Inheritance & Polymorphism

Inheritance

Significant Principles

Inheritance: the language feature that makes it possible to pass features from one class to another is called inheritance——`eg.` `ostream` is termed a **base class**, `ofstream` is termed a **derived class**.

- 当 Derived class is inheriting from base class，我们可以把**子类的对象都看成是父类的对象**；
- Base class 中的**所有东西 Derived class 都有**，但是 **Base class 的 private 成员在 Derived class 中不可见**；
  - Derived class 继承了所有 base class 的方法，意味着派生类的对象可以使用基类的方法；
- Base class is always constructed first, **Derived class 在构造的时候总是先构造一个 Base class**：
  - Derived class 可以看作是**拥有 Base class 整个完整的结构**，内存结构也是相同的；
  - 如果 Base class 没有默认构造函数，我们在直接构造一个 Derived class 的时候必须要 **explicitly 传递构造 Base class 的参数**，才能通过编译；
- A base class reference can refer to a derived class object without requiring a type cast **(基类引用可以指向派生类对象)**：
  - 所以可以定义一个接受基类引用作为参数的函数，而**在调用函数时，既可以将基类对象作为参数，也可以将派生类对象作为参数**——`eg.` 参数类型为 `ostream &` 的函数可以接受 `ofstream` 对象.

Examples

`eg.` base class: `Employee` class, derived class: `Manager` class

```cpp
class Employee {
    public:
        Employee(const std::string& name, const std::string & ssn);
        const std::string& get_name() const;
        void print(std::ostream& out) const;
        void print(std::ostream& out, const std::string & msg) const;
    private:
        std::string m_name;
        std::string m_ssn;
}
Employee::Emplyee(const string& name, const string& ssn)
    : m_name(name), m_ssn(ssn) {
        // initializer list sets up the values!
    }
inline const std::string& Employee::get_name() const {
    return m_name;
}
inline void Employee::print(std::ostream& out) const {
    out << m_name << endl;
    out << m_ssn << endl;
```

```
21  }
22  inline void Employee::print(std::ostream& out,
23                              const std::string& msg) const
24  {
25      out << msg << endl;
26      print(out);
27  }
```

- Now add Manager class(derived class) `class Manager : public Employ:`

```
1   class Manager : public Employ {
2       public:
3           Manager( const std::string& name,
4                    const std::string& ssn,
5                    const std::string& title );
6           const std::string title_name() const;
7           const std::string& get_title() const;
8           void print( std::ostream& out ) const;
9       private:
10          std::string m_title;
11  }
```

  - Think of inherited traits as an embedded object.
  - Base class is mentioned by class name `Employee::...`

```
1   Manager::Manager( const string& name, const string& ssn, const
    string& title="" ) : Employee(name, ssn), m_title( title ) {}
2
3   inline void Manager::print( std::ostream& out ) const {
4       Employee::print( out );      //call the base class print
5       out << m_title << endl;
6   }
7
8   inline const std::string& Manager::get_title() const {
9       return m_title;
10  }
11
12  inline const std::string Manager::title_name() const {
13      return string(m_title + ": " + m_name);//access base m_name
14  }
```

    - Uses:

```
1   int main () {
2       Employee bob( "Bob Jones", "555-44-0000" );
3       Manager bill( "Bill Smith", "666-55-1234",
4                     "ImportantPerson" );
5       string name = bill.get_name(); // okay Manager
6       inherits Employee
7       //string title = bob.get_title(); // Error --
8                                   // bob is an Employee!
9       cout << bill.title_name() << '\n' << endl;
10      bill.print(cout);
11      bob.print(cout);
12      bob.print(cout, "Employee:");
```

```
13        //bill.print(cout, "Employee:"); // Error hidden!
14    }
```

- If you **redefine** a member function in the **derived class**, **all other overloaded functions** in the base class are **inaccessible**.
- ☑️ What is not inherited?
  - Constructors，Base class 的构造函数在 Derived class 中是不存在的
    - Synthesized constructors use member wise initialization;
    - In explicit copy ctor, explicitly call base-class copy ctor or the default ctor will be called instead.
  - Destructors，Base class 的析构函数在 Derived class 中是不存在的
  - Assignment operation，运算符重载是不会从 Base class 传到 Derived class 的
    - synthesized operator `=` uses member wise assignment;
    - explicit operator `=` be sure to explicitly call the base class version of operator `=`
  - Private data is hidden, but still present.

Access Protection

- Members:
  - `public:` visible to all clients;
  - `protected:` visible to classes derived from self (and to friends);
  - `private:` visible **only to self** and to friends，在子类中有，但是访问不了；
- Inheritance:

```
1  public:     class Derived : public Base ...
2  protected:  class Derived : protected Base ...
3  private:    class Derived : private Base ...
```

Suppose class `B` is derived from `A`, then:

Inheritance Type (`B` is)	public	protected	private
public `A`	public in `B`	protected in `B`	hidden
private `A`	private in `B`	private in `B`	hidden
protected `A`	protected in `B`	protected in `B`	hidden

Base class 的 `private` 变量在 Derived class 中存在，但是在 Derived class 不能访问；但是如果 Base class 的 `protected` 变量在 Derived class 中存在，并且可以直接访问。

- examples:

```
1  class Base {
2  protected:
3      int alpha = 1;
4  public:
5      int blphb = 2;
6      Base( int k ) : alpha(k) {
7          cout << "Base()\n";
8      }
```

```
 9        ~Base() {
10            cout << "~Base()\n";
11        }
12        void prt() {
13            cout << alpha << " " << blphb << endl;
14        }
15    };
16
17    class Derived : public Base{
18    private:
19        int clphc = 3;
20    public:
21        Derived() : Base(9) {
22            cout << "Derived()\n";
23        }
24        ~Derived() {
25            cout << "~Derived()\n";
26        }
27        void prt() {
28            Base::prt();
29            cout << alpha << endl;      // Can access alpha of Base class if
   it is protected
30            cout << blphb << endl;
31            cout << clphc << endl;
32        }
33    };
34
35    class Eerived : public Derived {
36    public:
37        void pp() {
38            cout << alpha << endl;  // Can access alpha of Base class if it
   is protected
39        }
40    };
41
42    int main() {
43        /* The order of constructor */
44        Eerived ed;
45        ed.pp();
46        return 0;
47    }
48    ---------------------------------------------------------------------
49    Base()
50    Derived()
51    9
52    ~Base()
53    ~Derived()
```

- alpha 在 Base class 中是 protected 的，那么 alpha 在 public Derived class of Base class 以及 public derived class of Derived class 中都是可以进行访问的。

    - 如果 alpha 在 Base class 中是 private 的，那么在 Derived classes 都不可访问；

```
1    class Base {
2    private:
3        int alpha = 1;
4    public:
```

```
5      int blphb = 2;
6      Base( int k ) : alpha(k) {
7          cout << "Base()\n";
8      }
9      ~Base() {
10         cout << "~Base()\n";
11     }
12     void prt() {
13         cout << alpha << " " << blphb << endl;
14     }
15 };
16
17 class Derived : public Base{
18 private:
19     int alpha = 666;
20     int clphc = 3;
21 public:
22     Derived() : Base(9) {
23         cout << "Derived()\n";
24     }
25     ~Derived() {
26         cout << "~Derived()\n";
27     }
28     void prt() {
29         Base::prt();
30         cout << alpha << endl;      // Can access alpha of Base class if
   it is protected
31         cout << blphb << endl;
32         cout << clphc << endl;
33     }
34 };
35
36 int main() {
37     /* The order of constructor */
38     Derived td;
39     int *r = (int*)&td;
40     for ( int i = 0; i < sizeof(Derived) / sizeof(int); i++ ) {
41         cout << r[i] << " ";
42     }
43 }
44 ----------------------------------------------------------------------
45 Base()
46 Derived()
47 9 2 666 3
48 ~Base()
49 ~Derived()
```

- Derived 中含有：Base class 的 `private` 的 `alpha`，`public` 的 `blphb`，以及 Derived class 自己的 `private` 的 `alpha, clphc`
  - Base class 的 `alpha` 和 Derived 自己定义的 `alpha` 是互不影响的，Base class 的成员变量 Derived class 都有，且放在前面的内存空间；