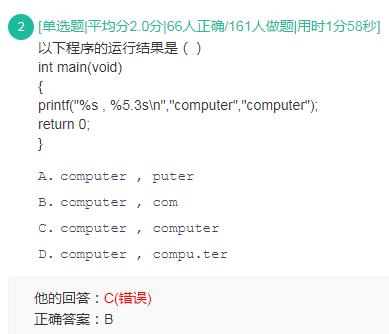
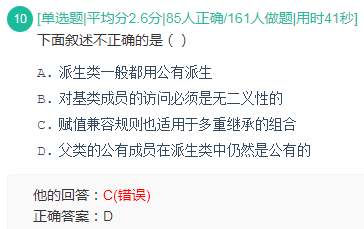
**11.6**



**%5.3s 输出占5列，但只取字符串中左端3个字符。这3个字符输出在5列的右侧，左补空格。所以输出前三个字符" com"**  
**%s**：例如:printf("%s", "CHINA")输出"CHINA"字符串（不包括双引号）  
**%ms**：输出的字符串占m列，如果字符串本身长度大于m，则突破m的限制,将字符串全部输出。若串长小于m，则左补空格。  
**%-ms**：输出的字符串占m列，如果字符串本身长度大于m，则突破m的限制,将字符串全部输出。如果串长小于m，则在m列范围内，字符串向左靠，右补空格。  
**%m.ns**：输出占m列，但只取字符串中左端n个字符。这n个字符输出在m列的右侧，左补空格。  
**%-m.ns**：其中m、n含义同上，n个字符输出在m列范围的左侧，右补空格。如果n>m，则自动取n值，即保证n个字符正常输出

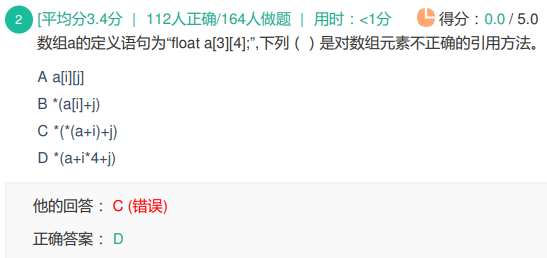


**自加运算符(++)的优先级要高于取值运算符(\*)的优先级，\*p++表示的是先将指针p指向下一个地址然后再取该地址的值，所以得到的结果并不是把year的值增至2010 选项B错误**

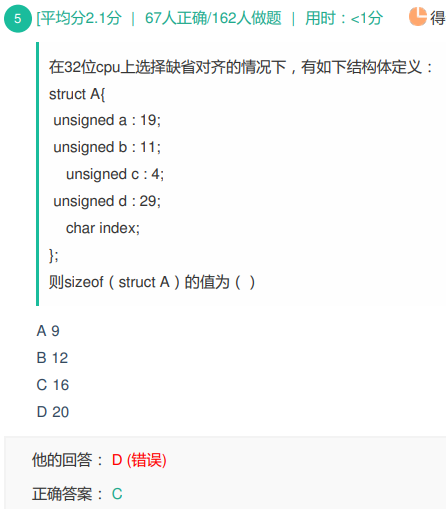


**C++语言中如果基类的成员是公有的，它被继承后在子类中该成员可能是公有的，也可能不是，主要看继承的权限.**

**11.7**

****

**实在不行可以把i=0和j=0带入验证**



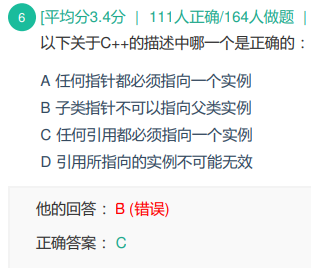
**unsigned a : 19;//表示变量a占19位，32位机每行32位，也就是4个字节，又因为题目支持缺省，所以 分为以下四行：19+11**

**4**

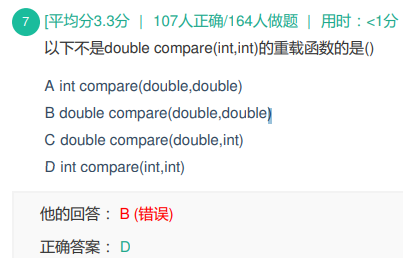
**29**

**8//char index 占1个字节，8个bit位**

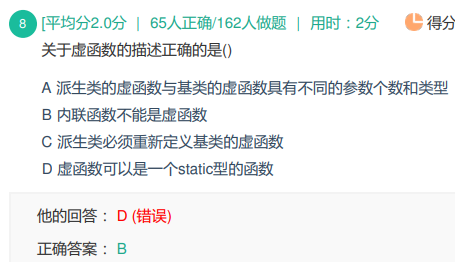
**一共四行，每行4个字节，所以结构体的大小是4\*4=16个字节（sizeof返回值以字节为单位）**



**B：可以强转**

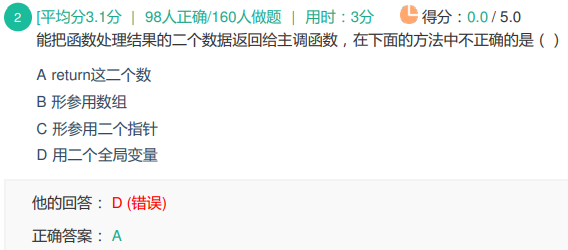


**因为重载函数与类型无关，所以不看类型，题目中的compare(int,int)与D中的compare(int,int)一样，编译器不能区分**

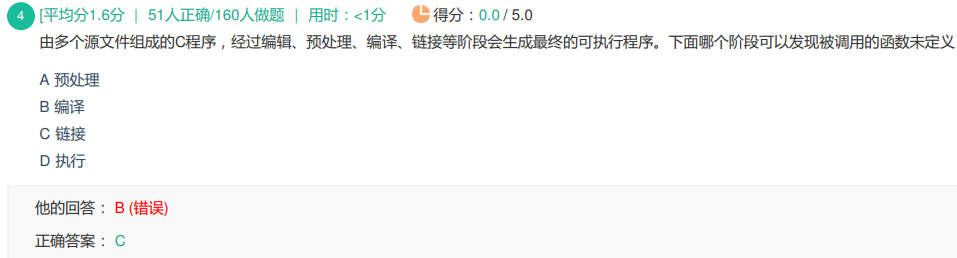




**11.8**

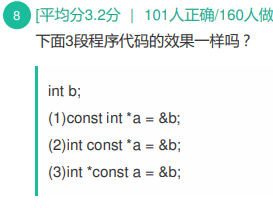


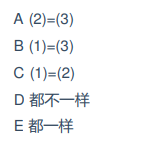
注意用两个全局变量在Clang编译器不支持

1**.预处理：相当于根据预处理指令组装新的C/C++程序。经过预处理，会产生一个没有宏定义，没有条件编译指令，没有特殊符号的输出文件，这个文件的含义同原本的文件无异，只是内容上有所不同。**

**2.编译：将预处理完的文件进行一系列词法分析、语法分析、语义分析及优化后，产生相应的汇编代码文件。**

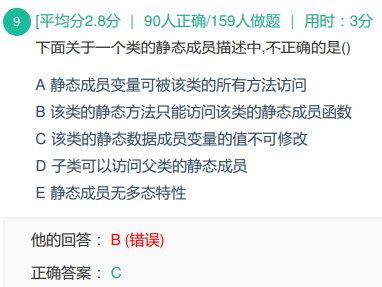
**3.链接：通过链接器将一个个目标文件（或许还会有库文件）链接在一起生成一个完整的可执行程序。 链接程序的主要工作就是将有关的目标文件彼此相连接，也就是将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来，使得所有的这些目标文件成为一个能够被操作系统装入执行的统一整体。在此过程中会发现被调用的函数未被定义。**





答案：C

**const在\*的左边，则指针指向的变量的值不可直接通过指针改变(可以通过其他途径改变);在\*的右边，则指针的指向不可变。简记为"左定值，右定向"。**



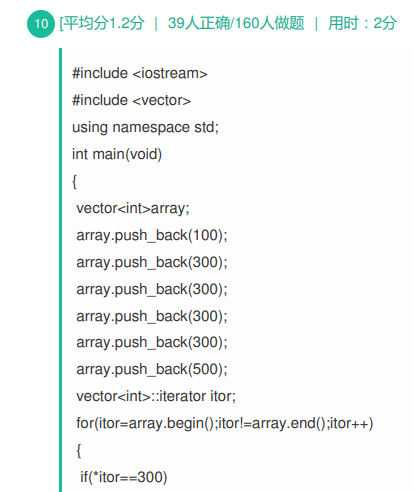
**类的静态成员属于整个类 而不是某个对象，可以被类的所有方法访问，子类当然可以父**

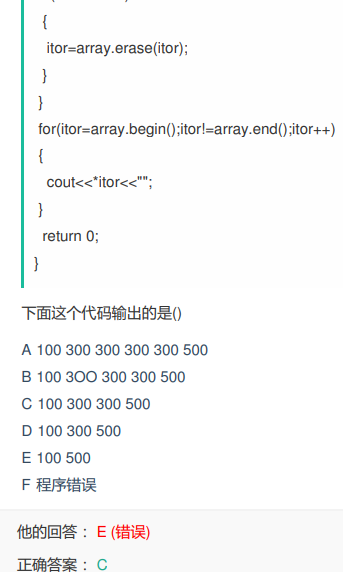
**静态成员；**

**静态方法属于整个类，在对象创建之前就已经分配空间，类的非静态成员要在对象创建**

**才有内存，所有静态方法只能访问静态成员，不能访问非静态成员；**

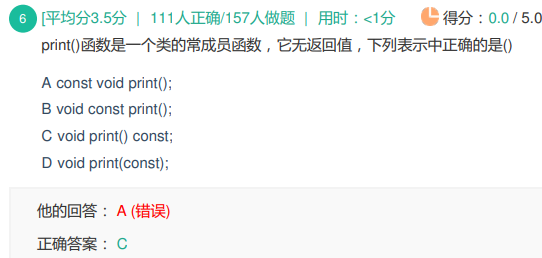
**静态成员可以被任一对象修改，修改后的值可以被所有对象共享。**





**防止迭代器失效：itor=array.erase itor指向替补上删除元素位置的下一个位置。**

**11.9**



**Const在函数名前和后的概念区别：**

**当const在函数名前面的时候修饰的是函数返回值，在函数名后面表示是常成员函数，该**

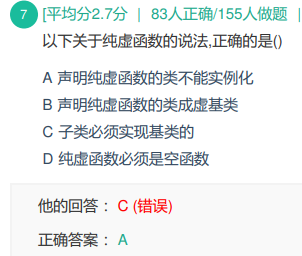
**函数不能修改对象内的任何成员，只能发生读操作，不能发生写操作。**

常成员函数说明格式如下：  
　　<类型说明符> <函数名> (<参数表>) const；

**一个函数名字后有const，这个函数必定是成员函数，也就是说普通函数后面不能有const**

**修饰，如：int print( ) const {.......} 这个函数必定为成员函数，即在类里面定义的函数。**

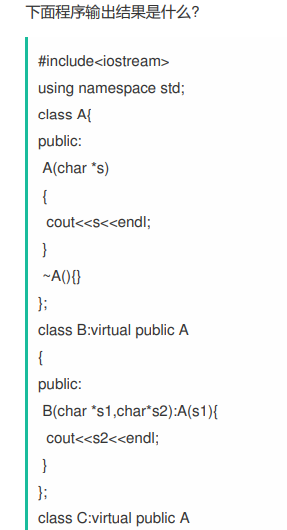
**在一个类里定义了一个const成员函数后，则此函数不能修改类中的成员变量，如果定义了一个类的const对象（非const对象可以调用const成员函数和非const成员hanshu )，它只能调用类中的const成员函数。**

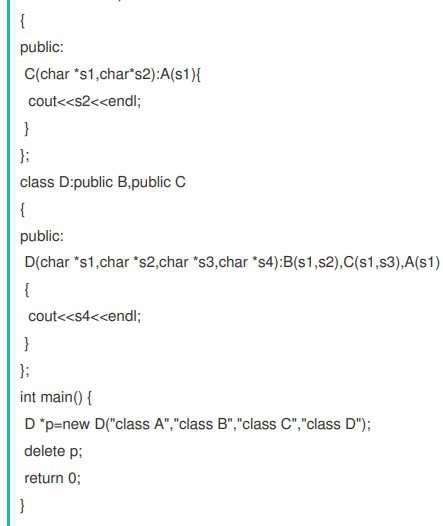


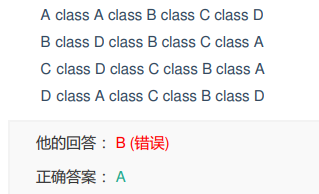
**含有纯虚函数的类是抽象类，对于继承这样的类的类（派生类）来说，如果派生类实现了基类的纯虚函数，则派生类可以实例化。**

**若派生类没有实现该纯虚函数，则该派生类也是抽象类，即不能实例化。**

**派生类不一定非得要实现基类的纯虚函数。**







**11.10**



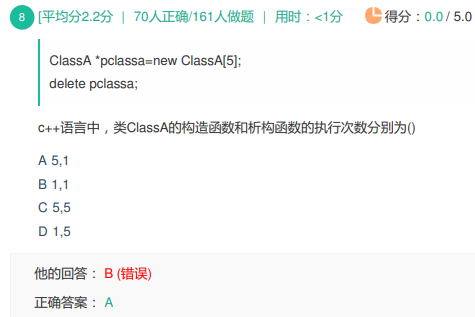


**注意是gcc编译器**

**printf函数执行的时候，会先把这三个数字压入栈里，然后再执行打印。压入栈的时候按照数据本身的长度来，首先把c和b压入，并且每一个都是8个字节（printf下自动转化为double）。然后再压入a是4个字节。然后再执行打印。打印的时候按照用户指定的格式来出栈。首先打印a，a打印正常。然后又打印4个字节长度的b，在栈里面由于b长度是八个字节，并且b目前是64位的表示方式，数据的后面全是0.（float 变double），电脑是小端存储方式，0存储在距离a近的地方。打印b的时候，打印的4个字节都是0。然后再打印c，c用正常的方式打印，会一下子读取8个字节，正好，读出来的八个字节前面四个字节全是0，自己可以算一下，实在太小了，因此为0。**

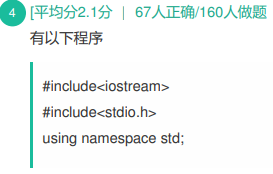


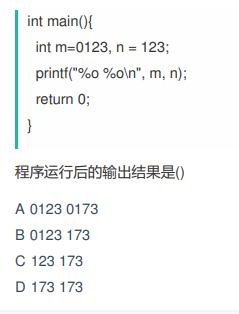
**A有歧义：只有在需要这些函数时才会创建。但D错误更明显，单例模式**



**delete和delete[] 的区别：delete和delete[]都能释放指针所指向的内存区域。但delete只会调用一次析构函数，而delete[]还会调用后续所有对象的析构函数。当数据类型为基本数据类型时，用delete和delete[]都可以，因为基本数据类型没有析构函数。**

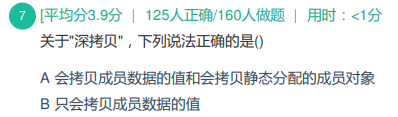
**11.11**

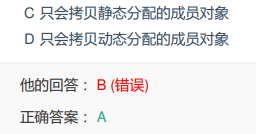


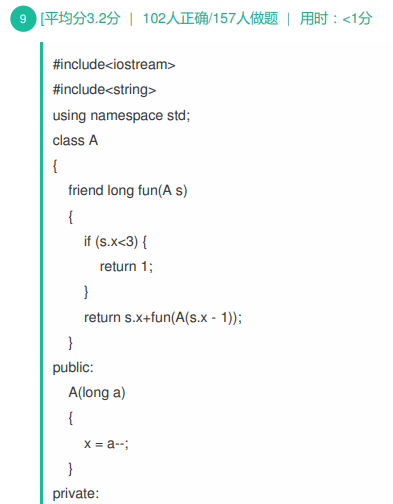


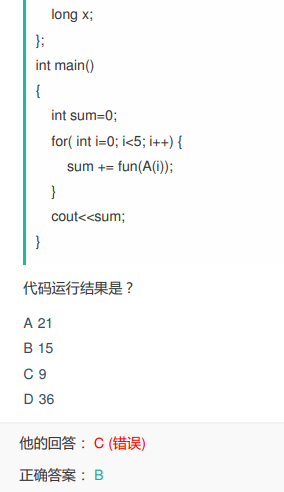


**注意八进制输出时，前面不会带o字母的，一定要注意。**

****

****

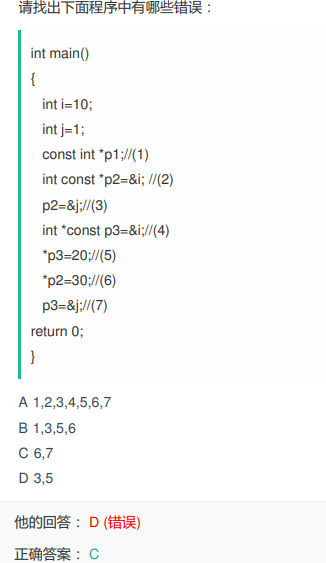
****

****

**11.12**

  
  
**一、count存储A-Z的个数，即count[0]存储A的个数，于是 ++count[a[i]-‘A’]； 'A’+i，count[i]；**

1. **count存储Z~A的个数，即count[0]存储Z的个数，于是 ++count[‘Z’-a[i]]； ’Z’-i，count[i]。 所以答案为D**

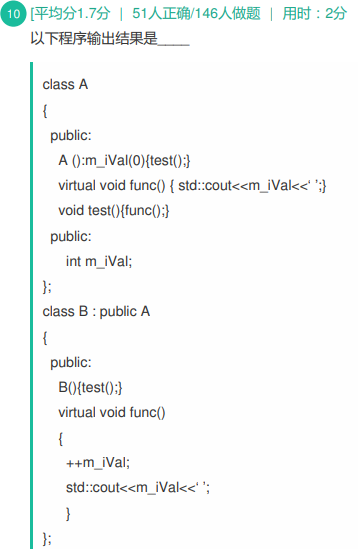


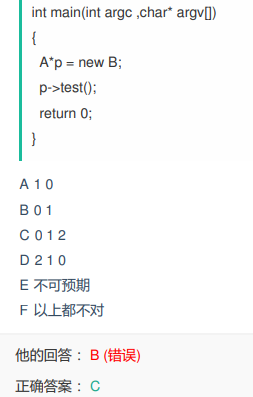
**左定值，右定向**

**const int\*p1：表示不能通过指针p1修改它指向的内存单元的值，但是p1本身可修改。**



1. **b各一次，c 三次，p是指针不会调用构造函数**





**这道题目考察派生类构造函数调用顺序：**

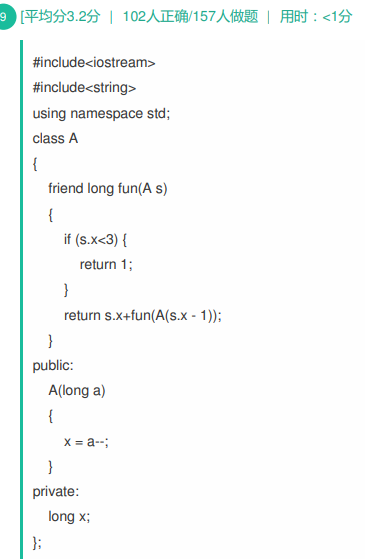
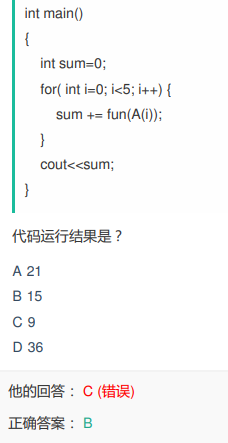
**生成派生类对象时，先调用基类的构造函数，再调用派生类自己的构造函数（析构时与构造的顺序相反）.**

**看main函数：**

**A\* p = new B; // 先调用基类A的构造函数, 输出m\_iVal 0; 接着调用B（派生类）自身的构造函数, 此过程最终调用的是void func(), 因基类中此方法是virtual, 故用B类自身方法, ++m\_iVal; 输出 1.**

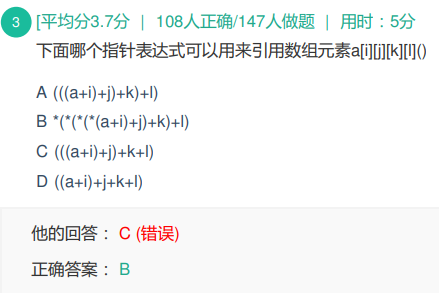
**p->test(); //指针指向B类对象，++m\_iVal 输出2.**

**11.13**

** **

**11.14 无**

**11.15**

****

a:整个四维数组的地址

\*(a+i) = a[i]

\*(a+i)+j = a[i] +j

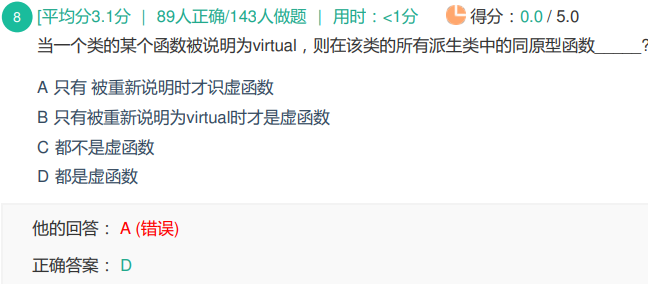
\*((a+i)+j) = a[i][j]

\*((a+i)+j)+k =  a[i][j]+k

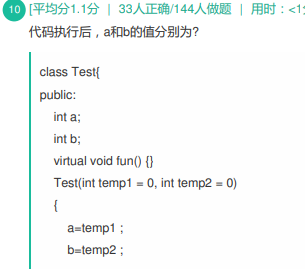
\*( \*((a+i)+j)+k ) = a[i][j][k]

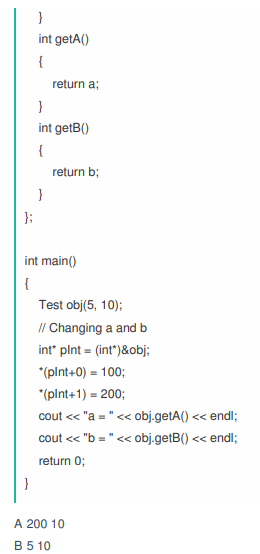
\*( \*((a+i)+j)+k )+l =  a[i][j][k]+l

**\*(** **\*(** **\*((a+i)+j)+k** **)+l** **) = a[i][j][k][l]**

****

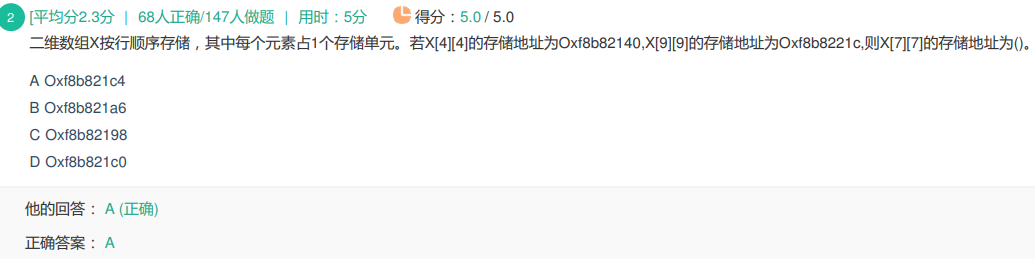
基[类的成员函数](http://baike.baidu.com/view/2345889.htm" \t "https://www.nowcoder.com/questionTerminal/_blank)设为virtual，其[派生类](http://baike.baidu.com/view/535532.htm" \t "https://www.nowcoder.com/questionTerminal/_blank)的相应的函数也会自动变为虚函数

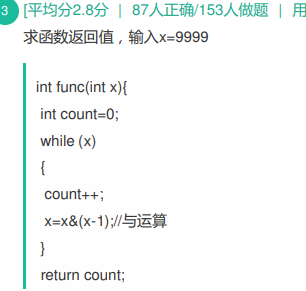






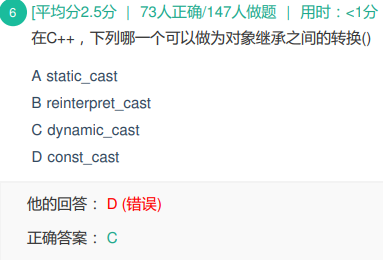
**11.16**

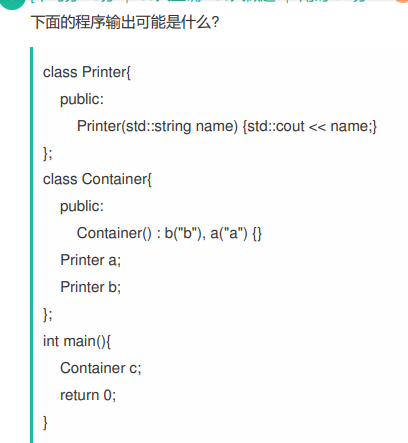


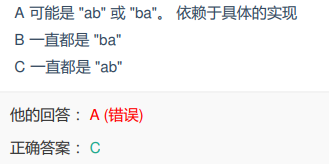


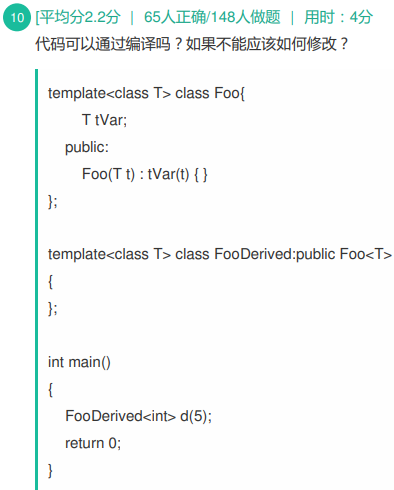
![)YG_PY](L@]$@H}~9J[4VMB](data:image/png;base64,)

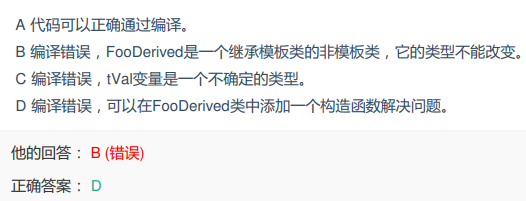
考察输入数据二进制中一的个数。9999 = 8192+1024+512+256+8+4+2+1；所以8个1.



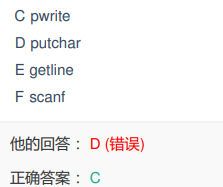
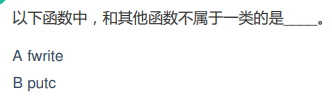




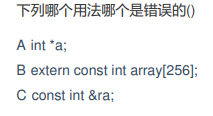




**11.17**

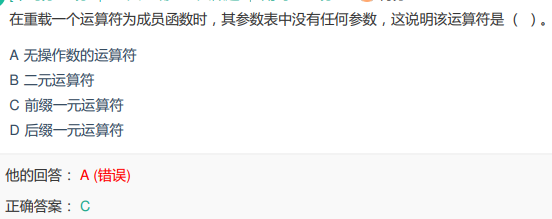
****

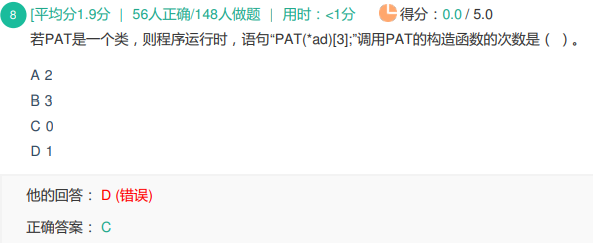
Pwrite是系统调用，其余的函数都是库函数





引用必须在声明时将其初始化，而不能像指针那样，先声明后赋值

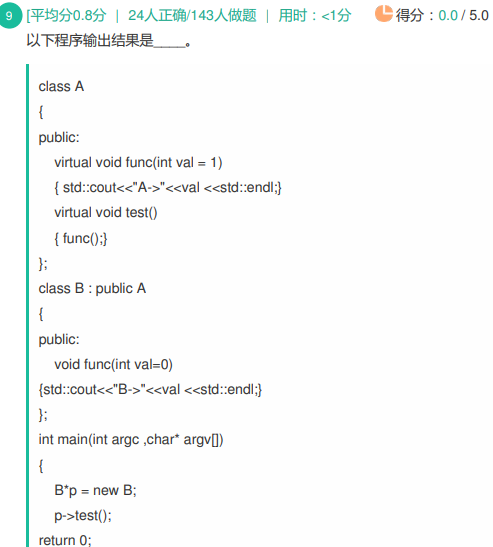


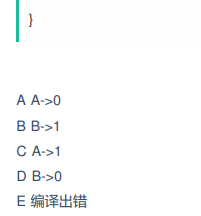


考察构造函数的调用次数.

pat (\*obj)[3]; 定义三个指向pat类型object的指针，所以构造函数调用次数为0

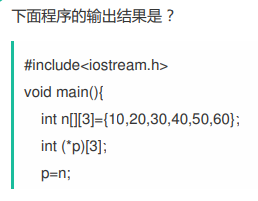
假如改为pat obj[3],则调用3次。

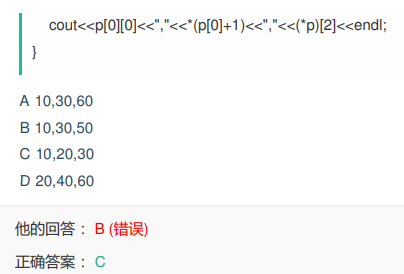






11.18





p[0][0]指的是第一个元素；

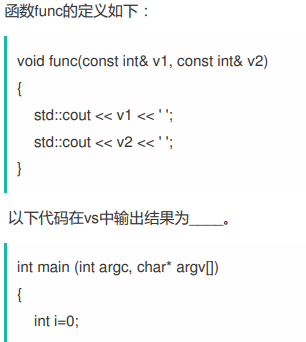
p[0]+1指的是第一行第二个元素；

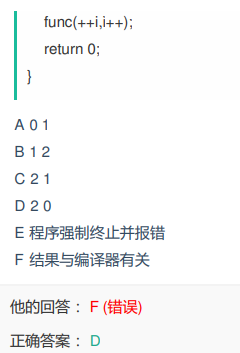
(\*p)[2]数组指针，表示第一行的第三个元素；

\*\*(p+1)表示第二行第一个元素；

  
  
1.在类A的析构函数中，delete一个非A类对象通常是没有问题的；   
在类A的析构函数中，delete一个类A的对象，就会造成死循环，堆栈溢出；  
在析构函数外使用 delete后，应该立即给指针赋值 NULL防止野指针。  
  
2.因为this是Myclass \* const this指针，也就是说this指针指向的对象(不是指向的对象的值)不可以改变，所以给this赋值在编译期间就不会通过，  
如果没有**this** = NULL这语句的话是栈溢出，因为会不停的调用析构函数。  
  
3.this被const修饰不能修改。

删掉this＝null后，在类的析构函数中调用delete this,delete this会去调用本对象的析构函数，而析构函数中又调用delete this，形成无限递归，造成堆栈溢出，系统崩溃。





C语言函数参数入栈顺序为由右向左,func(++i, i++)可以分解为  
参数 i 先入栈 输出0  
然后执行i++ 此时i为1  
接着参数 i 先执行++i i 此时为2 后入栈进行输出 2;

11.19