《C++面向对象程序设计》模拟试题（9）

参考答案

一、单选题

D, C, A, B, B, A

二、判断正误

1. 错误。double 占 8 字节，float 占 4 字节，转换是不安全的，不会发生隐式转换，而是语法错误。
2. 正确。
3. 错误。是大于或等于。
4. 错误。不一定。如用 explicit 修饰的构造函数，或私有的，或只声明但没有实现的。
5. 错误。不一定。如父类中有虚函数，则在创建子类对象时，仍会生成虚拟表。
6. 错误。可以。如 try { f( ); } catch(…) {}, 如果 f( )中产生异常，但 f( )中没有使用 try-catch 结构捕获该异常，则该异常会被 catch(…)捕获。
7. 错误。抽象类表示一种泛化关系，其接口函数应在子类中存在,以保证接口的一致性，包括创建对象和释放对象的接口，因此构造函数和拷贝构造函数应为 public。
8. 错误。非静态的非虚函数可访问虚函数。
9. 错误。析构函数不能重载。
10. 正确。拷贝构造时，会自动调用子对象所在类的拷贝构造函数，而子对象的拷贝构造过程是可能产生内存泄漏的。

三、回答下列各题（每题 4 分，共 28 分）

1. 问题或有待改进的地方
   a) 最好在 fun.h 中加入包含警戒
   b) fun.cpp 中应#include”fun.h”
   c) main.cpp 中应#include”fun.h”
   d) fun 函数的参数，在类定义和外联实现中均给出了缺省值，应去掉实现中的缺省值。
2. 区别：

   void fun(string); 以传值的方式传递一个字符串参数，参数匹配过程中，会调用 string 类的拷贝构造函数，构造一个新的字符串供 fun 函数存取。使用时，实参可以是任意一个 string 对象。

   void fun(string &);以传引用的方式传递一个字符串参数，参数匹配过程中，不用调用 string 类的拷贝构造函数，而是直接传递一个引用，在 fun 中对参数的任何修改，影响原始的实参。使用时，要求实参必须是变量型。

   void fun(const string&);以传引用的方式传递一个字符串参数，参数匹配过程中，不用调用

string 类的拷贝构造函数，而是直接传递一个引用。同时不允许在 fun 中对参数进行任何修改操作。使用时，实参可以使变量型，也可以使 const 型。

3. 不可以。若可以，就会产生诸如构造了一个对象，却调用了多次析构函数的情形。所以禁止显式调用析构函数。

4. 类 A 的设计

```
class A {
        public:
                static A * CreateA(int n) { return new A(n); }
        private:
              A(int n):val(n) { }
        private:    int val;
};
```

5. 某订单管理系统中，Customer 表示客户类，Product 表示产品类，ProductList 表示产品列表类，Order 表示产品订单类，OrderItem 表示产品订单中的条目类，OrderList 表示订单列表类，请分析并给出各类之间的关系。

Customer – OrderList　聚集关系或关联关系或依赖关系
OrderList – Order　聚集关系
Order– ProductList　聚集关系
ProductList-Product　聚集关系

6. 必须使用初始化列表
   a) 类中含有 const 修饰的数据成员(常数据成员)，如 const int num；
   b) 类中含有引用型数据成员，如 int& a；
   c) 类中含有对象成员，但其类型中，没有无参构造函数，必须在初始化列表中指明相应的有参构造函数。
   d) 基类中缺少无参构造函数，必须在派生的初始化列表中指明基类的有参构造函数。

7. 使得类 B 具有深拷贝和深赋值的能力：

```
B::B(const B& rhs)
    :A(rhs)
{
     p = new int(*rhs.p);
}
```

```
B& B::operator=(const B& rhs) {
      if ( this!=&rhs) {
           A::operator=(rhs);
           delete p;
           p= new int(*rhs.p);
      }
      return *this;
 }
```

四、 Triangle 类

```
class Triangle: public Shape {
public:
```

```
    Triangle( ):pXtriangle(new XTriangle) { }
    virtual ~Triangle() { delete pXtriangle; }
    virtual void Draw( ) { pXTriangle->DrawIt( ); }
private:
    XTriangle * pXtriangle;
};
```

五、实现 main 函数，计算并输出分别按线路 1 和线路 2 旅游的花费。

```
#include <iostream>
using namespace std;
int main( ) {
    City   a(120),b(80),c(70),d(100);
    int sum1 = a.Spend(7) + b.Spend(6) + c.Spend(5) + d.Spend(4);
    int sum2 = a.Spend(4) + b.Spend(5) + c.Spend(6) + d.Spend(7);
    cout<< “路线 1=”<<sum1<<endl;
    cout<< “路线 2=”<<sum2<<endl;
    return 0;
}
```

六、导弹飞行仿真系统

```
    class Missile{
     public:
          Missile(LanchMode& lm, FlyBehavior& fm):lmode(lm),fmode(fm) {}
          virtual ~Missile() {}
          virtual void Lanch( )    { lmode.Lanch( ) ;}
          virtual void Fly( )      { fmode.Fly( ) ;     }
     protected:
          LanchMode& lmode;
          FlyBehavior& fmode;
};
```

```
class LanchMode {
pulbic:
          virtual ~LanchMode() { }
virtual void Lanch( ) = 0;
    };
```

```
class FlyBehavior {
      public:
          virtual ~ FlyBehavior() {}
          virtual void Fly() =0;
};
```

```
    class SeaLanch:public LanchMode {
        public:
            virtual ~SeaLanch( ) {}
            virtual void Lanch( ) {
                cout<<”海基发射”;
            }
};
```

```
    class   SubSonicFly:public FlyBehavior {
    public:
            virtual ~ SubSonicFly( ) {}
            virtual void Fly( ) {
                cout<<”亚音速飞行”;
            }
};
```

```
class LandLanch:public LanchMode {
    public:
        virtual ~LandLanch( ) {}
        virtual void Lanch( ) {
            cout<<"陆基发射";
        }
};
```

```
class  SuperSonicFly:public FlyBehavior {
public:
        virtual ~ SuperSonicFly ( ) {}
        virtual void Fly( ) {
            cout<<"超音速飞行";
        }
};
```

七、 填写代码。

1) data = new int[len]
2) delete[] data
3) sum += obj.CallBackFunc(data[i])
4) virtual int CallBackFunc(int val) = 0
5) srv.Total(*this)
6) return val*val;
7) cout<<"平方和="<<srv.Total(*this)<<endl;

```
virtual int CallBackFunc(int val){
    cout<<val<<"  ";
    return val*val;
}
```

```
void RequestB(Server& srv) {
    int n = srv.Total(*this);
    cout<<"的立方和="<<n<<endl;
}
```

八、类 MyPic：

```
class DeviceWithOption {
public:
virtual ~ DeviceWithOption ( ) {}
virtual void Process(MyPic& pic) = 0;
};
```

```
class MyPic {
public:
void  SendTo(DeviceWithOption&  dwo)
    {   dwo.Process(*this);   }
};
```

```
class PrinterWithOption
            :public DeviceWithOption {
public:
    virtual ~ PrinterWithOption() {}
    PrinterWithOption(Printer& printer)
        :prt(printer) { }
    void SetXXX(int) {}
    virtual void Process(MyPic& pic)
      {   //略  }
private:
        Printer& prt;
```

BitmapWithOption 类略。	int marginLeft; int marginTop; int marginRight; int marginBottom; int alignKind; };

2.增加打印机打印选项。可以从 PrinterWithOption 派生新的子类。

增加位图输出选项。可以从 BitmapWithOption 派生新的子类。

3.增加新的设备。可以从 DeviceWithOption 派生新的子类。

（全卷完）