课程复习

第六章 数据抽象——类

基本概念

- 1. 对象
  - 是要研究的客观事物的抽象，是由数据及其操作所构成的封装体
- 2. 类
  - 是对象的模板，描述了一组具有相同属性和操作的对象
- 3. 方法
  - 指定义于某一特定类上的操作与法则。
- 4. 消息
  - 调用对象的操作，由对象名、消息名和实际变元构成
- 5. 通信
  - 指对象之间的消息传递，是引起面向对象程序进行计算的唯一方式

基本问题

- 1. 面向对象程序设计的主要特征是什么？
- 答：面向对象程序设计的主要特征是封装性，继承性和多态性，具体解释如下
  - 封装性：把数据和实现操作的代码集中起来放在对象内部。不能从外部直接访问或修改对象内部的数据和代码。
  - 继承性：是指能够直接获得已有的类的性质和特征，而不必重复定义它们。
  - 多态性：指某一论域中的一个元素存在多种解释。

- 2. 简述面向对象方法的四个要点
- 答：面向对象方法的四个要点为：
  - 1）认为客观世界是由各种对象组成的，任何事物都是对象，复杂的对象可以由比较简单的对象以某种方式组合而成。
  - 2）把所有对象都划分成各种对象类（简称为类Class），每个对象类都定义了一组数据和一组方法
  - 3）对象可以有继承关系，按照子类（或称为派生类）与父类（或称为基类）的关系，把若干个对象类组成一个层次结构的系统。
  - 4）对象彼此之间仅能通过传递消息互相联系。

- 3. 多态性有哪些体现，好处是什么？
- 面向对象程序的多态性表现在：
  - 对象类型的多态：派生类对象的类型既可以是派生类，也可以是基类，即一个对象可以属于多种类型。
  - 对象标识的多态：基类的指针或引用可以指向或引用基类对象，也可以指向或引用派生类对象，即一个对象标识符可以属于多种类型，它可以标识多种对象。
  - 消息的多态：一个可以发送到基类对象的消息，也可以发送到派生类对象，从而可能会得到不同的解释（处理）。
- 多态性的好处是：易于实现程序上层代码的复用，使得程序的扩充变得容易（只要增加底层的具体实现）。

基本知识

- 1. public, private, protected的访问控制权限
  - Public: 访问不受限制，在程序任何地方都可以访问
  - Private: 只能被本类或友元访问
  - Protected:只能被本类、派生类和友元访问
- 在派生类中，访问权限的变化

- 用C++的数组来实现一个栈类时，数组应该定义为__A__，push和pop方法应该定义为__C__。
  - A private
  - B protected
  - C public
  - D 都可以
- 下列关于成员访问权限的描述中，不正确的是_____B_____
  - A）公有数据成员和公有成员函数都可以被类对象直接处理
  - B）类的私有数据成员只能被公有成员函数及该类的任何友元类或友元函数访问
  - C）保护成员在派生类中可以被访问，而私有成员不可以
  - D）只有类或派生类的成员函数和友元类或友元函数可以访问保护成员

- 2. 构造函数和析构函数的作用
  - 构造函数：对对象进行初始化
  - 析构函数：释放对象占用的资源
- 3.　成员初始化表的作用
  - 对常量数据成员和引用数据成员进行初始化

例子

- 有下面类的说明，有错误的语句是:

```
  class X
  {
A) const int a;
B) X();
public:
C) X(int val) {a=2};
D) ~X();
};
```

答案：C不正确，应改成 **X(int val) : a(2) {};**

例子

```
class A
{ const int x,y;
   int z;
  public:
      A(): x(1),y(2)
     { z=x+y;};
      A(int i): x(1),y(x)
      { z=x+y+i};
   A(char *p): y(x),x(2)
      { z=x+y};
};
```

- 执行以下语句后，z的值？
- (1)A a1;

  a1.z=?
- (2)A a2(3);

  a2.z=?
- (3)A a3(“xyz”);

  a3.z=?

a1.z=3, a2.z=5, a3.z=4

- 4. 静态成员的作用
  - 同一类对象之间的数据共享提供了一种较好的途径
- 如何实现某类对象的计数

例：下面程序段输出结果是？

```
class A
{       static int count;
          …
    public:
        A() { count++; }
        void f(int x) {count+=x};
};

int A::count=1;
A a;
A b;
b.f(5);
cout << a.count << endl;
```

```
int A::count=1;  //count=1
A a;             // count=1+1=2
A b;             //count=2+1=3
b.f(5);          //count=3+5=8
cout << a.count << endl; //输出8
```

编程题

- 定义一个日期类，它能表示年、月、日，并提供以下操作
- (1) Date(int year, int month, int day) //构造函数
-
- (2) set(int year, int month, int day); // 设置日期

```
class Date
  {
   int year, month, day;
   public:
   Date(int y, int m, int d) //构造函数
       { year=y; month=m; day=d; }
   void set(int y, int m, int d)
       { year=y; month=m; day=d; }
}
```

思考：仿照日期类，定义一个时间类，包含时，分，秒，并写一个构造函数，和时间设置函数。

第7章 操作符重载

- 操作符重载的两种方法
  - 作为成员函数重载
  - 作为全局（友元）函数重载

编程题

- 对于前面定义的Date日期类，重载==符号，判断两个日期是否相等。

- 方法一，作为成员函数重载

```
class Date
  {
   int year, month, day;
   public:
   Date(int y, int m, int d) //构造函数
       { year=y; month=m; day=d; }
   void set(int y, int m, int d)
       { year=y; month=m; day=d; }
   bool operator == (Date d)
     {
         return (year==d.year) && (month==d.month) &&
   (day==d.day);
     }
  }
```

- 方法二，作为全局函数重载

```
class Date
  {
   int year, month, day;
   public:
   Date(int y, int m, int d) //构造函数
       { year=y; month=m; day=d; }
   void set(int y, int m, int d)
       { year=y; month=m; day=d; }
   friend bool operator == (Date d1, Date d2)
}
…
 bool operator == (Date d1, Date d2)
      {
          return (d1.year==d2.year) && (d1.month==d2.month)
  && (d1.day==d2.day);
      }
```

- 思考题：对于时间类，如何重载==操作符，实现判断两个时间是否相等？

第8章 继承-派生类

基本概念

- 1. 单继承：指派生类只能有一个直接基类
- 2. 多继承：指派生类可以有一个以上的直接基类
- 3. 静态绑定：编译时确定所调用的函数
- 4. 动态绑定：运行时根据具体的对象确定所调用的函数
- 4. 虚函数：为了实现动态绑定而声明的函数
- 6. 纯虚函数：只给出声明没有给出实现的虚函数
- 7. 抽象类：包含纯虚函数的类

基本问题

- 1. 什么是抽象类，它的作用是什么？
- 答：抽象类是指包含纯虚函数的类，它的主要作用在于为派生类提供一个基本框架和公共的对外接口，由其派生类对纯虚函数进行实现。

基本知识

- 1. 继承方式和访问控制权限

继承方式的含义

基类 / 派生类 / 继承方式	public	private	protected
public	public	不可直接访问	protected
private	private	不可直接访问	private
protected	protected	不可直接访问	protected

派生类对基类成员的访问权
原则 （1）对基类私有数据不可见 （2）以最小的权限为准

例：派生类的访问权限

```
class A
{  public:
        void a();
   protected:
        void b();
   private:
        void c();
};
```

1. 公有派生：

```
class B: public A
{   public:
          void d();
//在B类里，a()和d()是public,b()是protected，c()不可见
};
B x; x.a(); //OK
```

2. 保护派生:

```
class B: protected A
{   public:
          void d();
//在B类里，d()是public, a(),b()是protected，c()不可见
};
B x; x.a(); //error,a()是protected
```

3. 私有派生:

```
class B: private A
{   public:
          void d();
//在B类里，d()是public, a(),b()是private，c()不可见
};
B x; x.a(); //error,a()是private
```

- 不管以何种派生方式，B类都可以访问A类的a()和b()，不可访问c()

- 2. 派生类对象的初始化
- 顺序：基类构造函数>成员类构造函数>自己的构造函数

  原则：先祖先（基类），再客人（成员对象），后自己（派生类）
- 析构：相反

例子：写出以下代码段的输出

```
class A
{       int x;
   public:
        A() { x = 0; cout<< “x=” <<x; }
        A(int i) { x = i; cout<< “x=” <<x; }
};
class B: public A
{       int y;
   public:
        B() { y = 0; cout<< “y=” <<y; }
        B(int i, int j): A(i) { y = j; cout<< “y=” <<y; }
};
.....
B b1;  //执行A::A()和B::B()，b1.x等于0，b1.y等于0。打印 x=0 y=0
B b3(1,2);  //执行A::A(int)和B::B(int,int)，b3.x等于1，
                  //b3.y等于2。打印 x=1 y=2
```

- 3. 派生类的同名处理原则：出现了同名成员，则基类的成员不可见，要使用基类成员必须用类名受限
- 4. 多继承带来的二义性问题
  - 名冲突，解决：基类名受限
  - 重复继承，解决方法：使用虚基类

下面程序段输出结果是？

```
class A //基类
{  public:
        int x,y;
        int f() {x=1; return x} ;
        int g() {y=2; return y};
};
class B: public A //派生类
{       int x,z;
   public:
        int f() {y=3; return y;};
        int h() { x=f()+g()+y; return x};
};
B b;
cout<<b.h();
```

解答：根据同名处理的原则，
调用h()函数时，f()使用B的f，返回3，
g()使用A的g，返回2，y等于2，
故f()+g()+y=3+2+2=7

例子

- 设置虚基类的目的是_____。

  A）简化程序

  B）消除二义性

  C）提高运行效率

  D）减少目标代码

答案：B

例子

- 关于多继承二义性的描述中，错误的是_____。

A）一个派生类的基类中都有某个同名成员，在派生类中对这个成员的访问可能出现二义性
B）解决二义性的最常用的方法是对成员名的限定法
C）基类和派生类同时出现的同名函数，也存在二义性问题
D） 一个派生类是从两个基类派生出来的，而这两个基类又有一个共同的基类，对该基类成员进行访问时，可能出现二义性

答案：C

编程题

- 1. 编写一个时间类Time，要求包含时，分，秒，实现时间设置和打印功能。
- 2. 编写一个时间类的派生类MTime，它的功能和时间类基本相同，但是要求时间的打印以12小时制的格式显示。

```
class Time
{  protected:
       int hour,minute,seconds;
   public:
   Time(int h,int m,int s) //构造函数
   {    hour=h; minute=m; seconds=s;
   }
   void setTime(int h,int m,int s) //设置时间
   {    hour=h; minute=m; seconds=s;
   }
   void printTime() //以24小时制格式打印时间
   {  cout<<“时间是 ”<<hour<<“:”<<minute<<“:”<<second;
   }
}
class MTime: public Time //派生的时间类
{
   public:
   MTime(int h,int m,int s): Time(h,m,s){}
   void printTime() //以12小时制格式打印时间
   {    if (hour<12)
              cout<<“时间是 上午 ”<<hour<<“:”<<minute<<“:”<<second;
        else  cout<<“时间是 下午 ”<<hour-12<<“:”<<minute<<“:”<<second;
   }
}
```

第9章 类属机制——模板

基本概念

- 类属性：一个程序实体能对多种类型的数据进行操作或描述

编程题

- 用函数模板写一个max(a,b)函数，求两个任意类型的数的最大值？

```
template <class T>
T max(T a, T b)
{
  return a>b?a:b;
}
```

- max(3,5);//返回两个整数的最大值，为5，
- 也可以写成max<int>(3,5)
- max(2.5/3,7.0/8);//返回两个实数的最大值

第10章 输入输出

插入操作符“<<”的重载的例子

```
class A
{    int x,y;
   public:
      A(int i,int j) {x=i,y=j};
      ......
   friend ostream& operator << (ostream& out,
                                const A &a);
};
ostream& operator << (ostream& out, const A &a)
{  out << “x=” << a.x << “, y=” << a.y;
   return out;
}
.....
A a(2,3),b(6,7);
cout << a << endl << b;
      //输出结果是:
      // x=2, y=3
      // x=6, y=7
```

第十一章 异常处理

基本概念

- 鲁棒性/健壮性：程序在各种极端情况下能够正确运行的程度。

基本知识

- 程序的错误包括
  - 语法错误：指程序的书写不符合语言的语法规则，这类错误可由编译程序发现。
  - 逻辑错误：指程序设计不当造成程序没有完成预期的功能，这类错误通过测试发现。
  - 运行异常：指由程序运行环境问题造成的程序异常终止，如：内存空间不足、打开不存在的文件进行读操作、程序执行了除以0的指令等等。

- 处理异常的策略：
  - 1）就地处理（在发现错误的地方处理异常）
    - 调用C++标准库中的函数exit或abort终止程序执行。
    - 根据不同的情况给出不同的处理。
  - 2) 异地处理（在非异常发现地处理异常）
    - 通过函数的返回值或指针/引用类型的参数把异常情况通知函数的调用者。
    - 通过语言提供的结构化异常处理机制进行处理。

基本问题

- 1. C++异常处理机制的主要思想是什么？
- 答：C++异常处理机制的主要思想是：
- (1)把有可能造成异常的一系列操作（语句或函数调用）构成一个try语句块。
- (2)如果try语句块中的某个操作在执行中发现了异常，则通过执行一个throw语句抛掷（产生）一个异常对象。
- (3)抛掷的异常对象将由能够处理这个异常的地方通过catch语句块来捕获并处理之。

例子：

```
void f(int a, int b) //实现两个非负整数的除法
{
    if b=0 throw 0;
    if b<0 throw 1;
    if a<0 throw “a为负数”;
    cout<<a/b;
    return;
}

int test(int a, int b)
{
    try{  f(a,b) }
    catch (int i)
          {
                if (i=0)   cout<<“除数为0”;
                if (i=1)   cout<<“b为负数”;
          }
    catch (char* s)
          {
                cout<<s;
           }
}
```

写出以下语句的输出结果：

test(-2,3); //输出: a为负数

test(2,-3); //输出:b为负数

test(3,0); //输出:除数为0

test(6,3); //输出:2

第12章 MFC编程基础

基础知识

- 一个Windows应用程序由以下对象构成
- （1）窗口对象
- （2）文档对象
- （3）应用程序对象

基本问题

- 简述Windows中“文档-视”软件体系结构的好处？
  - 在这种结构中，程序所处理的数据保存在“文档”对象中，数据的显示以及与用户的交互功能则由“视”对象来完成，一个“文档”对象可以对应多个“视”对象
  - 使得数据的内部存储形式和数据的外部表示形式相互独立，对同一个文档数据可以用不同的方式进行显示和操作。

面向对象分析

基本问题

- 简述面向对象分析的基本步骤
- 答：五个步骤
  - 标识对象；
  - 标识结构；
  - 定义主题；
  - 定义属性；
  - 定义服务；

题型

- 选择题
- 名词解释
- 简答题
- 程序阅读题
- 编程题