C++ for Science and Engineering COSC3000/6000

2018 Spring Semester

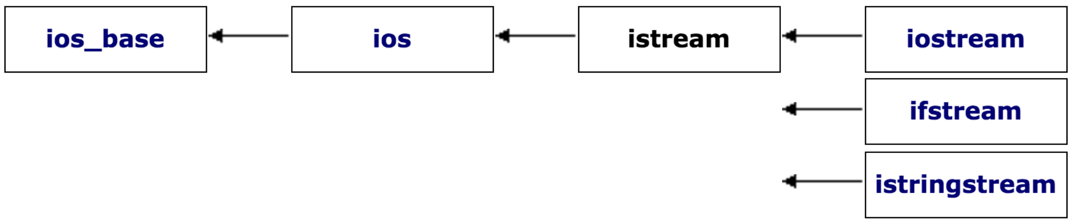
Part XV
Inheritance

1 Inheritance Basics

- Inheritance is the process by which a new class,called a **derived class**, is created from another class, called the **base class**
  - A derived class automatically has all the member variables and functions of the base class
  - A derived class can have additional member variables and/or member functions
  - The derived class is a child of the base or parent class

1.1 Inheritance and Streams

- **cin** and an input-file stream are input streams
  - Input-file streams are members of the class **ifstream**
    - Can be connected to a file
  - **cin** is a member of the class **istream (no 'f ')**
    - Cannot be connected to a file
  - The **ifstream** class is a derived class of the **istream** class



http://www.cplusplus.com/reference/istream/istream/

1.1.1 Stream Parameters

- Example:

```
void two_sum(ifstream& source_file)
{
        int n1, n2;
        source_file >> n1 >> n2;
        cout << n1 " + " << n2 << " = " << (n1 + n2) << endl;
}
```

- This code could be called using
  ```
  ifstream fin;
  fin.open("input.dat");
  two_sum (fin);
  ```
- Suppose you wished to use function **two_sum** with **cin**
- Since **cin** and input-file streams are both input streams, this call to two_sum seems to make sense:
  ```
  two_sum(cin);
  ```
  but it will not work!
- This version of **two_sum** works with **cin**:
  ```
  void better_two_sum(istream& source_file)
  {
          int n1, n2;
          source_file >> n1 >> n2;
          cout << n1 << " + " << n2 << " = " << (n1 + n2) << endl;
  }
  ```
- **better_two_sum** can be called with:
  ```
  better_two_sum(cin);
  ```

1.1.2 Derived Classes and Parameters

- **better_two_sum** can also be called with:
  ```
  ifstream fin;
  fin.open("input.dat");
  better_two_sum(fin);
  ```
- **fin** is of two types
  - **fin** is an input-file stream
  - **fin** is also of type **istream**
  - **fin** has all the features of the input stream class, plus added capabilities
- A formal parameter of type **istream** can be replaced by an argument of type **ifstream**

1.1.3 sample17.cpp

```
void better_two_sum(std::istream& source_file)
{
    int n1, n2;
    source_file >> n1 >> n2;
    std::cout << n1 << " + " << n2 << " = " << (n1 + n2) << std::endl;
}
int main(int argc, const char *argv[])
{
    // input from keyboard
    std::cout << "Input two integer numbers : ";
    better_two_sum(std::cin);
    // input from a file
    std::ifstream fin;
    fin.open("input.dat");
    better_two_sum(fin);
    fin.close();
    return 0;
}
```

1.1.4 Derived Class Arguments

- A restriction exists when using derived classes as arguments to functions
  - A formal parameter of type **istream**, can only use member functions of the **istream** class
  - Using an argument of type **ifstream** with a formal parameter of type **istream** does not allow using the open and close methods of the **ifstream** class!
    - ∗ Open files before calling the function
    - ∗ Close files after calling the function

1.2 Example : Employee Classes

- To design a record-keeping program with records for salaried and hourly employees...
  - Salaried and hourly employees belong to a class of people who share the property "employee"
  - A subset of employees are those with a fixed wage
  - Another subset of employees earn hourly wages
- All employees have a name and SSN
  - Functions to manipulate name and SSN are the same for hourly and salaried employees

1.2.1 A Base Class

- We will define a class called Employee for all employees
- The Employee class will be used to define classes for hourly and salaried employees

```
#ifndef employee_h
#define employee_h
#include <string>
namespace cosc3000
{
    class Employee
    {
    public:
        /// default constructor
        Employee();
        /// constructor takes name and ssn
        Employee(std::string the_name,std::string the_ssn);
        /// get name
        std::string get_name() const;
        /// get ssn
        std::string get_ssn() const;
        /// get new pay
        double get_net_pay() const;
        /// set name
        void set_name(std::string new_name);
        /// set ssn
        void set_ssn(std::string new_ssn);
        /// set net pay
        void set_net_pay(double new_net_pay);
        /// print check (do not call)
        void print_check() const;
    private:
        std::string name;///< Name
        std::string ssn; ///< SSN
        double net_pay;  ///< Net pay
    };
}//cosc3000
#endif /* employee_h */
```

- Implementation

```
#include <iostream>
#include <string>
#include <cstdlib>
#include "employee.h"

namespace cosc3000
{
    Employee::Employee():name("NA"),ssn("NA"),net_pay(0)
    {
        //intentionally blank
    }

    Employee::Employee(std::string the_name,std::string the_ssn)
    :name(the_name),ssn(the_ssn),net_pay(0)
    {
        //intentionally blank
    }
    std::string Employee::get_name() const
    {
        return name;
    }
    std::string Employee::get_ssn() const
    {
        return ssn;
    }
    double Employee::get_new_pay() const
    {
        return net_pay;
    }
    void Employee::set_name(std::string new_name)
    {
        name = new_name;
    }
    void Employee::set_ssn(std::string new_ssn)
    {
        ssn = new_ssn;
    }
    void Employee::set_new_pay(double new_net_pay)
    {
        net_pay = new_net_pay;
    }
    void Employee::print_check() const
    {
        std::cout << "ERROR\n";
        std::cout << "print_check FUNCTION CALLED FOR\n";
        std::cout << "AN UNDIFFERENTIATED EMPLOYEE.\n";
        exit(-1);
    }
}//cosc3000
```

Function print_check

- Function **print_check** will have different definitions to print different checks for each type of employee

- An Employee object lacks sufficient information to print a check
- Each derived class will have sufficient information to print a check

1.2.2 Class HourlyEmployee

- **HourlyEmployee** is derived from Class **Employee**
  - **HourlyEmployee** inherits all member functions and member variables of **Employee**
  - The class definition begins

    ```
    class HourlyEmployee : public Employee
    ```

    - **:public Employee** shows that **HourlyEmployee** is derived from class **Employee**
  - **HourlyEmployee** declares additional member variables **wage_rate** and **hours**

    ```
    #ifndef hourlyemployee_h
    #define hourlyemployee_h
    #include <string>
    #include "employee.h"

    namespace cosc3000
    {
        class HourlyEmployee:public Employee
        {
        public:
            /// default constructor
            HourlyEmployee();
            /// constructor takes name, ssn, wage rate, and hours
            HourlyEmployee(std::string the_name,std::string the_ssn,
                           double the_wage_rate,double the_hours);
            /// set wage rate
            void set_rate(double new_wage_rate);
            /// get wage rate
            double get_rate() const;
            /// set hours worked
            void set_hours(double hours_worked);
            /// get hours worked
            double get_hours() const;
            /// print check
            void print_check();
        private:
            double wage_rate;
            double hours;
        };
    }// cosc30000
    #endif /* hourlyemployee_h */
    ```

Inherited Members

- A derived class inherits all the members of the parent class
  - The derived class does not re-declare or re-define members inherited from the parent, except...
  - The derived class re-declares and re-defines member functions of the parent class that will have a different definition in the derived class
  - The derived class can add member variables and functions

Implementing a Derived Class

- Any member functions added in the derived class are defined in the implementation file for the derived class
  - Definitions are not given for inherited functions that are not to be changed

- The **HourlyEmployee** class is defined as :

```cpp
#include <iostream>
#include "hourlyemployee.h"
namespace cosc3000
{
    HourlyEmployee::HourlyEmployee():Employee(),wage_rate(0),hours(0)
    {
        /// intentionally blank
    }
    HourlyEmployee::HourlyEmployee(std::string the_name,
                                   std::string the_ssn,
                                   double the_wage_rate,
                                   double the_hours)
     :Employee(the_name,the_ssn),wage_rate(the_wage_rate),hours(the_hours)
    {
        /// intentionally blank
    }
    void HourlyEmployee::set_rate(double new_wage_rate)
    {
        wage_rate = new_wage_rate;
    }
    double HourlyEmployee::get_rate() const
    {
        return wage_rate;
    }
    void HourlyEmployee::set_hours(double hours_worked)
    {
        hours = hours_worked;
    }
    double HourlyEmployee::get_hours() const
    {
        return hours;
    }
    void HourlyEmployee::print_check()
    {
        /// Compute net: pay = hours * wage_rate
        set_net_pay(hours * wage_rate);
        std::cout << "_______________________________________________\n";
        std::cout << "Pay to the order of " << get_name() << std::endl;
        std::cout << "The sum of " << get_net_pay() << " Dollars\n";
        std::cout << "_______________________________________________\n";
        std::cout << "Check Stub : NOT NEGOTIABLE\n";
        std::cout << "Employee Number: " << get_ssn() << std::endl;
        std::cout << "Hourly Employee.\n";
        std::cout << "Hours worked: " << hours;
        std::cout << " Rate: " << wage_rate << " Pay: " << get_net_pay() << std::endl;
        std::cout << "_______________________________________________\n";
    }
}// cosc3000
```

1.2.3 Class SalariedEmployee

- The class SalariedEmployee is also derived fromEmployee
  - Function **print_check** is redefined to have a meaning specific to salaried employees
  - **SalariedEmployee** adds a member variable salary

```
#ifndef salariedemployee_h
#define salariedemployee_h

#include <string>
#include "employee.h"

namespace cosc3000
{
    class SalariedEmployee:public Employee
    {
    public:
        /// default constructor
        SalariedEmployee();
        /// constructor takes name. ssn, and weekly salary
        SalariedEmployee(std::string the_name,std::string the_ssn,
                         double the_weekly_salary);
        /// get salary
        double get_salary() const;
        /// set salary
        void set_salary(double new_salary);
        /// print check
        void print_check();
    private:
        double salary;///< Weekly Salary
    };
} // cosc3000
#endif /* salariedemployee_h */
```

  - Implementation is:

```
#include <iostream>
#include "salariedemployee.h"
namespace cosc3000
{
    SalariedEmployee::SalariedEmployee():Employee(),salary(0)
    {
        /// intentionally blank
    }
    SalariedEmployee::SalariedEmployee(std::string the_name,
                                       std::string the_ssn,
                                       double the_weekly_salary)
    :Employee(the_name,the_ssn),salary(the_weekly_salary)
    {
        /// intentionally blank
    }
    double SalariedEmployee::get_salary() const
    {
        return salary;
    }
    void SalariedEmployee::set_salary(double new_salary)
    {
        salary = new_salary;
    }
    void SalariedEmployee::print_check()
    {
        /// Compute net: pay = salary
        set_net_pay(salary);
        std::cout << "_______________________________________________\n";
        std::cout << "Pay to the order of " << get_name() << std::endl;
        std::cout << "The sum of " << get_net_pay() << " Dollars\n";
        std::cout << "_______________________________________________\n";
        std::cout << "Check Stub : NOT NEGOTIABLE\n";
        std::cout << "Employee Number: " << get_ssn() << std::endl;
        std::cout << "Salaried Employee. Regular Pay:" << salary << std::endl;
        std::cout << "_______________________________________________\n";
    }
}//cosc3000
```

Parent and Child Classes

- Recall that a child class automatically has all the members of the parent class
- The parent class is an **ancestor** of the child class
- The child class is a **descendent** of the parent class
- The parent class (Employee) contains all the code common to the child classes
  - You do not have to re-write the code for each child

Derived Class Types

- An hourly employee is an employee
  - In C++, an object of type **HourlyEmployee** can be used where an object of type **Employee** can be used
  - An object of a class type can be used wherever any of its ancestors can be used
  - An ancestor **cannot** be used wherever one of its descendent can be used

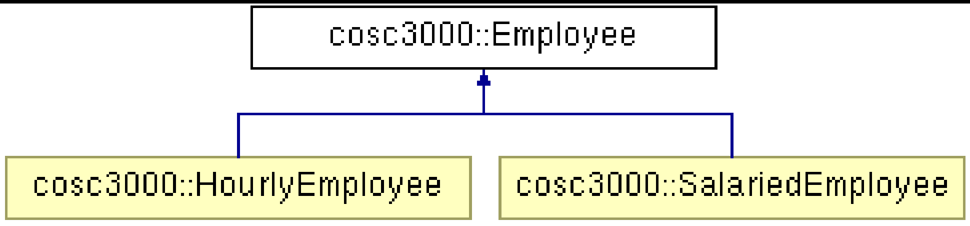
Derived Class Constructors

- A parent class constructor is not inherited in a derived class
  - The parent class constructor can be invoked by the constructor of the derived class
  - The constructor of a derived class begins by invoking the constructor of the parent class in the initialization section:

    ```
    HourlyEmployee::HourlyEmployee() : Employee( ), wage_rate(0), hours(0)
    {
            //no code needed
    }
    ```

  - Default constructor of parent class is automatically called if not explicitly specified.

    ```
    HourlyEmployee::HourlyEmployee() : wage_rate(0), hours(0)
    {
            // Default constructor of Employee class is automatically called
    }
    ```



1.2.4 Default Initialization

- If a derived class constructor does not invoke a base class constructor explicitly, the base class default constructor will be used
- If class B is derived from class A and class Cis derived from class B
  - When a object of class C is created
    - The base class A's constructor is the first invoked
    - Class B's constructor is invoked next
    - C's constructor completes execution

1.2.5 Private is Private

- A member variable (or function) that is private in the parent class is not accessible to the child class
  - The parent class member functions must be used to access the private members of the parent
  - This code would be **illegal**:

```
void HourlyEmployee::print_check( )
{
        net_pay = hours * wage_rage;
        :
```

    - **net_pay** is a private member of Employee!

1.2.6 The protected Qualifier

- protected members of a class appear to be private outside the class, but are accessible by derived classes
  - If member variables name, **net_pay**, and **ssn** are listed as **protected** (not private) in the **Employee** class, this code, illegal above, becomes **legal**:

```
void HourlyEmployee::print_check( )
{
        net_pay = hours * wage_rage;
        :
```

1.2.7 Programming Style

- Using protected members of a class is a convenience to facilitate writing the code of derived classes.
- Protected members are not necessary
  - Derived classes can use the public methods of their ancestor classes to access private members
- Many programming authorities consider it bad style to use protected member variables

1.2.8 Redefinition of Member Functions

- When defining a derived class, only list the the inherited functions that you wish to change for the derived class
  - The function is declared in the class definition

- **HourlyEmployee** and **SalariedEmployee** each have their own definitions of **print_check**

```
///
/// sample48.cpp
///
#include <iostream>
#include "hourlyemployee.h"
#include "salariedemployee.h"

int main(int argc, char **argv)
{
    cosc3000::HourlyEmployee hideki;
    hideki.set_name("Hideki Fujioka");
    hideki.set_ssn("123-45-6789");
    hideki.set_rate(20.50);
    hideki.set_hours(40);
    std::cout << "Check for " << hideki.get_name()
            << " for " << hideki.get_hours() << " hours.\n";
    hideki.print_check();
    std::cout << std::endl;

    cosc3000::SalariedEmployee boss("Dr. Big Shot","987-65-4321",10500.50);
    std::cout << "Check for " << boss.get_name() << std::endl;
    boss.print_check();
    return 0;
}
```

1.2.9 Redefining or Overloading

- A function **redefined** in a derived class has the same number and type of parameters
  - The derived class has only one function with the same name as the base class
- An **overloaded** function has a different number and/or type of parameters than the base class
  - The derived class has two functions with the same name as the base class
    - One is defined in the base class, one in the derived class

1.2.10 Access to a Redefined Base Function

- When a base class function is redefined in a derived class, the base class function can still be used
  - To specify that you want to use the base class version of the redefined function:

```
HourlyEmployee sally_h;
sally_h.Employee::print_check( );
```

1.3 Documented by Doxygen

`http://www.stack.nl/~dimitri/doxygen/`

See : `https://www.googledrive.com/host/0BynBZDpf_6IbM3BGWkRRUkdnVW8/C++2016/html/index.html`