面向对象程序设计(OOP)

C++(第3章：类和对象)

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)

对象数组

- 创建对象数组的方法与创建其他数据类型的数组相同。例如：

  创建对象数组实例

- 同样可以对对象数组进行初始化，例如：

  单个参数的对象初始化方法

  多个参数的对象初始化方法

动态分配与释放对象

- 动态建立和删除对象：

```
Myclass *p; //p为指向对象的一个指针
p=new Myclass();//等价于p=new Myclass;
/*使用new给p分配空间(仅为数据成员)*/
//…….用形如p->year格式使用数据成员
delete p;//delete释放p所指向的内存空间
```

- 对象动态内存分配和释放实例1

第3章：类和对象

动态分配与释放对象

- 可用p=new Myclass(…);调用(缺省)构造函数设置数据成员的初始值。
- 用p=new Myclass[…];为对象数组申请空间(不能再设初始值，所以必须有无参构造函数)；但必须用delete[] p;释放对象数组指针，否则执行时将报告错误(编译不报错)。
- delete语句会自动执行相应类的析构函数。
- 对象动态内存分配和释放实例2

常对象与常对象成员

- 常对象是指对象常量，其定义格式为：

  const 类名 对象名(……)；

- 常对象具有如下特点：

  1、常对象在定义时必须初始化，而且在程序中不允许再对其进行更新；

  2、通过常对象只能调用类中的常成员函数，而不能调用类中的非常成员函数。

第3章：类和对象

常对象与常对象成员

- 用const说明的成员函数称为常成员函数，说明格式如下：
  类型 成员函数名（形参表）const;
- 常成员函数不能改变对象的成员变量的值；也不能调用一个非const的成员函数（否则就可能通过它间接修改数据成员）。
- 在函数原型和定义处均必须加上const。
- const关键字可以参与区分重载函数。
- 常成员函数实例

第3章：类和对象

常对象与常对象成员

- 用const说明的数据成员称为常数据成员。
- 不能在构造函数中直接用赋值语句对常数据成员进行赋值，需要利用构造函数所附带的初始化表进行初始化，即在构造函数的括号后面加上“：”和初始化表，其格式为：

```
类名::类名(参数表):成员初始化参数表 {
    //构造函数的函数体
}//包含引用或常成员的类不能使用默认构造
```

- 常数据成员的初始化实例　　圆类实例

对象赋值

- 如果两个对象是同一种类型，则其中一个对象就可以赋给另一个对象，但两个对象仍然是完全独立的。例如：

  对象赋值实例

- 不同类的对象不能进行赋值，即使两个类具有完全相同的成员。

  不同类的对象不能进行赋值实例

第3章：类和对象

对象赋值

- 赋值运算符的重载实例

  不能重载为友元，否则报告下述错误：error C2801: ’operator =’ must be a <Unknown> member

- 对对象进行赋值可能有副作用。例:如果对象A包含指向另一个对象的B的指针,在复制A时,在副本中也将包含指向B的字段。因此对对象B进行修改就会影响到两个对象。此时需要重载赋值运算符。 对象赋值副作用实例

第3章：类和对象

对象赋值：必须背下来

```
List &List::operator=(const List &x) {
  if (this!=&x) {//重载时,必须当心自赋值
   !
      delete[] data;
      n=x.n;
      MaxSize=x.MaxSize;
      data=new int[MaxSize];
      for (int i=0;i<n;i++) data[i]=x.data[i];
  }
  return *this;
}
```


解决对象赋值副作用实例

拷贝(复制)构造函数

- 拷贝/复制构造函数是一种特殊的构造函数。
- 函数原型：类名（const类名 &）；
  //必须使用引用参数，应为public访问权限
- 作用：通过建立已有类对象的副本来初始化新的对象，即用一个对象去构造另一个对象。
- 不论何时需要复制对象时，都会调用拷贝构造函数。

第3章：类和对象

拷贝（复制）构造函数

- 当类的定义中没有拷贝构造函数，则编译系统将自动产生一个具有上述形式的默认的拷贝构造函数，作为该类的公有成员。
- 拷贝构造函数的调用时机(掌握)：

1) 用一个已定义的类对象初始化另一个被定义的同类对象时，
2) 在类的实参值传送给对应值参的过程中，
3) 在函数中将类对象作为值返回时。

第3章：类和对象

传递对象给函数

- 值调用：创建对象的副本（不会调用类的构造函数，调用类的拷贝构造函数，也调用类的析构函数---所以**影响运行时间**），不会改变实参对象的值。
- 引用调用：不会创建对象的副本（当然也就不会调用类的构造函数、拷贝构造函数、析构函数等），但改变实参对象的值。

第3章：类和对象

拷贝（复制）构造函数

- 与对象赋值可能有副作用一样。默认拷贝构造函数也可能存在致命的问题。解决办法是编写自己的拷贝构造函数。拷贝构造函数的编写与重载赋值运算符=的程序基本类似。正常情况下，**一旦重载赋值运算符同时也必须编写自己的拷贝构造函数**。
- 自己编写拷贝构造函数实例

第3章：类和对象

拷贝（复制）构造函数(背)

```
List::List(const List &x) {
  //自定义拷贝构造函数实现“深拷贝”
  n=x.n;
  MaxSize=x.MaxSize;
  data=new int[MaxSize];
  for (int i=0;i<n;i++)
     data[i]=x.data[i];
}
```

第3章：类和对象

综合实例：对象赋值和拷贝构造函数

- Myclass x(y);完全等价于Myclass x=y;它们均调用(默认)拷贝构造函数。但不完全等价于Myclass x;x=y;它执行(默认)赋值操作。因此**编写自己的拷贝构造函数的同时一般也要重载赋值运算符=**。
- 对象按引用调用或返回对象引用时，由于不制作对象副本所以并不调用拷贝构造函数。
- 同普通变量一样，对象按值调用并不影响实参对象的值。

第3章：类和对象

综合实例：对象赋值和拷贝构造函数

- 对象按值调用时，系统将调用拷贝构造函数制作对象的副本，并在函数返回时调用析构函数销毁对象。
- 函数返回对象前，系统先调用拷贝构造函数制作对象的副本，并在函数返回完成(赋值)操作后调用析构函数销毁对象。
- 对象赋值和拷贝构造函数综合实例

第3章：类和对象

系统默认成员函数总结(背)

- 系统默认构造函数：有自定义只要构造函数时默认构造函数失效，此时还需要无参构造函数时必须自己创建一个
- 系统默认析构函数：可自定义
- 系统默认拷贝构造函数、系统默认赋值函数：一般创建自己的拷贝构造函数同时也必须重载赋值运算符。

类作用域

- 局部对象的生存期与局部变量一样。
- 当成员函数中定义了同名的局部变量时，则必须使用作用域运算符“::”来限定，如：

```
class Myclass {
   int m;
public:
   Myclass(int m){    //构造函数
      Myclass::m=m;   //或this->m=m;
   }
};
```

第3章：类和对象

变量和类同名

- **变量可以和struct、class、union、enum类型同名**，此时声明必须加上其前缀struct、class、union、enum等。实例：


```
class i {
public:
  int ix;
};
int i=10;
class i j;
j.ix=20;
```

结构与类的区别

- 声明结构的一般形式如下：

```
struct  <结构名> {
[public:]  //默认为public成员
   [<公有数据成员和成员函数>]//外部接口
private:
   [<私有数据成员和成员函数>]
protected:
   [<保护数据成员和成员函数>]
} [对象列表]; //其他与类完全相同
```

联合(共用体union):了解

- union Utype { //定义联合类型，共享内存
  short int i;
  char ch;
  } u_var;//使用的唯一目的是节省内存空间
- 声明变量：Utype u_var;
- 本质上就是类，所有元素共用内存。成员默认是公有的。**可以包含成员函数、构造和析构函数等，很少用，一般只包含数据**。使用时必须遵守一些限制条件(参见其他书籍)。

匿名(anonymous)联合:了解

- 匿名联合(共用体)不包括类型名称，不能声明变量。匿名联合用于通知编译器其成员变量共享同一个内存单元，可以直接引用变量而不用.限定，也不能和同一个作用域的变量同名。使用匿名联合的唯一目的同样是为了节省内存空间。
- 匿名联合(共用体)使用实例

多文件结构

- 惯用的方法是：

  类声明文件（*.h），必须包含内联函数

  类定义/实现文件（*.cpp）

  类的使用文件/主函数文件（*.cpp）

- 实例(类声明、类定义、类使用)

第3章：类和对象

多文件结构

- 如果将类的内联函数放在类定义体的外面(用inline声明)，则必须保证将它们的实现和类的定义放在一个头文件中，否则调用函数展开时将找不到函数体。

类与对象复习

编写、测试类的步骤：

1、确定数据成员

2、编写构造、析构函数（需要时），并测试

3、编写set和get族成员函数，重载<<和>>运算符，并对它们进行测试

4、编写其他成员/友元函数、重载其他需要的运算符(一般一次编写一个测试一个)，然后进行全面的测试。

第3章：类和对象

综合实例：Complex

```
//自定义复数类
class Complex {
  private:
    double rpart;          //实部
    double ipart;          //虚部
    double abs() const; //计算负数的模
    double norm() const;//负数模的平方
  public:
    //……
```

第3章：类和对象

综合实例：Complex

```
Complex(); //缺省构造函数
Complex(const double &r);
//(类型)转换构造函数
Complex(const double &r,const double &i);
//初始化构造函数
Complex(const Complex &c);//拷贝构造函数
void setValue(const double &r,const
   double &i); //设置复数值
```

第3章：类和对象

综合实例：Complex

```
//重载运算符
Complex &operator=(const Complex &c);//赋值=
Complex operator-();                 //负号-
//重载算术运算符
friend Complex operator+
        (const Complex &c1,const Complex &c2);
friend Complex operator-
        (const Complex &c1,const Complex &c2);
friend Complex operator*
        (const Complex &c1,const Complex &c2);
friend Complex operator/
        (const Complex &c1,const Complex &c2);
```

第3章：类和对象

综合实例：Complex

```
//重载算术运算符
Complex operator+=(const Complex &c);
Complex operator-=(const Complex &c);
Complex operator*=(const Complex &c);
Complex operator/=(const Complex &c);
 //重载>>和<<，不能重载为类的成员函数
friend istream &operator>>
        (istream &is,Complex &c);
friend ostream &operator<<
        (ostream &out,const Complex &c);
```

第3章：类和对象

综合实例：Complex

```
//重载关系（比较）运算符
friend bool operator==
      (const Complex &c1,const Complex &c2);
friend bool operator!=
      (const Complex &c1,const Complex &c2);
friend bool operator>
      (const Complex &c1,const Complex &c2);
friend bool operator>=
      (const Complex &c1,const Complex &c2);
friend bool operator<
      (const Complex &c1,const Complex &c2);
friend bool operator<=
      (const Complex &c1,const Complex &c2);
```

第3章：类和对象

综合实例：Complex

```
Complex::Complex() {//缺省构造函数
  rpart=ipart=0.0;
}
```



```
Complex::Complex(const double &r) {
  //（类型）转换构造函数
  rpart=r;
  ipart=0.0;
}
```

第3章：类和对象

综合实例：Complex

- 完整的程序(**课后务必认真阅读、测试**)：

1、Complex.h

2、Complex_utility.cpp

3、Complex_arithmetic_operator.cpp

4、Complex_compare_operator.cpp

5、Complex_main.cpp

本章内容讲授到此结束！

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)