《C++面向对象程序设计》模拟试题（10）

参考答案

一、单选题

BABCD　DDACC

二、判断正误

1. 错 应是 const A* const
2. 对
3. 错 B 可以提供类型转换函数，或者 A 提供类型转换构造函数
4. 错 析构函数只有一个无参的
5. 错 可以是类 A 或者 A 的子类或者其它提供了相应的类型转换函数的类型，而且是否为 const 均可。
6. 对
7. 错 名字空间也可以嵌套
8. 错 静态数据成员的初始化时机和顺序，与实例化对象的时机和顺序无关。
9. 错 非静态成员函数可以直接访问本类的静态成员.
10. 错 非常对象也可以

三、按要求回答问题

1. ***a.h*** 和 ***b.h*** 中存在多处不妥

   a.h 中：

   1. 缺包含警戒
   2. Out 应带 const 修饰。
   3. Square 函数返回 A 类对象，但 A 是抽象类，不能实例化。
   4. 缺少虚的析构函数

   b.h

   1.缺包含警戒

   2. #include <iostream>应放#include “a.h” 前
   3. 类 B 缺自定义构造函数,无法创建对象
   4. 使用 using namespace std; 或 将 cout 改为 std::cout
   5. （不做要求）定义虚的析构函数
   6. Out 也应带 const 修饰
   7. 不能访问基类私有成员 val

2.

   对于普通内置类型（如整数等），表达式 （a=b）= c 是合法的。因此，对于自定义类型表达式 （a=b）= c 也应该是合法的，保证用户开发程序时的一致性。但是，

   若按值返回，（a=b）= c，最终 a==b，而不是 a==c，违反通常的赋值运算含义。

   若返回 void，（a=b）= c，语法上就不合法，编译通不过。

按引用返回，可以解决上述问题。

3.

为所有图元设计一个共同的基类 Shape，在 Shape 中提供一个虚函数:
virtual bool PickMe(const MousePos & pos) ; 每个图元类根据各自的判断方法重新实现这个虚函数。

4. 填空
   (1) Shape*
   (2) Shape
   (3) Shape
   (4) Container

5.

Member – GroupMember	(公有)继承/派生/泛化/一般化
Group-GroupMember	聚集
GroupList-Group	聚集
Member – Friend	(公有)继承/派生/泛化/一般化
FriendList – Friend	聚集

6. 创建 Configure 的唯一实例(用指针或静态对象均可)

```
class Configure {
public:
      static Configure& GetInstance(    ) {
            static Configure cfg;
            return cfg;
      }
private:
      Configure( ) {}
      Configure(const Configure&);
      Configure& operator=(const Configure&);
public:
     int GetConfigureData( ) { return data;} //获取配置信息
     void SetConfigureDate( int m_data )   {data=m_data;}   //设置配置信息
private:
      int data;       //配置信息
};
```

四、阅读代码，回答下列各问题

1. (1)、(2)、(3)处代码, static, p->Item(i); , n*n,
2. (4)处代码: Item(n-1)+ Item(n-2)
3. 定义类 T。

```
class T : public Series {
public :
    virtual ~T( ) { }      //不做要求
    virtual int Item( int n ) const {
          int count=0;
          for(int i=0; i<n-1; ++i)    {
             if( a[i] < a[n-1] )    ++count;
          }
          return       count;
    }
};
```

```
int main( ) {
          int      a[ ] = {1,-3,4,2};
          T      t;
          Series::Show(&t,4);
          return 0;
}
```

五、设计方案：

对测算产量的全局函数抽象到 MethodA 类中(是否是抽象类，随意)，在其派生类中 override 不同的 Amount 测算方法；

对测算价格的全局函数抽象到 MethodB 类中(是否是抽象类，随意)，在其派生类中 override 不同的 Price 测算方法；

MethodA 和 MethodB 是否有共同父类，不做要求。

改写 Farm 类中 ForecastXXXX 函数；

将进一步处理 Result 的过程，抽象到独立的类 ResultProcess 中，改写代码。

使用依赖关系或关联关系，不做限定。

```
class MethodA {
public :
  virtual ~MethodA( ) {}
  virtual void Amount(Data data[ ],int len)=0;
};

class MethodA01 : public MethodA    {
public :
virtual void Amount (Data data[ ],int len)
     {/*略*/}
};
```

```
class MethodB {
public :
virtual ~MethodB( ) {}
virtual void Price(Data data[ ],int len)=0;
};

class MethodB01 : public MethodB    {
public :
virtual void Price(Data data[ ],int len)
```

```
class Farm {
public:    static const int N = 20;
float ForecastAmount(MethodA & ma) {
        float amount = ma.Amount (history,N);
        /*  略  */
        return amount;
}
float ForecastPrice(MethodB& mb) { //测算价格
     float price = mb. Price (history,N);
      /*  略  */
      return price;
}
float Income(MethodA& ma,    MethodB& mb,
              ResultProcess& mc;) {
      //测算收入
      float result
      =ForecastAmount (ma)*ForecastPrice(mb);
       //进一步分析处理 result,略
       result = mc.Process (result);
       return result;
```

```
    {/*略*/}
};
```

```
}

void    SetData( ) { /*  略  */ }
private:
        Data history[N];
};
```

```
class ResultProcess{
public :
virtual ~ ResultProcess ( ) {}
virtual float Process ( float val)
   { /*  进一步分析处理 result,略  */     }
};
```

六、某饮料店

本题的设计方案多样，不做限定，只要求满足题目输出要求。

Drink 和 Condiment，可以是聚集关系、关联关系，也可以抽象出一个公共基类。

聚集关系时，可以使用定长数组、动态数组、向量、有序集合、指针等。

```
const int N=10;
class Condiment {
public:
        Condiment(int c,string str)
          :cost(c),name(str) {}
        virtual ~Condiment( ) { }
        int GetCost( ) const { return cost; }
        string GetName() const { return name;}
private:
        int    cost;
        string name;
};

class Strawberry:public Condiment {
public:
        Strawberry():Condiment(2,"草莓") {}
};

class Pudding:public Condiment {
public:
```

```
class Drink {
public:
        Drink(int theCost,string str)
           :cost(theCost),index(0),name(str)
        {    for(int i=0;i<N;++i) others[i] = NULL; }
        virtual ~Drink() {}
        void Add(Condiment& condiment) {
                if(index<N)
                        others[index++] = &condiment;
        }
        int SumCost() const {
                int sum = cost;
                for(int i=0;i<N;++i)
                        if(others[i])
                           sum += others[i]->GetCost();
                return sum;
        }
        void Show( ) const {
                for(int i=0;i<N;++i)
```

```
    Pudding():Condiment(3,"布丁") {}
};
```

```
        if(others[i])
            cout<<others[i]->GetName()<<" ";
        cout<<name<<" "<<SumCost()<<endl;
    }
protected:
    int    cost;
    int    index;
    string name;
    Condiment *    others[N];
};
class Milk:public Drink {
public:
    Milk():Drink(8,"牛奶") {}
};

class MilkShake:public Drink {
public:
    MilkShake():Drink(9,"奶昔") {}
};

class Coffee:public Drink {
public:
    Coffee():Drink(10,"咖啡") {}
};
```

七、开发视频播放系统

设计方案：

将 RMVB_Player, MPG_Player 和 AVI_Player 抽象出一个基类 VideoPlayer，其中有函数 Play，参数为文件名，依次完成解码和显示的播放过程。解码函数应为虚函数。

在 VideoPlayer 类中，添加数据成员 OS * pOS，并将解析后的标准视频流，最为实参传给 Display 函数。

将不同操作系统抽象出一个基类 OS，其中含有抽象的虚函数 Display (StandardVideo& video);,在 OS 的子类 WinOS 中给出具体实现。

共 17 个类，VideoPlayer 类、10 个 VideoPlayer 类的子类，class OS，5 个 class OS 的子类。

```
class VideoPlayer {
public:
    VideoPlayer(OS* p) : pOS(p){}
    virtual ~VideoPlayer( ) { delete pOS; }
    void Play (string fileName) {
        StandardVideoStream* pStream
```

```
class OS {
public:
    virtual ~OS( ) {}
    virtual void Display(StandardVideoStream*
pStream)=0;
};
```

```
                = Decode (fileName);
        pOS->Display(pStream);
        delete pStream;
    }
virtual StandardVideoStream* Decode (string fileName)=0;
protected:
    OS * pOS;
};

class RMVB_Player : public VideoPlayer {
public:
    RMVB_Player(OS* pOS)
        :VideoPlayer(pOS){}
    virtual StandardVideoStream* Decode(string file_name)
    {/*略*/};
};
class MPG_Player : public VideoPlayer {
//类似 RMVB_Player, 略

};
```

```
class LinuxOS : public OS {
public:
    virtual void Display(StandardVideoStream* pStream) {/*略*/}
};
class WinOS : public OS  {
public:
  virtual void Display(StandardVideoStream* pStream){ /*略*/}

};
```

（全卷完）