Inheritance

CS A150 – C++ Programming I

Inheritance

- Important concept in **Object-Oriented Programming** (**OOP**)
  - **Inheritance**
    - A mechanism for *enhancing* existing classes
    - General form of class is defined
      - *Specialized* versions then inherit properties of general class
      - And add to it/modify its functionality for its appropriate use

Inheritance Basics

- New class inherited from another class
- **Base class**
  - "General" class from which others derive
- **Derived class**
  - New class
  - Automatically has base class's:
    - Member variables
    - Member functions
  - Can then add additional member functions and variables

Derived Classes

- Consider example:
  Class of "Employees"
- Composed of:
  - Salaried employees
  - Hourly employees
- Each is "subset" of employees
  - Another might be those paid fixed wage each month or week

Derived Classes

- Don’t "need" type of generic "employee"
  - Since no one’s just an "employee"
- General concept of employee is helpful!
  - All have names
  - All have social security numbers
  - Associated functions for these "basics" are same among all employees
- So "general" class can contain all these "things" about employees

Employee Class

- Many members of "employee" class apply to all types of employees
  - **Accessor** functions
  - **Mutator** functions
  - Most data items:
    - SSN
    - Name
    - Pay
- We will *not*, however, have "objects" of this class

Example 1

- Project: Employee Class
  - Employee.h
  - Employee.cpp

  - **NOTE:** The Employee class is *overly* simplified with one member variable only → the employee's SSN

Terminology

- **Base** class also called
  - **Parent** class
  - **Ancestor** class
- **Derived** class also called
  - **Child** class
  - **Descendant** class

Deriving from a Class

- The **derived** class automatically "inherits" from **base** class:
  - Member **variables**
  - Member **functions**
  - But does **NOT** inherit the constructor
- The derived class can add
  - ***New*** member **variables**
  - ***New*** member **functions**

HourlyEmployee Class

- In the **derived** class definition, we declare that the class is derived:

```
class HourlyEmployee : public Employee
```

  - The **:** symbol denotes **inheritance**
  - The keyword **public** is required to be able to invoke an Employee member function on an **HourlyEmployee** object elsewhere
    - If you forget, the compiler will think it is ***private***, which will violate the spirit of using inheritance

HourlyEmployee Class (cont.)

- We do not have to re-declare the variable **`ssn`** since we are inheriting it from the parent class
- But we have a new variable

```
double wageRate;
```

```
class Employee
{
public:
    Employee( );
    Employee(const string& newSSN);
    string getSSN( ) const;
    void setSSN(string newSSN);

private:
    string ssn;
};
```

← Base class definition

```
#include "Employee.h"

class HourlyEmployee : public Employee
{
public:
    HourlyEmployee( );
    HourlyEmployee( const string& newSSN,
                    double newRate);
    void setRate(double newRate);
    double getRate( ) const;

private:
    double wageRate;
};
```

Derived class → definition

HourlyEmployee Class (cont.)

- How do you set the **name** and the **ssn** for an hourly employee?
  - We do ***not*** inherit the base constructor, but
  - We can ***call*** the base constructor

```
HourlyEmployee::HourlyEmployee
  (paramater types...) : Employee (parameters...)
```

- **Note:** If you *omit* the base-class, then the base object is constructed with the default constructor of the base class

Derived class implementation

```
#include "HourlyEmployee.h"

HourlyEmployee::HourlyEmployee()
{
        wageRate = 0.0;   //no need to set ssn
}
HourlyEmployee::HourlyEmployee
                ( const string& newSSN,  double newRate )
                                        :Employee(newSSN)
{
        //set the variable of the derived class only
        wageRate = newRate;
}
void HourlyEmployee::setRate(double newRate)
{
        wageRate = newRate;
}
double HourlyEmployee::getRate( ) const
{
        return wageRate;
}
```

Example 2

- Project: Employee Class
  - HourlyEmployee.h
  - HourlyEmployee.cpp

HourlyEmployee Class Interface

- **Note:** Class definition begins **same** as any other:
  - `#ifndef` structure
  - Includes required libraries
  - Also `#include "Employee.h"`

HourlyEmployee Class Additions

- **Derived** class interface only lists new members
  - Since all others inherited are already defined
  - i.e.: "all" employees have **ssn**
- HourlyEmployee adds:
  - **Constructors**
  - **`wageRate`** variable
  - **`setRate()`** and **`getRate()`** member functions

The `protected` Qualifier

- **Derived** class "inherits" **private** member variables
  - But still **cannot** directly access them
  - Use **`protected`** if you want members to be accessed by all **derived** classes, but *not* by other classes

- **Note:** Many feel this "violates" information hiding

Redefining Functions

- If a derived class requires a different implementation for an inherited member function, the function may be "**redefined**" in the derived class by
  - Listing a declaration in the definition of the derived class (even though the declaration is the same as in the base class)
  - **Redefining** → Must have:
    - ***same*** number and
    - ***same*** type of parameters

```
class Employee
{
public:
    ...
    void print() const;

private:
    string ssn;
};
```

← **Base class definition**
has a function **print**

```
#include "Employee.h"

class HourlyEmployee : public Employee
{
public:
    ...
    void print( ) const;

private:
    double wageRate;
};
```

Derived class definition →
also has a function **print**

Redefining Functions (cont.)

- How can the **derived** print function **print** the member variable (ssn) of the **base** class?
  - **Solution 1:**
    - Call the **parent's print function**
    - Specify that it is the <u>parent's</u> print function and not its own function.
  - **Solution 2:**
    - Call the **parent's accessor function**

Solution 1

```
#include "HourlyEmployee.h"

// Other member functions...

Void HourlyEmployee::print( ) const
{
        Employee::print();              // Calls the parent's
                                        // print function
        cout << wageRate << endl;
}
```

Solution 2

```
#include "HourlyEmployee.h"

// Other member functions...

Void HourlyEmployee::print( ) const
{
        cout << getSSN() << endl;       // Calls the parent's
                                        // accessor function
        cout << wageRate << endl;
}
```

Example 3

- Project: Employee Class
  - SalariedEmployee.h
  - SalariedEmployee.cpp

To Sum Up…

- Functions that are **NOT** inherited:
  - Constructors
  - Private member functions
  - Destructors
  - Assignment operator =
  - Copy constructor → will be automatically generated if not defined, but does not work correctly everywhere, so it is better to define it

Note:
We will cover these two next semester

- **Why** are not these inherited?
  - Because they all need new information that only the child class has
  - For example, new member variables to create the new object

Common Errors

- Private inheritance
  - Forget the keyword public that must follow the colon after the derived-class name

```
class HourlyEmployee : public Employee
```

- Attempting to access **private** base-class fields
  - A derived class inherits all fields from the base class. However, if the fields are ***private***, the derived-class functions **cannot** access them
  - Need to use the ***get*** functions

Multiple Inheritance

- **Derived** class can have **more than one base** class
  - Syntax just includes all base classes separated by commas:
    class derivedMulti : public base1, base2
    {…}
- Possibilities for ambiguity are endless!
- Dangerous undertaking!
  - Some believe should never be used

Remember…

- The ***most*** important feature about inheritance
  - Allows for code re-use

Inheritance (end)