C++面向对象

Tags	theories
Last edited time	@December 27, 2023 12:09 PM

构造函数

```
// 定义构造函数
Box(int h, int w, int len):height(h), width(w), length(len){}

// 显式调用构造函数
Box b1 = Box(6,6,6);

// 隐式调用构造函数
Box b1 = {6,6,6};
// 注意：隐式法无法调用无参构造函数，也就是不能写成: Box b3={};
```

1. 构造函数中new一个东西，在析构函数中一定要delete掉
2. 只能为最后的参数设置默认值（要给前面的元素设默认值，那么也得给后面的元素设默认值）
3. 构造函数的初始化是**从右至左**的

继承与派生

派生类构造函数和析构函数的执行顺序

通常情况下,当创建派生类对象时, **首先执行基类的构造函数, 随后再执行派生类的构造函数**; 当撤消派生类对象时, 则**先执行派生类的析构函数, 随后再执行基类的析构函数（需要动态绑定，即把父类析构函数设置成虚函数，否则将只执行父类的析构函数）**。

派生类构造函数和析构函数的构造规则

当基类的构造函数没有参数,或没有显式定义构造函数时, 派生类可以不向基类传递参数,甚至可以不定义构造函数。**派生类不能继承基类中的构造函数和析构函数**。当基类含有带参数的构造函数时,派生类必须定义构造函数,以提供把参数传递给基类构造函数的途径。

多重继承

1. 使用成员名限定可以**消除二义性**,例如:

- obj .X∶∶f( ) ;

  / / 调用类 X 的 f( )

- obj .Y∶∶f( ) ;

  / / 调用类 Y 的 f( )

2. 多重继承的构造函数

```
派生类构造函数名(参数表) :基类 1 构造函数名 ( 参数表), 基类 2 构造函数名
{
    // …
}

class Hard
{
protected:
    char bodyname[20];
public:
    Hard(char * bdnm );     // 基类 Hard 的构造函数
    // …
};
class Soft
{
protected:
    char os[10];
    char Lang[15];
public:
```

```
    Soft( char * o, char * lg);// 基类 Soft 的构造函数
    // …
} ;
class System: public Hard, public Soft
{
private:
    char owner[10] ;
public:
    System( char * ow, char * bn, char * o, char * lg) // 派生类
    :Hard( bn), Soft(o, lg);
    // 缀上了基类 Hard 和 Soft 的构造函数
    // …
};
```

运算符重载

不能重载的运算符有：

- **.**：成员访问运算符
- **.***, **->***：成员指针访问运算符
- **::**：域运算符
- **sizeof**：长度运算符
- **?:**：条件运算符
- **#**： 预处理符号

能重载的运算符有：其他

双目算术运算符	+ (加)，-(减)，*(乘)，/(除)，% (取模)
关系运算符	==(等于)，!= (不等于)，< (小于)，> (大于)，<=(小于等于)，>=(大于等于)
逻辑运算符	||(逻辑或)，&&(逻辑与)，!(逻辑非)
单目运算符	+ (正)，-(负)，*(指针)，&(取地址)
自增自减运算符	++(自增)，--(自减)
位运算符		(按位或)，& (按位与)，~(按位取反)，^(按位异或),，<< (左移)，>>(右移)

赋值运算符	=, +=, -=, *=, /= , % = , &=, |=, ^=, <<=, >>=
空间申请与释放	new, delete, new[ ] , delete[]
其他运算符	**()**(函数调用)，**->**(成员访问)，**,**(逗号)，**[]**(下标)

运算符重载的参数类型

```
// 成员函数重载"+"
Complex operator+(Complex &c2){
        Complex tmp(this->a + c2.a, this->b + c2.b);
        return tmp;
    }

// 友元函数重载"+"
friend Complex operator+(Complex &c1, Complex &c2);
Complex operator+(Complex &c1, Complex &c2){
    Complex tmp(c1.a + c2.a, c1.b + c2.b);
    return tmp;
}

// 成员函数重载前置"++"
Complex& operator++(){
        this->a++;
        this->b++;
        return *this;
    }

// 友元函数重载前置"++"
friend Complex& operator++(Complex &c1);
Complex& operator++(Complex &c1){
    c1.a++;
    c1.b++;
    return c1;
}

// 成员函数重载后置"++"
Complex operator++(int){
```

```
        Complex tmp = *this;
        this->a++;
        this->b++;
        return tmp;
    }

// 友元函数重载后置"++"
friend Complex operator++(Complex &c1, int);
Complex operator++(Complex &c1, int){
    Complex tmp = c1;
    c1.a++;
    c1.b++;
    return tmp;
}

// 友元函数重载输入/输出运算符
friend ostream &operator<<(ostream &out , const Point &a);
ostream &operator<<(ostream &out , const Point &a){
    out << "<Point>( " << a.x << ", " << a.y << ")";
    return out;
}
```

虚函数与纯虚函数

定义

- 类中声明前带有`virtual`关键字的函数称为虚函数：

```
class A {
    virtual void example() {}
}
```

- 类中申明格式如下的函数称为纯虚函数：

```
class A {
    virtual void example() = 0;
}
```

即在函数声明中含有`virtual`和`= 0`两个关键字。

区别

- 纯虚函数的特点：
  - 只有声明，没有实现/定义
  - 含有纯虚函数的类称为抽象类，抽象类不能被实例化
  - 抽象类的派生类如果想成为具体的类（能够被实例化），则必须重写纯虚函数。
- 虚函数的特点：
  - 必须实现/被定义
  - 虚函数所在类可以被实例化

函数重写与函数重载

- 重写(overreide):
  指派生类中存在重新定义的函数。其函数名，参数列表，返回值类型，
  所有都必须同基类中被重写的函数一致。只有函数体不同（花括号内），**派生类调用时会调用派生类的重写函数**，不会调用被重写函数。基类中被重写的函数必须是虚函数/纯虚函数。
- 重载(overload):
  指同一可访问区内被声明的几个具有
  不同参数列表（参数的类型，个数，顺序不同）的同名函数，根据参数列表确定调用哪个函数，重载**不关心函数返回类型**。

区别：

1. 范围区别：**重写**和被重写的函数在**不同的类**（基类和派生类）中，**重载**和被重载的函数在**同一类**中。

2. 参数区别：**重写**与被重写的函数**参数列表一定相同**，**重载**和被重载的函数**参数列表一定不同**。
3. virtual的区别：**重写**的基类必须要有**virtual**修饰，重载函数和被重载函数可以被virtual修饰，也可以没有。

动态绑定

动态多态：子类重写父类的虚函数

小坑

1. 使用父类指针创建对象时，要用`->`来指向方法
2. 类中声明变量时，不能赋初值
3. 对于不希望改变的参数，用`const`来修饰