2011 年 C++

一、选择题（2*20）

1.封装是将数据与_____进行有机结合?

2. for(s;;s1)等价于下面那个句子？（ ）

A for(s;0;s1)　　B for(s;1;s1)　　C for(s;s;s1)　　D for(s;s1;s1)

3.输入输出流中的 cerr 是？（ ）

A 对象　　B 类　　//p483

4. ifstream 默认的方式是（ ）

A ios：：in　　B ios：：out　　C ios：：in||out

5.运算符重载有两种形式：友元和成员函数，下面说法正确的是（ ）

A 两者都含有 this 指针

B 友元比成员函数声明时多一个关键字

C 两者的参数不同，但是实现形式一样

6. 下列函数中含有 this 指针的是（ ）

A 构造函数　　B 静态成员函数　　C 成员函数　　D 析构函数

7.下列说法不正确的是（ ）

A 抽象类不能作为基类

B 抽象类没有实现部分//存疑

C 抽象类有虚函数

D 抽象类不能定义对象

8 .A 为基类，B 为派生类，a 是 A 的对象，b 是 B 的对象，int *pa=&a; *pb=&b，则下列式子正确的是（ ）

A a=b　　B b=a;　　C pa=pb;　　D pb=pa;

9 template<class T>，T 是（ ）

A 自定义数据类型　　B 类型说明符

二、填空（4*6）

1 .A 为一个类名，a 是 A 的对象，则 delete a 时将自动调用_____

2. 进行运算符重载时，如果是类的的成员函数，c1+c2 被编译器解释为_____；如果是友元函数，c1+c2 被编译器解释为_____

3 .函数 f（）需要访问类 A 的私有成员，则需要在类 A 中对函数 f（）的声明_____

2013 年 C++

一、选择题（10 * 3）

二、名词解释（5 * 6）

1.函数对象

2.类模板

3.封装

4.动态绑定

5.容器

2014 年 C++

一、选择题（1*2 分）

1. 静态成员函数没有（ ）

A. 返回值　　B. this 指针　　C. 指针参数　　D. 返回类型

2. 假定 AB 为一个类，则执行“AB a(2), b［3］,*p［4］;”语句时调用该类构造函数的次数（指针不返回，a 对象初始化调用一次，数组每一个元素一次，指针数组不调用，因为暂时不指向对象）为（ ）

A. 3　　B. 4　　C. 5　　D. 9

3. 有关多态性说法不正确的是（ ）

A. C++语言的多态性分为编译时的多态性和运行时的多态性

B. 编译时的多态性可通过函数重载实现

C. 运行时的多态性可通过模板和虚函数实现

D. 实现运行时多态性的机制称为动态多态性

4. 设有函数模板

```
template <class Q>
   Q Sum(Q x,Q y)
{ return (x)+(y);   }
```


则下列语句中对该函数模板错误的使用是（ ）

A. Sum(10,2);

B. Sum(5.0,6.7)；

C. Sum(15.2f,16.0f);

D. Sum(“AB”,”CD”);

5. 所谓多态性是指 （ ）

A. 不同的对象调用不同名称的函数

B. 不同的对象调用相同名称的函数

C. 一个对象调用不同名称的函数

D. 一个对象调用不同名称的对象

6.下列不是描述类的成员函数的是（ ）

A. 构造函数　　B. 析构函数　　C. 友元函数　　D. 拷贝构造函数

7.. 在私有继承的情况下，基类成员在派生类中的访问权限（ ）

A. 受限制　　B. 保持不变　　C. 受保护　　D. 不受保护

8.. 已知：p 是一个指向类 A 数据成员 m 的指针，A1 是类 A 的一个对象。如果要给 m 赋值为 5，正确的是（ ）

A. A1.p=5;　　B. A1->p=5;　　C. A1.*p=5;　　D. *A1.p=5;

9. 如果采用动态多态性，要调用虚函数的是（ ）

A. 基类对象指针　　B. 对象名　　C. 基类对象　　D. 派生类名

10. 类 B 是类 A 的公有派生类，类 A 和类 B 中都定义了虚函数 func(),p 是一个指向类 A 对象的指针，则 p->A::func()将（ ） ///指定调用

A. 调用类 A 中的函数 func()

B. 调用类 B 中的函数 func()

C. 根据 p 所指的对象类型而确定调用类 A 中或类 B 中的函数 func()

D. 既调用类 A 中函数，也调用类 B 中的函数

二、填空题（1*2 分）

1.C++语言中如果调用函数时，需要改变实参或者返回多个值，应该采取_____方式。

2.面向对象的四个基本特性是多态性、继承性、和封装性、_____。

3. 派生类的成员一般分为两部分，一部分是_____，另一部分是自己定义的新成员。

4.C++支持的两种多态性分别是_____多态性和运行多态性。

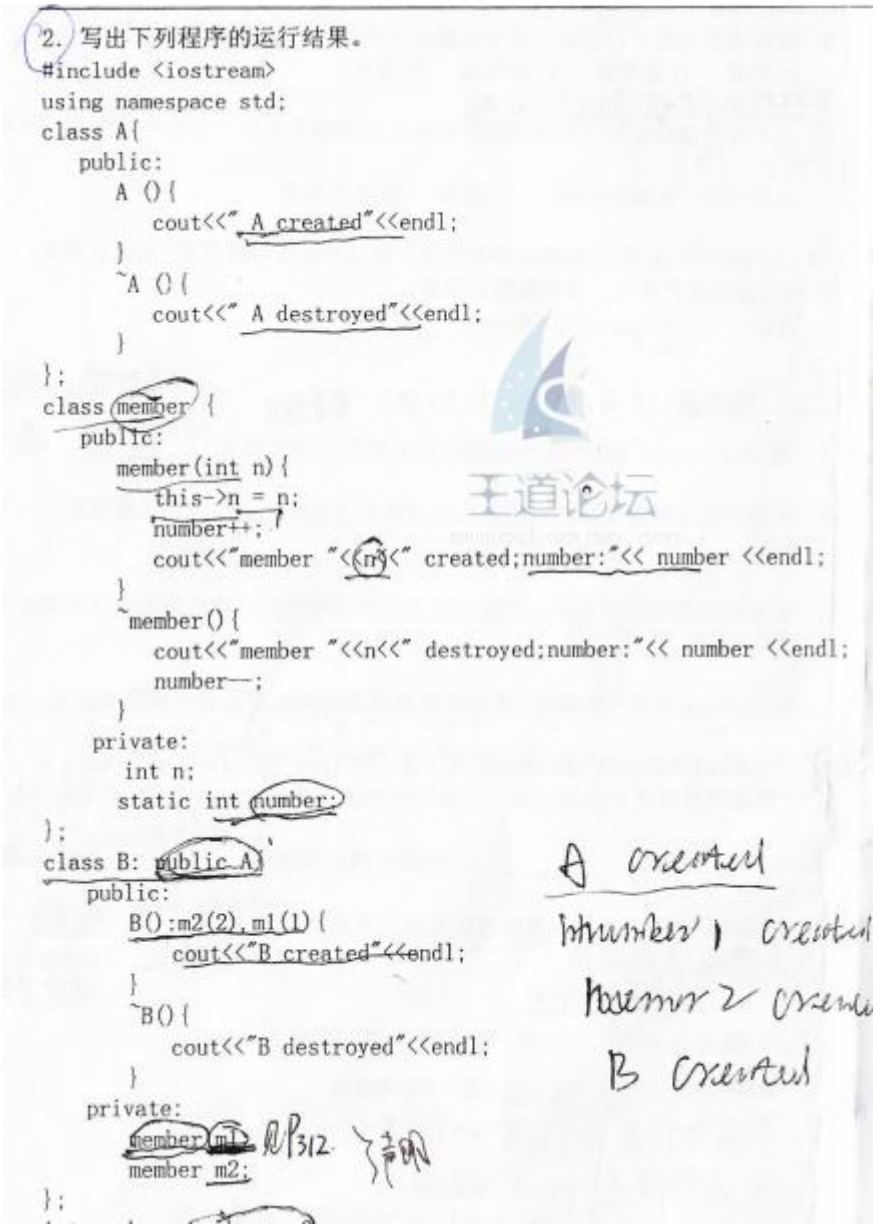
5. 定义类的动态对象数组时，系统只能够自动调用该类的_____构造函数对其进行初始化。

三、问答题

1.如果没有自定义拷贝构造函数，系统会自动调用默认拷贝构造函数，会出现什么现象？

2.友元函数、全局函数、成员函数的不同

3.写出下面程序运行结果。



2. 写出下列程序的运行结果。

```
#include <iostream>
using namespace std;
class A{
   public:
      A (){
         cout<<" A created"<<endl;
      }
      ~A (){
         cout<<" A destroyed"<<endl;
      }
};
class member {
   public:
      member(int n){
         this->n = n;
         number++;
         cout<<"member "<<n<<" created;number:"<< number <<endl;
      }
      ~member(){
         cout<<"member "<<n<<" destroyed;number:"<< number <<endl;
         number--;
      }
    private:
      int n;
      static int number;
};
class B: public A{
   public:
      B():m2(2),m1(1){
         cout<<"B created"<<endl;
      }
      ~B(){
         cout<<"B destroyed"<<endl;
      }
   private:
      member m1;
      member m2;
};
```

A created
member 1 created
member 2 created
B created

2015 年 C++

一、选择

二、判断

三、简答

1. 类与对象的关系

2. 静态绑定与动态绑定的相同点不同点

3. 分析程序，写出结果，涉及到继承

4. 分析程序写结果，只不过这道题涉及到 try catch 异常

2016 年 C++

一、 选择（20′）

二、填空（20′）

三、简答（40′）

1.对象和类的联系和区别

2.都有什么方法可是实现代码重用，并简单举例说明。

3.分析程序结果：try catch 抛异常

4.构造析构函数的调用顺序

2017 年 C++

二、填空（35’）

1.计算面积的 shape 类

2.一个有关字符数组的

三、读程序分析结果

1.虚函数继承时的多态，给了三角形、矩形、圆的面积函数，要写出对同一虚函数调用时的不同行为。

2.分析程序中某个函数的作用，写结果，好像是一个计数的 count 函数

2018 年 C++

一、选择（20′）

1.编写 c++程序的经过：_____

2.7.0/2.0 和 7.0/2 结果类型是否相同_____

3.对象是 c 语言的结构体变量是否正确？_____

4.静态成员函数没有什么指针？_____

5..不能继承的的构造函数是：_____、_____

6.类有一个指针成员*p,应该如何调用赋值？_____

7.抽象函数的定义：_____

8.字符数组实参传形参是_____传递。

9.何为多态性？_____

二、填空（20′）

1.内联函数的关键字：_____

2.继承的访问权限：_____

3.构造函数的执行次数:_____

4.运算符重载形参的个数?_____

5.对象是类?_____

6.静态成员函数，友元函数是成员函数?_____

7.类默认的访问权限是：_____

8.面对对象的四大特性：_____

三、简答（40′）

第二部分　C++真题选填答案

一、2011 年

(一)选择题

1.封装是将数据和功能(函数)结合，填功能。

2.无穷循环，选 B。

3.cout 和 cerr、clog 都是预定好输出流对象。cout：标准输出流对象；cerr：标准错误输出流，无缓冲；clog：同 cerr，但有缓冲。选 A。

4.ifstream 流默认传入方式是：ios::in。

5. A 友元函数没有 this 指针 B 对 C 参数不同，实现方式不一样。故 B。

6. 静态成员函数独立于类不会有 this 指针。析构函数和友元函数也不会 this 指针。？？？

7. A 错误，抽象类就是用来继承的。

B 含有虚函数的就是抽象类，抽象类其它函数可以有实现部分。存疑

D 抽象类不能有对象但可以有指针。正确

故选 A

8. 基类指针或引用可指向派生类对象，反之不成立；同时基类和派生类对象之间不能直接赋值。故 C 正确。

9. 就像平时定义一个类一样，Class A，A 是自定义类型。故 A 正确。

(二) 填空题

1.a 的析构函数。删除对象调用析构函数，注意不是 A，应该是类对象。

2.成员函数解释为：c1.oprator+(c2) ;友元函数解释为：operator+(c1,c2);

3.friend void f( );

三、2013 年

(一)选择题

(二)名词解释

1.**函数对象**。如果一个类将()运算符重载为成员函数，这个类就称为函数对象类，这个类的对象就是函数对象。

2.**类模板**。允许用户为类定义个一种模式，使得类中的某些数据成员、默认成员函数的参数，某些成员函数的返回值，能够取任意类型(包括系统预定义的和用户自定义的)

3.**封装**。封装是面向对象方法的一个重要原则，就是把对象的属性和服务结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节同时以特定权限访问类成员和方法。

4.**动态绑定**。动态绑定是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。C++中，通过基类的引用或指针调用虚函数时，发生动态绑定。

5.**容器**。用于存放数据的类模板。可变长数组、链表、平衡二叉树等数据结构在 STL 中都被实现为容器。

四、2014 年

(二)填空题

1.传地址或引用。传地址或引用改变实参好理解，怎么返回多个值？也许是说传数组吧？

2.四大特性：多态、继承、封装、抽象。

3.继承基类的成员。

4.同上，多态分为静态多态性和动态多态性。

5.动态对象数组如：point* p = new point[5]系统只能调无参构造函数（也没法传参数）

(一)选择题

1.常考，静态成员析构函数无 this 指针。故 B

2.创建对象 a，调用带参构造函数一次；创建对象数组 b[3]调用无参数构造函数一次；创建指向对象指针不会调用构造函数。故 4 次，选 B。

3.多态性分为：

1.静态多态又称编译时多态。包括函数重载、模板（注意不是动态）、运算符重载

2.动态多态又称运行时多态。主要通过 继承+虚函数 实现。故此题选 C

4.字符串不能直接相加，D。

5.按题中意思应该是说动态多态性，基类和派生类都有相同名称函数。根据基类指针指向的不同对象调用相同函数，故 B。

6.友元函数不是成员函数，故 C。

7.私有继承，派生类不能访问基类所有成员；公开继承，不改变。故 A

8.多次考，记住如何访问类指针成员**赋值**：对象名.*指针 = 5。故 C

9.要实现多态性，只能是基类对象指针或引用，不能是基类或者派生类对象，故 A。

10.p 是基类 A 的指针，由于有虚函数虽然没有静态绑定基类 func，但也指向动态对象。这里及时不加类名限定 A::也是调用 A 中的函数 func。故 A。

(三)问答题

1.**没有自定义构造函数，会发生浅复制**。如果类中有需动态分配空间成员如字符指针可能会发生以下后果：复制后有两个或多个指向同一内存空间的成员，任何一个修改都会导致他们指向的内容被改变；虚构函数释放会导致对同一空间释放多次，并可能导致其他指针成为悬挂指针。

2.**友元函数、全局函数、成员函数区别**。(1)成员函数是在类中定义的，可以访问类的成员和调用类中其他函数、全局函数。(2)友元函数是另外一个类的成员函数，如果其是公有，本类也只能通过对象名调用。(3)不在类中定义的都是全局函数。

五、2015 年

(一)选择题

(二)填空题

(三)简答题

1.**类和对象的关系**。类是定义同一类所有对象的变量和方法的蓝图或原型，类是抽象的。对象是具体的的，对象是类的一个实例，对象继承类的方法和属性。

2.静态绑定和动态绑定的相同和不同点。静态绑定是编译阶段已经决定,即编译阶段已知道会调用哪个函数。动态绑定是运行时确定的，通常由虚函数实现运行时绑定,又称动态联编.

3.

4.

六、2016 年

(三) 简答题

1.**对象和类的区别与联系**。同 2015 年第一题简答

2.**实现代码重用方法**。1.模板，将不同的对象的类型作为模板参数。如模板实现指定类型的数组类。2.使用宏，将不同类型的的对象作为宏参数。3.对不同的对象做一次抽象封装，提取公共的基类，在基类中抽象虚函数。如实现 circle 类、rectangle 类继承了 shape 类，shape 类就是把他们公共的函数方法等提起出来作为基类。

3.

4.**析构函数的调用顺序**。析构函数的调用顺序恰和构造函数的调用顺序相反：1.先执行析构函数的函数体 2.然后对派生类新增的类类型的成员对象进行清理（构造函数是新增的成员对象初始化）3.最后调用基类析构函数对所有基类继承来的成员进行清理（构造函数是执行基类构造函数对基类成员初始化，按被继承的顺序）。

八、2018 年

(一) 选择题

1. 编写 c++程序经过：编写、编译、链接、运行
2. 相同，运算符左右操作不同会发生类型提升，变成同一类型。
3. 对象时 C 语言的类类型变量？
4. 静态成员没有 this 指针。
5. 不能继承的函数有：构造函数、析构函数、友元函数
6. 对象名.*p = 5
7. 抽象函数即指：这是其他高级语言如 C#的说法。这里应该说的是 C++中纯虚函数。

   定义为： virtual void func() = 0;
8. 数组都是地址（引用）传递。
9. 同前。不同对象调用同名函数即是多态性。

(二) 填空题

1. inline
2. private 继承的访问权限：见后名词解释汇总
3. 什么。。题目不全。
4. 重载成员函数形参个数 = 操作符目数-1 ；重载非成员函数形参个数 = 操作符目数。
5. 对象是类的实例（类是对象的模板）
6. 静态和友元函数都不是成员函数。
7. private
8. 抽象、封装、多态、继承。

九、C++名词解释补充

1. 继承

继承是让某个类型的对象获得另一个类型的对象的特征。例如：C++中子类对父类的继承，子类具有父类的特性，同时还可以拥有自己的新特性。

2. 多态

多态是指不同类型的对象接收相同的消息时产生不同的行为。多态机制使具有不同内部结构的对象可以共享相同的外部接口，通过这种方式减小代码的复杂度。例如函数的重载。

3. 什么是 this 指针？为什么要用 this 指针？

this 指针是类中的一个特殊指针，当类实例化时，this 指针指向对象自己；而在类的声明时，指向类本身。通过它不仅可以提升成员函数操作的效率，而且能简化运算符重载代码。

4. 叙述公有、私有、保护成员在类中、类外和对象中的访问权限。

(1) 对于 public 成员来说，他们是公有的，可以在类外和对象中访问。

(2) 对于 private 成员来说，他们是私有的，不能在类外和对象中访问，数据成员只能由类中的函数使用，成员函数只允许在类中调用。

(3)对于 protected 成员来说，他们是受保护的，具有半公开性质，可以在类中与子类中访问。

5.构造函数和析构函数的作用是什么？

构造函数的功能：是在创建对象时，给数据成员赋初值，即给对象初始化。

析构函数的功能：是释放一个对象，在对象删除前，用来做一些内存释放等清理工作。

6.什么是类的继承和派生？

继承是指一个事物可以继承其父辈全部或部分的特性，同时本身还有自己的特性。当一个新类从一个已定义的类中派生后，新类不仅继承了原有类的属性和方法，并且还拥有自己新的属性和方法，称为类的继承和派生。

7. 派生类 public 继承方式有那些特点？

（1）在派生类中，基类的公有成员、保护成员和私有成员的访问属性保持不变。在派生类中，基类的私有成员是无法访问的，虽然基类的私有成员被派生类继承了。

（2）派生类对象只能访问派生类和基类的公有成员。

8. 派生类 protected 继承方式有那些特点？

（1）在派生类中，基类的公有成员、保护成员的访问属性都变成了保护的。

（2）在保护继承方式下，派生类中仍可以访问基类的公有成员和**保护成员（继承前的访问属性）**，但基类的私有成员是无法访问的。

（3）派生类对象只能访问派生类的公有成员。

9.派生类 private 继承方式有那些特点?

（1）在派生类中，基类的公有成员、保护成员和私有成员的访问属性都将变成私有的。

（2）私有继承方式下，派生类中仍可以访问基类的公有成员和**保护成员**，但基类的私有成员是无法访问的。

（3）派生类对象只能访问派生类的公有成员。

10. 在定义派生类的过程中，如何对基类的数据成员进行初始化?

通过调用基类的构造函数来设定基类的数据成员的初值。

格式为 <派生类名>（形参表）：基类 1（参数表），基类 2（参数表）...对象成员 1（参数表），对象成员 2（参数表）...{}

11. 什么是虚基类？它的作用是什么?

虚基类是指在派生类中指定的基类是虚继承的方式。

使用虚基类的目的是在多重派生中使用共有基类时，在派生类中只有一个拷贝从而解决有多个基类拷贝所产生的二义性问题。

12. 在函数调用过程中，什么是赋值传递，什么是引用传递?

赋值传递是将实参赋值给形参变量，然后执行被调函数体。赋值传递有两种形式，一是直接传常量或变量值，二是传变量的地址。(**引用传递≠地址传递**)

引用传递是将形参引用给形参，需要形参与实参指的是同一变量。

13. const 有什么用途？（请至少说明两种）

（1）可以定义 const 常量

（2）const 可以修饰函数的参数、返回值，甚至函数的定义体。被 const 修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。

14.类的指针成员为什么要用 new 另辟内存空间?

为了保证类的封装性，类中的指针成员所指向的内存空间必须在类的定义中自行独立开辟和释放。

15. 虚析构函数有什么作用?

对象销毁时，需要调用析构函数。在多态调用时，是用基类的指针访问派生类的对象。如果析构函数是非虚函数，则基类指针只能访问基类的析构函数，而不能访问派生类的析构函数，导致派生类对象销毁时，没有调用派生类的析构函数，只是调用了基类的析构函数。

如果把析构函数定义成虚函数，则可克服这个问题。

16. 拷贝构造函数在哪几种情况下调用?：

1.用一个对象初始化另一个对象时

2.当用对象作为函数参数传递时

3.当函数返回对象时

17. 函数重载与函数覆盖有什么不同，它们与多态有什么关系?

函数重载是指函数名相同，而函数的参数个数或类型不同；覆盖是指在派生类中成员函数与基类成员函数的函数名、参数个数、类型与返回值均相同；C++中正是通过虚函数的覆盖，实现多态的功能。

18. C++继承是如何工作的?

继承使得派生类能够使用基类的公有和保护成员，从而实现代码的复用，派生类可以增加成员，也可以隐藏和覆盖基类的成员。对于公有继承，基类成员的访问权限在派生类保持不变。

19. 类与对象有什么区别?

类是类型，是对象的抽象，对象是类的具体实例。一个类可以有多个对象，每个对象都有自己的存储单元，而类不占存储单元。

第三部分　C++真题编程题

1. （2011）设计一个类 MyARR，要求实现以下功能：

数据成员：

1 .数组长度　　2.数组名(Q1：定义指针数组，动态分配？可数组)

函数成员：

MyARR()　　//构造函数（Q2：要传 int[]参数吧？要不怎么初始化？对）

Void deletesame()　//删除数组中元素相同的，只剩一项

Void show()　//要求显示出各个元素

要求：顺序数组 b[16]={1,2,2,2,3,4,5,5,5,6,6,7,8,9,10}，用数组 b 去初始化 MyARR，去掉重复，调用 show()输出元素为 1,2,3,4,5,6,7,8,9,10，实现这个类并测试。

```cpp
#include<iostream>
#include<cstring>

using namespace std;
class myARR
{
private:
    int a[100]; //ATTENTION 1 :定义数组不用深拷贝，浅拷贝就可以
    int len;
public:
    myARR(const int* a,const int len); //ATTENTION 2:整型数组需传长度
    void deletesame();
    void show();
}; //这里分号

//构造函数实现：初始化私有成员
myARR::myARR(int a[],int len)
{
  this->len = len;
  memcpy(this->a,a,len * sizeof(int)); //ATTENTION 3:字符数组用 strcpy()
}

//[背]函数：使顺序数组成员的元素都不重复
void myARR::deletesame()
{
  int temp = a[0]; //先判断第一个元素是否重复
  for(int i =1 , j = 1 ; i < len ; i++)
  {
```

```
        if(a[i] == temp ) //有重复
        {
            continue; //什么都不做 i 后移动
        }
        else
        {
            a[j++] = a[i];
            temp = a[i];//[删除数组指定元素模板增加的部分]判断下个元素是否重复
        }
    }
    len = j; //易错 1：记得修改长度
}

//函数：输出数组成员
void myARR::show()
{
    for(int i = 0 ; i < len ; i++)
    {
        cout<<a[i]<<" ";
    }
}
int main()
{
    int b[16] = {1,2,2,2,3,4,5,5,5,6,6,7,8,9,10,10};
    myARR m(b,16);
    m.deletesame();
    m.show();   //测试输出：1 2 3 4 5 6 7 8 9 10
    return 0;
}
```

2. （2012）实现一个 myString 类，拥有长度 len 和 char 数组成员。重载+：使得字符串可以直接相加；重载<<:使得 cou<<s 可以直接输出 myString 对象 s 的 char 数组成员。

```
#include<cstring>  //玄：去掉 std+.h,即可编译(iostram.h,ostream.h 保留)
#include<iostream.h>
#include<ostream.h>
#define MAX_LEN 1000

class MyString
{
    char s[MAX_LEN];
    int len;
public:
    MyString();
    MyString(const char *str);
    friend MyString operator + (const MyString &s1,const MyString &s2)
    friend ostream& operator << (ostream &out, MyString& s);
};

MyString::MyString()
{
    len=0;
}
//构造函数实现：初始化私有成员
MyString::MyString(const char *str)
{
    len = strlen(str);
    strcpy(s,str);
}

//ATTENTION 1:这里重载非成员。
//重载成员的问题：1.也许不希望改变当前对象 2.传递的对象无法使用私有成员
            // 3.MyString::错误，非成员 4.不要写 friend  5.返回对象!
MyString operator + (const MyString &s1,const MyString &s2)//无 friend
{
    MyString s;
    strcpy(s.s,s1.s); //先复制 s1.s
    strcat(s.s,s2.s); //再连接 s2.s
    return s;
}
//重载<<非成员输出 s.s :1.就是相当先输出 s.s??? 2.有 ostream 类，只能非成员!!
                    //3.返回引用，是需要连续复制（否则返回对象也可）
ostream& operator << (ostream &out, MyString& s)
{
```

```
    out<<s.s;
    return out;
}
int main()
{
    MyString s1("1234");
    MyString s2("5678");
    MyString s3;
    s3=s1+s2;
    cout<<s3<<endl;//输出 : 12345678
    return 0;
}
```

3. （2013）实现一个单链表，并用高效的算法实现查找链表中倒数第 K 个元素。

分析：

1. 分两部分分别定义节点和链表（含成员 head）
2. 节点的 node* next、head 是指针，指向 node 对象
3. 模板类不能为 typedef 定义且：
   (1) 记得每个函数/类/结构体，用到模板类都要前面加
   (2) 初始化： `LinkList<int>*  L = new LinkList<int>;`
   (3) 作参数时：`void Insert (LinkList<type>* L, type e)`

```
#include <iostream>
using namespace std ;

//定义：节点
 template<class type>
 struct LNode
 {
    type data ;
    LNode<type>* next ;
};

//定义：链表。有成员:头结点
 template<class type>
 struct LinkList
 {
     LNode<type>* head ;//易错 0：不分配空间
 };

 /*
   //插入：尾插法（顺序），放在初始化前
          1.生成节点（最后一个节点）给节点赋值且指向 NULL（注意）
          2.找到链表最后一个节点
          3.指向生成的节点
 */
template<class type>
void Insert (LinkList<type>* L, type e)
{
    LNode<type>* p = L->head; //工作指针找到最后一个节点
    LNode<type>* node = new LNode<type>; //ATTENTION 1:这里要声明指针!
    node->data = e;
    node->next = NULL;
    while(p->next != NULL)
```

```
        p = p->next;
    p->next = node;
}

 /*           head
    //生成链表 : 口-->口-->...-->口
      1.利用前面声明的 insert 函数
 */
template<class type>
void init (LinkList<type>* L) //L 初始已经指向了头结点
{
    type e;
    cout<<"请输入元素值，按下 0 结束："<<endl;
    while(1)
    {
      cin>>e;
      if(e == 0) break;
      Insert(L,e);
    }
}

/*
  查找倒数第 k 个元素：
         1. 设置工作指针 p q 初始都执行 L
         2. p 先走 k 次，然后 pq 同时走，直至 p 到末尾
         3. 此时 q 所指向就是倒数第 K 个元素
*/
template<class type>
type findK(LinkList<type>* L,int k)
{
    LNode<type>* p = L->head ,*q = L->head;
    while(k-- > 0)  p = p->next;
    while(p != NULL)
    {
        p = p->next;
        q = q->next;
    }
    return q->data;
}

/*
  删除:指定节点删除
     1.设置 p = head ， pre = NULL（注意）
     2.p 遍历节点：下次遍历之前都要 pre = p 直至找到
```

```
77.      3.找到节点：pre->next = p->next;
78. */
79. template<class type>
80. void del (LinkList<type>* L , LNode<type>* delNode)
81. {
82.     LNode<type>* p = L->head ,*pre = NULL;
83.     while(p != delNode)
84.     {
85.         pre = p;
86.         p = p->next;
87.     }
88.     pre->next = p->next;
89.     delete(p);
90. }
91.
92. //输链表出：遍历
93. template<class type>
94. void show(LinkList<type>* L)
95. {
96.      LNode<type>* p = L->head->next; //易错 1：头节点别输出
97.      while(p != NULL)
98.      {
99.        cout<<p->data<<" ";
100.        p = p->next;
101.      }
102. }
103.
104. int main ()
105. {
106.     LinkList<int>*  L = new LinkList<int>; //链表 L 同时也是头结点
107.     L->head = new LNode<int>;  //使用头结点记得分配空间
108.     L->head->next = NULL ; //易错 2：初始指向 NULL
109.     init(L);
110.     del(L,L->head->next); //删除第一个节点
111.      show(L);
112.     cout<<"倒数第 2 个元素值为："<<findK(L,2)<<endl;
113.     return 0;
114. }
```

4.（2013）创建一个抽象类 stack，分别用顺序结构和链式结构实现，要求栈内元素可以是任意类型。

分析 1：链栈类

1.链栈类实现和链表一样分为两部分：节点和栈（含成员 top）

2.但是这里把 struct stack ---> class stack(节点还是：struct node),产生的变化有：

(1) stack 里可以放函数了（而且有 stack 类型参数可不传了）

```
#include<iostream>
using namespace std;

//定义:链栈节点
template <class type>
struct Node
{
    type date;
    Node* next;
};

//定义：链栈类
template <class type>
class stack
{
    private:
        Node<type>* top; //ATTENTION 0:下面设计栈顶指针初始==NUll
    public:
        //stack(const stack& s); //ATTENTION 1:有动态指针一定要复制构造
        //Node<type>* getTop() { return top ;} //用不到，成员函数直接获取
        stack(){ top = NULL;} //易错 1:不同前给链表初始化，指针 head 分配空间
        void push(type e);
        type pop();
        void init(); //可不传 stack 类型参数了
        void show();
};

//入栈：和前一样先实现入栈(插入)方便下面函数
template <class type>
void stack<type>::push(type e)
{
  //Node<type>* top = getTop(); //成员函数直接获取就行
  Node<type>* node = new Node<type>;
  node->date = e;
```

```
35.   node->next = top;
36.   top = node;
37. }
38.
39. //出栈:是返回元素，不是返回指针！（指针已被删除无法被返回！）
40. template <class type>
41. type stack<type>::pop()
42. {
43.   type temp_e = top->date;
44.   if(top == NULL)
45.   {
46.       cout<<"underflow"<<endl;
47.       return NULL ;
48.   }
49.   Node<type>* temp = top; //保存当前栈顶指针
50.   top = top->next;        //栈顶指针改变
51.   delete(temp);           //删除原有栈顶指针
52.   return temp_e;
53. }
54.
55. //初始化：（实际答题可考虑省略）
56. template <class type>
57. void stack<type>::init()
58. {
59.     type e;
60.     cout<<"请输入元素值，按 0 结束："<<endl;
61.     while(1)
62.     {
63.         cin>>e;
64.         if(e == 0) break;
65.         this->push(e);
66.     }
67. }
68.
69. //输出值:
70. template <class type>
71. void stack<type>::show()
72. {
73.   while(top != NULL)
74.   {
75.     cout<<top->date<<"  "<<endl;
76.     top = top->next;
77.   }
78. }
```

分析 2：顺序栈类

1. 和前链表/链栈不同：顺序栈不定义节点类，只有栈类。
2. 栈类存放全部节点（数组）和~~-长度（初始-1）代替~~top

```
#include<iostream>
using namespace std;
#define MAX 1000

template<class type>
class stack  //除 1.初始声明 2.构造函数。凡是 stack 全部 stack<type>
{
    private:
        type elem[MAX];
        int top;
    public:
        stack(){ top = -1 ;} //同前，top 初始-1
        void push(type elem);
        type pop();
        //void init();//同前，若要简单，可不实现
        void show();
};

/***********顺序栈实现************/

//1.push:(1)判断是否栈满 & (2)压入改变 top 指针即可
template<class type>
void stack<type>::push(type elememt)
{
    if(top == MAX-1)
    {
        cout<<"overflow"<<endl;
        exit(1);
    }
    elem[++top] = elememt; //压栈同时改变了指针
}

//2.pop:(1)判断是否栈空 & (2)弹出改变 top 指针即可
template<class type>
type stack<type>::pop()
{
    if(top == -1)
    {
        cout<<"underflow"<<endl;
```

```
        exit(1);
    }
    return  elem[top--]; //弹出同时改变了指针
}

//3.show：遍历输出
template<class type>
void stack<type>::show()
{
    int p = top;    //易错 1：不要直接用 top，改变 top 值
    while(p > -1)
        cout<<elem[p--]<<" "<<endl;
}

int main()
{
  stack<int> s;
  s.push(0); s.push(1); s.push(2); s.push(3);
  s.show();
  s.pop(); s.pop();
  s.show();
  return 0;
}
```

5.(2014)有一个抽象类 shape，含有计算周长、面积的成员函数。派生出类 rectangle 和 square，试着写出 main 函数，用多态性验证结果。

分析：

同前例题：第八章-第五小题

```
class Shape
{
  public:
    Shape(){}
    virtual float GetArea() =0 ;//ATTENTION 1：声明纯虚函数才是抽象类!
};
class Circle : public Shape
{
  public:
    Circle(float radius):itsRadius(radius){}
    float GetArea() { return 3.14 * itsRadius * itsRadius; }
  private:
    float itsRadius;
};
class Rectangle : public Shape
{
  public:
  //易错 1：别忘构造，算我求你了
   Rectangle(float len，float width):itsLength(len)，itsWidth(width){};
   virtual float GetArea() { return itsLength * itsWidth;}//virtual 可省
  private:
   float itsWidth;
   float itsLength;
};
int main()
{
   Shape * sp;
   sp = new Circle(5);
   cout << "The area of the Circle is " << sp->GetArea () << endl;
   delete sp;
   sp = new Rectangle(4，6);
   cout << "The area of the Rectangle is " << sp->GetArea() << endl;
   delete sp;
   return 0;
}
```

6.（2016）设计并实现一个栈类，有出栈和进栈的功能。当进栈满是输出 overflow，出栈无数据时输出 underflow。

分析：

参照：2013 年真题 C++编程真题

7.（2016）设计并实现一个通用的 dictionary，其中存储的一个数据分为键和值两个元素，键是唯一的，值不唯一。

分析：

主要就是包装一下 multilmap 进行字典类实现（我认为。。）

```cpp
#pragma warning(disable : 4786)  //去除愚蠢的 4786 错误
#include<iostream>
#include<map>
#include<utility>  //用来构造二元组
#include<string>
using namespace std;

typedef multimap<char,string>::iterator  mmp_iter; //太长了用别名
template<class KEY,class ELEM>
class dictionary
{
public:
    multimap<KEY,ELEM> m; //第一次 c++用 string 想想还有点紧张呢
public:
    void insert(KEY k,ELEM e);
    void del(KEY key);
   pair<mmp_iter,mmp_iter> find(KEY key);//key 为 a-z 返首尾迭代器二元组
};

/*********************实现*****************************/
/*
  插入：利用 multilmap.insert,但不用构造二元组即可插入
*/
template<class KEY,class ELEM>
void dictionary<KEY,ELEM>::insert(KEY k,ELEM e)
{
   m.insert(make_pair(k,e));
}

/*
  删除：利用 multilmap.erase,删除所有键值为 key 的元素
*/
template<class KEY,class ELEM>
void dictionary<KEY,ELEM>::del(KEY key)
{
    m.erase(key);
}
```

```
38.
39. /*
40.   查找：1.利用 multilmap.equal_range 函数，返回二元组迭代器组
41.         (1)二元组 pair<key 对应首元素迭代器，key 对应尾元素迭代器>
42.         (2)且这两迭代器，指向的也是个二元组 pair<字母索引,对应字符串>
43.   例如：verctor<int>::iterator = v.begin() /v.end() //函数返回的是迭代器
44.        ostream<int>::iterator = ostream_iterator<int>(cout,"");注意泛型
45. */
46. template<class KEY,class ELEM>
47. pair<mmp_iter,mmp_iter> dictionary<KEY,ELEM>::find(KEY key)
48. {
49.     return m.equal_range(key);
50. }
51.
52. int main()
53. {
54.   dictionary<char,string> dic;
55.   dic.insert('a',"abondom");
56.   dic.insert('b',"big");
57.   dic.insert('c',"central");
58.   dic.insert('c',"cental");
59.   pair<mmp_iter,mmp_iter> p = dic.find('c'); //再错 0：p 非指针 p.引用成员
60.   //易错 1：*(iter->second)错误，它不是指针 iter 才是
61.   for(mmp_iter iter = p.first ; iter != p.second ; iter++)
62.       cout<<iter->second<<" ";
63.   return 0;
64. }
```

8.（2017）利用 vector 数组模板，从标准输入输入 10 个数，逆序排序后从标准输出输出。

分析：

1.定义一个 vector<int> s

2.用户输入初始化+对**每个**元素操作：ctrl+Z +回车 结束

transform(istream_iterator<**int**>(cin),istream_iterator<**int**>(),istream_iterator<**int**>(cout,“”),对每个元素操作函数)

3.用户输入初始化+对**整体**元素操作（排序）：

(1) for　s.push_back(元素)　(循环输入插入元素)

(2) sort 排序

(3)reverse()

```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

int main()
{
  vector<int> v; //vector 是可扩展的数组，不用指定大小
  int i,elem;
  vector<int>::iterator iter; //ATTETTION 0:如何声明 vector 型迭代器
  cout<<"请输入 10 个元素，空格分开:"<<endl;
  for(i = 0 ; i < 10 ; i++)//输入 4 1 3 5 2 ...
  {
      cin>>elem;
      v.push_back(elem);
  }
  sort(v.begin(),v.end()); //顺序排序
  reverse(v.begin(),v.end()); //倒转：实现逆序排序
  for(iter = v.begin() ; iter != v.end(); iter++)
      cout<<*iter<<" ";
  return 0;
}
```

9.（2017）定义一个电脑类，数据成员有 cpu 速度，内存大小，磁盘容量，并有复制构造函数、构造函数。同时对<< 、>>、=、>、< 等运算符进行重载。

分析：

1.重载成员/非成员?

(1)有非自定义类如：重载<<、>>有 ostream/istream 就一定要重载非成员。

(2)其余时候成员和非成员都可以。如重载 >、<。

2.返回引用/对象?

(1)需连续赋值一定要返回引用，如：=、[]、前置++。

(2)其余时候能返回引用则返回（局部对象引用一定不能返回）。否则返回对象。

3.参数引用?

(1)重载参数**都引用**，开销小

```
#include<iostream.h>
#include<ostream>
#include<istream>
class computer
{
private:
    double speed;  double memsize;  double disksize;
public:
    computer(double speed,double memsize,double disksize);
    computer(const computer& c);
    computer& operator = (const computer& c); //返回引用：需连续赋值,下同
    int operator < (const computer& c);//成员也可
    int operator > (const computer& c);
    friend ostream& operator << (ostream& out,const computer& c);
    friend istream& operator >> (istream& cin, computer& c);
};
/**********************实现*****************************/
//构造函数：易错 1：列表初始化不能用 this 指针，此时还没对象还没初始化完成！
computer::computer(double _speed,double _memsize,double _disksize):
              speed(_speed),memsize(_memsize),disksize(_disksize){}
//复制构造函数：题目要求，无动态其实可以不用
computer::computer(const computer& c)
 {
   speed = c.speed;  this->memsize = c.memsize; disksize = c.disksize;
 }
/*
  重载=：为什么重载=，有复制构造不就成了吗？
         复制构造用于初始化；重载=用于不同对象赋值
*/
 computer& computer::operator = (const computer& c)
```

```
 {
    speed = c.speed;  memsize = c.memsize; disksize = c.disksize;
    return *this;
 }
//重载 <：1.无非自定义类，重载成员函数也可。2.这里比较 speed 大小
        //3.重载成员要加 computer::，非成员无需
int computer::operator < (const computer& c)
{
  if(this->speed < c.speed)  return 1;
  else                       return 0;
}
//重载 >:
int computer::operator > (const computer& c)
{
  if(this->speed > c.speed)  return 1;
  else                       return 0;
}
//重载<<:注意 out 是自定义的 ostream 对象，不是 cout
ostream& operator << (ostream& out,const computer& c)
{
    out<<"速度："<<c.speed<<"  内存："<<c.memsize
        <<"  外存："<<c.disksize<<endl;
    return out;
}
//重载>>:相当叫用户连续输入 cpu 速度、内存、外传大小。
     //易错 2：这里别加 const computer& c!
istream& operator >> (istream& in, computer& c)
{
    cout<<"请输入 cpu 速度、内存、外传大小："<<endl; //系统对象 cout
    in>>c.speed>>c.memsize>>c.disksize;//自定义对象 in
    return in;
}
int main()
{
  computer c1(100,100,100),c2(200,200,200);
  cout<<c1;    //输出：  100 100 100
  cout<<c2;    //输出：  200 200 200
  cin>>c2;     //输入：  300 300 300
  cout<<c2;    //输出：  300 300 300
  int b = c1 > c2 ; //b = 0
  c1 = c2;
  cout<<c1;    //输出 300 300 300
  return 0;
}
```

10.（2018）定义一个 Time 类，有数据成员 minute、second、hour。要求重载构造函数，复制构造，析构函数（输出 goodbye）。成员函数 setTime()设定时间，printTime()打印时间。

分析：

```cpp
#include <iostream>
using namespace std;

class time
{
    private:
        int Hour,Minute,Second;
    public:
        time(int h,int m,int s);
        time();
        time(const time& t);
        ~time();
        Settime(int h,int m,int s) { Hour=h; Minute=m;Second=s; }
        PrintTime() {
    cout<<"Hour: "<<Hour<<"Minute: "<<Minute<<"Second: "<<Second<<endl;
        }
};
//构造函数：列表初始化
time::time(int h,int m,int s): Hour(h),Minute(m),Second=(s){ }
//委托构造函数初始化
time::time(): time(0,0,0){ }
//复制构造函数
time::time(const time& t)
{
    Hour=t.Hour;   Minute=t.Minute;  Second=t.Second;
}
//析构函数
time::~time()
{
    cout<<"Good bye!"<<endl;
}
int main()
{
    time A ,B(10,40,50);
    A.Settime(9,20,30);
    A.PrintTime();
    B.PrintTime();
    return 0;
}
```

11.（2018）定义一个 circle 类，继承于 point 类，有 x y 圆心坐标，半径大于 0。重载 =：实现对象赋值，重载<<:可以输出圆信息，重载+实现圆偏移。

分析：　　=　　<<　　+

1.重载成员/非成员：　都可　|有非自定义类，成员|　同左，成员

2.返回引用/对象：　连续赋值，引用|　连续赋值，引用　|对象（不能返局部对象引用）

3.参数引用/非引用:　全都引用

```
#include<iostream.h>
#include<ostream>

class point
{
private:
    double x , y;
public:
    point(double x1,double y1):x(x1),y(y1){}
    point(){ x = 0; y = 0; }
    double X(){ return x; }
    double Y(){ return y; }
};
class circle:public point
{
private:
    point  centre;
    double r;
public:
    //再错 1：这里不能调用委托构造函数初始无参构造？
    circle(point centre1,double r1):centre(centre1),r(r1){};
    circle(){centre = point(0,0); r = 0 ;};
    circle& operator =(const circle& c);
    friend ostream& operator <<(ostream& out,circle& c);
    friend circle  operator +(circle& c,const int offset);
};
/*****************实现************************/

/*
  重载=：实现对象赋值
  [注 1]私有成员是类外部对象不能访问,这里函数都是类内部定义的。对象 c 可以访问
  [注 2]如下面 c.centre.x 访问就会出错，c.centre 是外部对象
*/
```

```
34. circle& circle::operator=(const circle& c)
35. {
36.     centre = c.centre; r = c.r;
37.     return *this;
38. }
39. 
40. //重载<<:输出圆类信息
41. ostream& operator <<(ostream& out,circle& c)
42. {
43.      out<<"坐标为：("<<c.centre.X()<<","<<c.centre.Y()<<")"<<";半径为："<<c.r<<endl;
44.     return out;
45. }
46. 
47. //重载+:实现圆偏移
48. circle  operator + (circle& c,const int offset)
49. {
50.     circle temp;
51.     temp.centre = point(c.centre.X()+offset,c.centre.Y()+offset);
52.     temp.r = c.r;
53.     return temp;
54. }
55. 
56. int main()
57. {
58.     point centre(1,1);
59.     circle c0,c1(centre,1);
60.     c0 = c1;
61.     cout<<c0;      //输出：坐标为(1,1),半径为1
62.     c1 = c1 + 5;
63.     cout<<c1;      //输出：坐标为(6,6),半径为1
64.     return 0;
65. }
```