalariedEmployee class definition (the header file) we have a prototype for the constructor:

SalariedEmployee.h

```
class SalariedEmployee : public Employee {

public:
   SalariedEmployee( const string &, const string &, const string &, double =
0.0);
    .
    .
    .
```

The prototype is a normal prototype.

SalariedEmployee.cpp

In the function definition for the constructor (the cpp file), we have:

```
SalariedEmployee::SalariedEmployee( const string & first, const string & last,
const string & ssn, double salary ) : Employee (first, last, ssn)
{
   setWeeklySalary ( salary );
}
```

The parameters first, last, and ssn are passed to the Employee constructor. When the Employee constructor has finished, then the body of the SalariedEmployee constructor will execute.

Practice Problems:

Implement a base class **Student** with 3 data members: firstName (string), lastName (string), and studentID (int).

Include a constructor that accepts 3 parameters corresponding to firstName, lastName, and studentID.

Implement a derived class UndergraduateStudent that inherits from Student. UndergraduateStudent should have data members gpa (double) and major (string).

Include a constructor for UndergraduateStudent that accepts 5 parameters corresponding to firstName, lastName, studentID, gpa, and major.