《C++面向对象程序设计》模拟试题（8）
参考答案

一、单选题(共 20 分，每题 2 分)

C B C C D　C B B A B

二、判断正误，**对于你认为错误的论述，说明原因或举出反例。**（共 20 分，每题 2 分）

T T T F F　F F T F F

三、回答下列各题（每题 4 分，共 20 分）

1. 存在两方面问题，一是 head.h 被 mymain.cpp 重复包含两次，导致出现类型重定义错误；二是 int data=10;是变量定义，不应该放在头文件。改进方法是在 head.h 中加入包含警戒，同时将 int data=10;改为 extern int data;，并将 int data=10;移动到某个.cpp 文件;或者将 int data=10;改为 const int data=10;
2. virtual void func(FOO f) const; 或 virtual void func(const FOO f) const;
   virtual void func(const FOO & f) const;
   virtual void func(const FOO * f) const;
   上述函数原型中形参名可省略。
3. 名字空间可以解决 C++中的变量（对象）、类、函数等的命名冲突。
4. 单例模式，示例代码如下：

```
class Log {
public:
    static Log& GetInstance( ) { return aLog;}
    void Write(char * msg)     { 略 }
private:
    Log( ) { }
    Log(const Log&);
private:
    static Log aLog;
    //其它数据成员
};
```

5. 思路一：添加一个抽象鱼(Fish)，用来表示会游泳的动物，游泳(Swim)是该类的接口，鲸鱼(Whale)、鲤鱼(Carp)与鱼(Fish)是实现关系，鱼群(Shoal)与鱼(Fish)之间是聚合关系。
   思路二：从生物学考虑，添加一个哺乳动物(Mammal)，鲸鱼(Whale)与哺乳动物(Mammal) 是泛化关系，添加一个鱼(Fish)，鲤鱼(Carp)与鱼(Fish) 是泛化关系，将游泳(Swim)抽象为一个接口，鲸鱼(Whale)、鲤鱼(Carp)与游泳(Swim)是实现关系，鱼群(Shoal)与游泳(Swim)抽象接口之间是聚合关系或与鱼(Fish)之间是聚合关系。

四、

```
Bus::Bus( int maxCapacity):capacity(maxCapacity),passagers(0),income(0) { }
void Bus::ToStation(int nUp,int nDown) {
    Down(nDown);
    Up(nUp);
}
int Bus::GetIncome( ) const    {
    return income;
}
void Bus::Up(int num) {
    if ( num<=0 ) return;
    int upNum = (passagers+num >capacity ? capacity-passagers : num);
    passagers += upNum;
    income += 2* upNum;
}
void Bus::Down(int num) {
    int downNum = (num< 0 || num>=passagers ?    passagers: num);
    passagers -= dwonNum;
}
```

五、分别写出下面两个程序的运行结果（共 10 分）

1. 0 1 3 4 4 9 8 7 7
2. Derived::vf( )
   Base::vg( )
   Derived::vf( )
   Base::nvh( )
   Derived::vf( )
   Base::nvh( )
   Derived::vf( )

六、1)

```
class Door {
public:
   virtual ~Door( );
   virtual void Lock( );
   virtual void UnLock( );
};
class Alarm{ public : void Beep( ){ /*略*/ } };
class AlarmDoor: public Door {
public:
```

```
    AlarmDoor(Alarm & param):ra(param) { }
    virtual ~AlarmDoor( );
    virtual void Beep( ) { ra.Beep( ); }
  private:
    Alarm & ra;
  };
```

2)能适应新要求。

七、最好先定义一个抽象基类：

```
class Vector {
public:
    Vector( double values[N] )      {
        for( int i = 0; i < N; ++i ) {
            items[i] = values[i];
        }
    }
    virtual ~Vector() {}
    virtual double NormalValue() const = 0;
    double Item( int index ) const
        {   return items[index];   }
    Vector& Standard( )   {
        double nv = NormalValue( );
        for( int i = 0; i < N; ++i ) {
            items[i] /= nv;
        }
        return *this;
    }
protected:
    double items[N];
};
```

派生子类可以两种思路：

1. (最直接)从基类派生子类 VectorOrder1,VectorOrder2,VectorOrderInf，需要 K 阶时，派生子类 VectorOrderK 即可。在每个子类中给出 NormalValue 的不同实现。
2. (或者)从基类派生子类 VectorOrderK,VectorOrderInf

```
  class VectorOrderK :public Vector {
  public:
      VectorOrderK(double values[N],int k) :Vector(values),order(k) {   }
      virtual double NormalValue( ) const {
          double sum = 0.0;
```

```
            for( int i = 0; i < N; ++i ) {
                sum += pow( fabs( items[i] ), order );
            }
            return pow( sum / N, 1.0 / order );
        }
protected:
        int order;
};

class VectorOrderInf:public Vector {
public:
      VectorOrderInf (double values[N]) :Vector(values),order(k) {    }
      virtual double NormalValue( ) const {
            double result = 0.0;
            for( int i = 0; i < N; ++i )
                result = max( result, fabs( items[i] ) );
                return result;
      }
};
```

（全卷完）