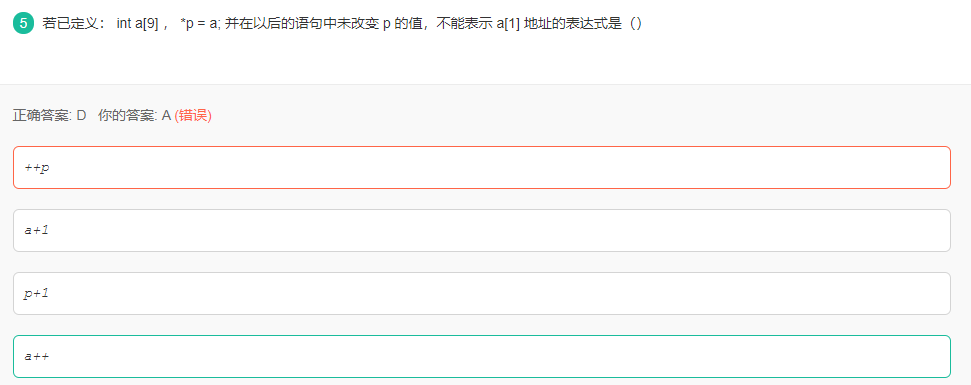
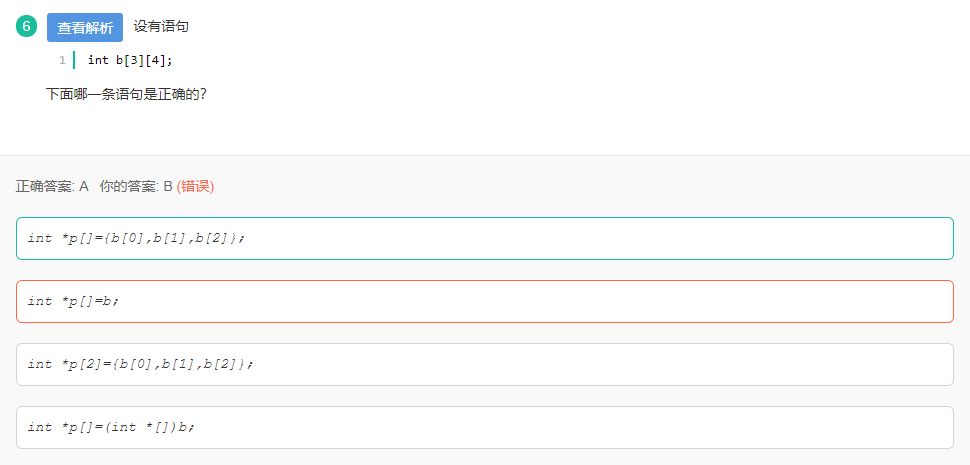
C++错题

## 数组



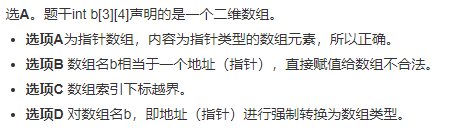
解答：

a表示数组名是个常量，不能进行自增自减操作。



解析：

b本身为数组指针，b[0]退化为一级指针，可以被添加到一个指针数组中。





解析：

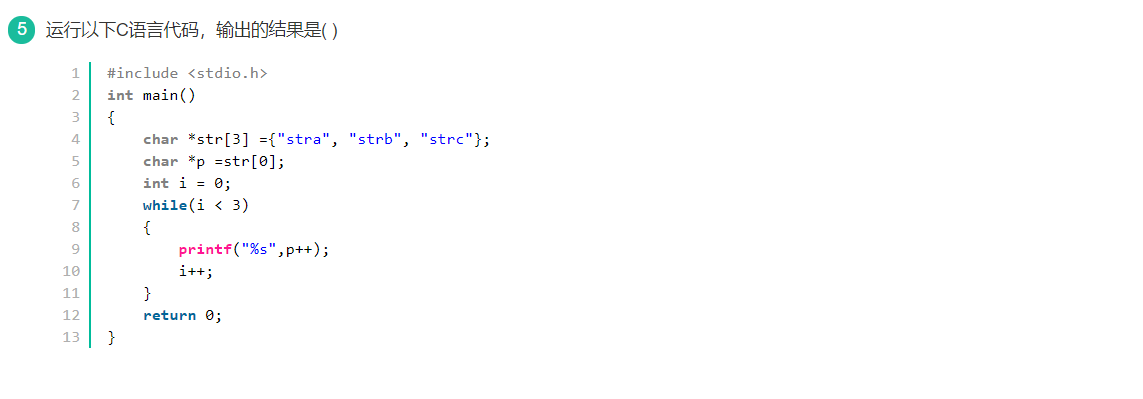
给数组初始化应该使用初始化列表{}。而int\* p = new int[40](); 是可以的，默认初始化都为0

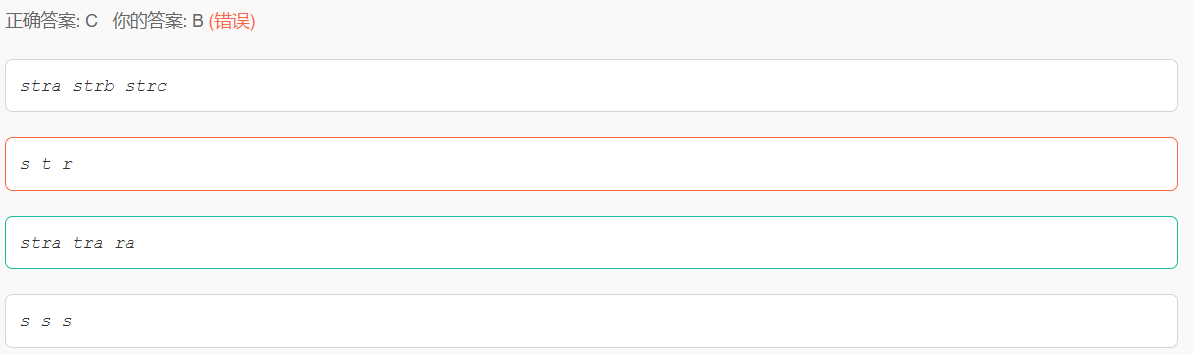


解析：

定义二维数组并初始化的时候，可以省略第一维的大小，但是第二维不可省略。

D中初值列表有一个是空的，这个是C++不允许的。





解析：

p指向str[0]，而不是str（p是char\*类型，而不是char\*\*类型），一次++并不能跳跃到str[1]。同时，输出是%s，就是碰到第一个结束字符才结束。



解析：

b指向二维数组，\*操作一次，析出一层（\*优先级高于+）。b直接进行+，是在行上（第一维）进行移动。\*b进行+，是在列上（第二维）进行移动。最后再加一个\*表示取出元素。C选项\*b+i表示b[0][i]。

C选项，表示的是b[0][i]+j的结果



解析：题目中int \*p[4]，根据优先级[]>\*应该理解成，含有4个int指针的数组，是指针数组。

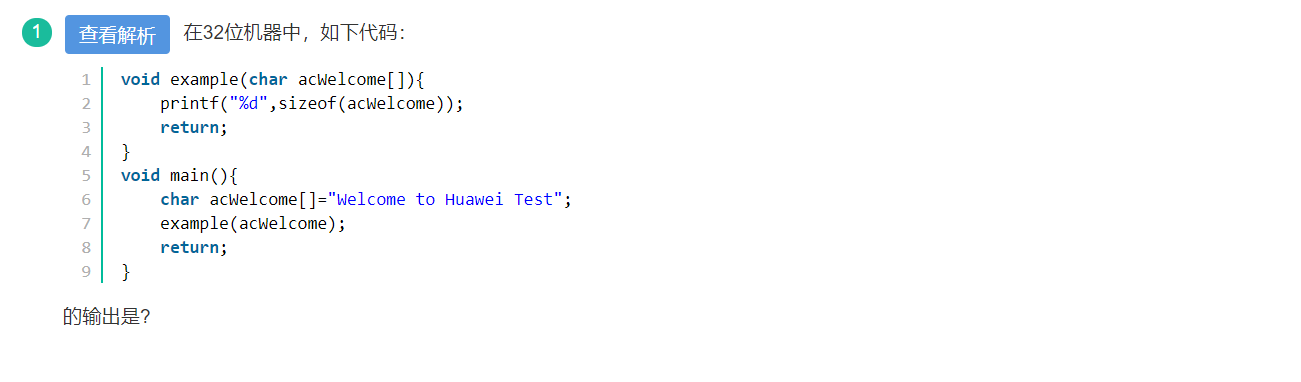
A是二维数组，不满足。B是二级指针，满足。C是二级指针数组，不满足。D是数组指针。



解析：

跟上题类似，数组指针应该是：类型名 (\*数组名)[下标表达式]。

区分数组指针，指针数组的本质，是看后面部分是指针还是数组。



解析：数组作为参数时，会退化成指针。而32位中指针长度为4。

## 继承



解析：

在派生类的构造函数中可以调用基类构造函数，但是不能对基类的子对象初始化（编译器会在派生类构造的时候会找不到基类的子对象，因为其实基类构造时，基类的子对象已经被构造了，也不需要重复构造）。同时，派生类构造函数中只能调用直接基类的构造函数，不能调用间接基类的。

class Item{

public:

int i;

Item(){i=0;printf("Item默认构造\n");}

Item(int i):i(i){printf("Item含参构造\n");}

};

class A{

public:

int a;

Item atemp;

A(int n):a(n){printf("A构造函数\n");}

};

class B:public A{

public:

int b;

Item btemp;

B(int n,int m):A(n),b(m),btemp(3){printf("B构造函数\n");}

};

int main(){

B bb(1,2);

cout<<bb.b<<" "<<bb.a<<" "<<bb.btemp.i<<" "<<bb.atemp.i<<endl;

return 0;

}

构造顺序：

基类构造函数，对象成员构造函数，派生类构造函数

析构顺序：

派生类析构函数，对象成员析构函数，基类析构函数

若在B类成员初始化列表中初始化atemp，则编译器会说找不到

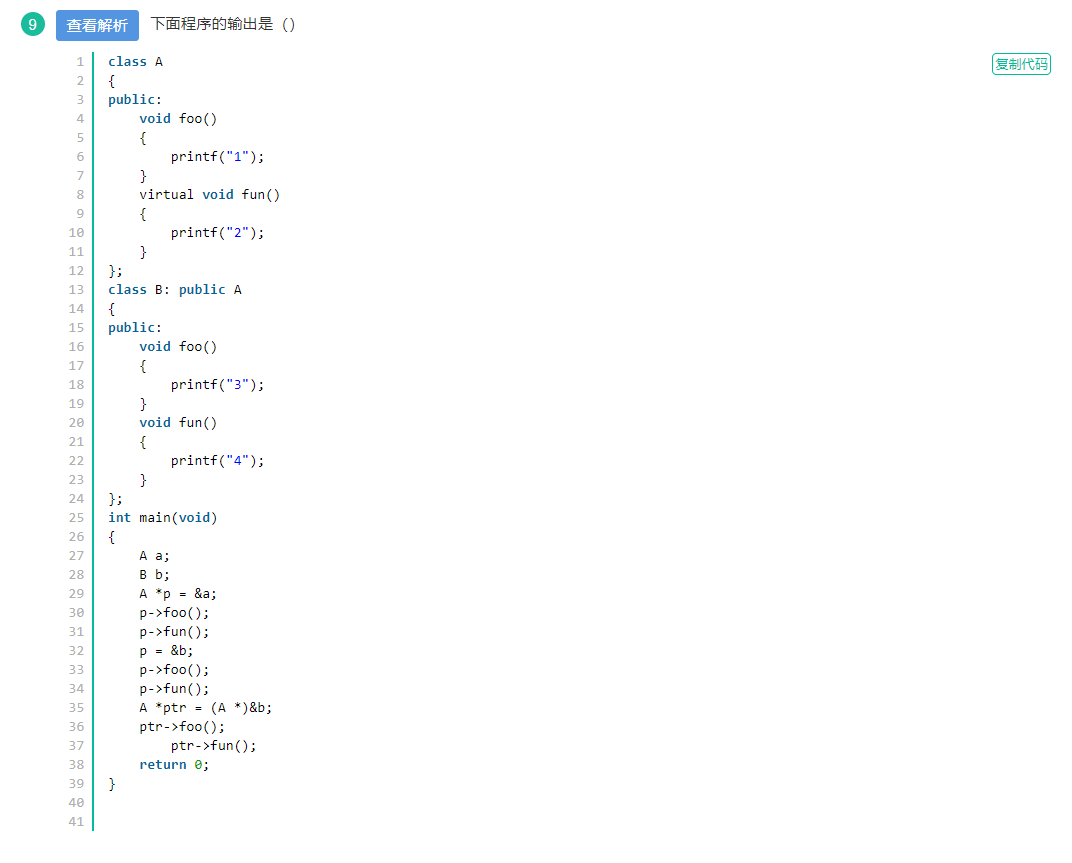


解析：

需要基类的析构函数为虚函数。但是也可以用智能指针。注意A选项不要漏了。



解析：虚函数必须是成员函数。不能是友元函数或者静态函数。同时，构造函数不能是虚函数，但是析构函数可以是虚函数。



Ans: 121414

解析：首先要弄明白**重载，覆盖以及隐藏**的区别。

（1）重载：发生在同一个类中。同名函数具有不同的参数列表（参数的类型，个数以及顺序）

（2）覆盖：发生在基类与派生类中。**函数名，参数列**都必须与父类相同，同时必须有virtual关键字。[覆盖相当于把函数指针指向的函数替换了]。注意：返回类型可以不一样。

（3）隐藏：发生在基类与派生类中。基类与派生类存在同名函数，不论参数列表是否相同。

即使知道了有隐藏这回事，但是不知道隐藏背后发生了什么。**隐藏机制触发之后，指针的调用仅取决于指针的类型**！

例如：假设A中为void foo(float a)，B中为void foo(int a): 做如下调用:

B \*pb=&b; pb->foo(3.14);

因为触发了隐藏机制，基类的void foo(float)会被隐藏，所以这里调用的仍然是派生类中的void foo(int)

总结:

1.判断要点:**如果不是重载也不是覆盖，派生类和基类中一旦出现同名函数，一定触发隐藏机制**(这是个简便判断技巧，你可以考虑除去重载和覆盖的任何同名函数情况，一定满足隐藏机制触发的两条规则)。

2.**隐藏触发的结果:指针对成员的函数调用取决于指针类型**。

若本身是基类指针(不管指向基类还是派生类)则仍然调用基类成员(不会牵扯到派生类，此处是隐藏，和多态没关系，按第1点已说明隐藏的触发可以首先排除覆盖，也就是多态问题);

若本身是派生类指针，这时你就会看到隐藏的威力！此时不是简单地继承基类的成员，然后根据参数匹配调用，而是隐藏基类成员，只会调用派生类成员。

3.注意，不涉及指针就不会发生多态行为。



解析：B，需要根据派生类对基类的不同继承方式而分情况讨论。C，this指针只适用于非静态基类保护成员。

**类的继承以及访问权限**：

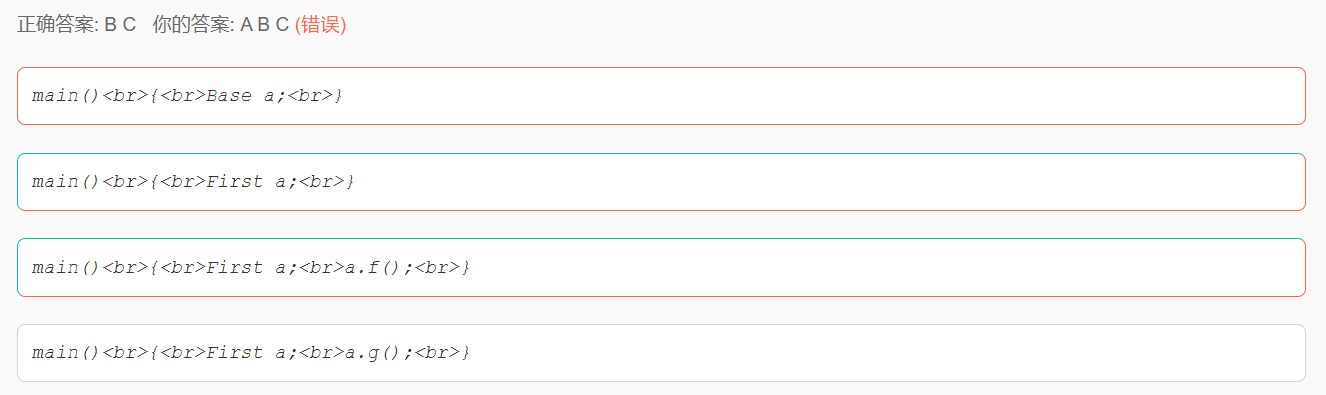
* Private继承：父类protected和public属性在子类中变为private
* Protect继承：父类protected和public属性在子类中变为protected
* Public继承：父类中ptotected和public属性在子类中不变

Private访问权限：该类中的成员函数，友元函数

Protect访问权限：该类中的成员函数，派生类中的成员函数，友元函数

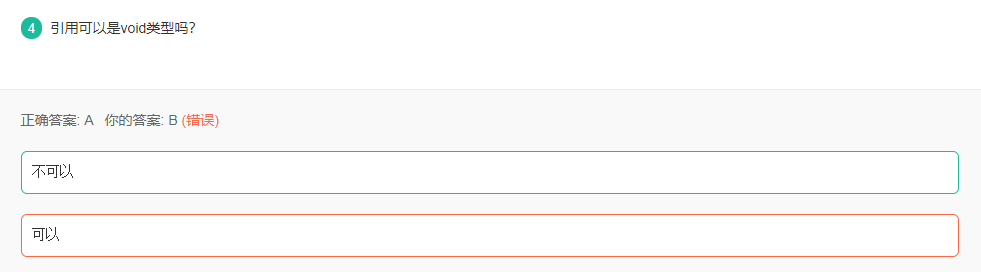
Public访问权限：该类中的成员函数，派生类中的成员函数，友元函数，该类的对象





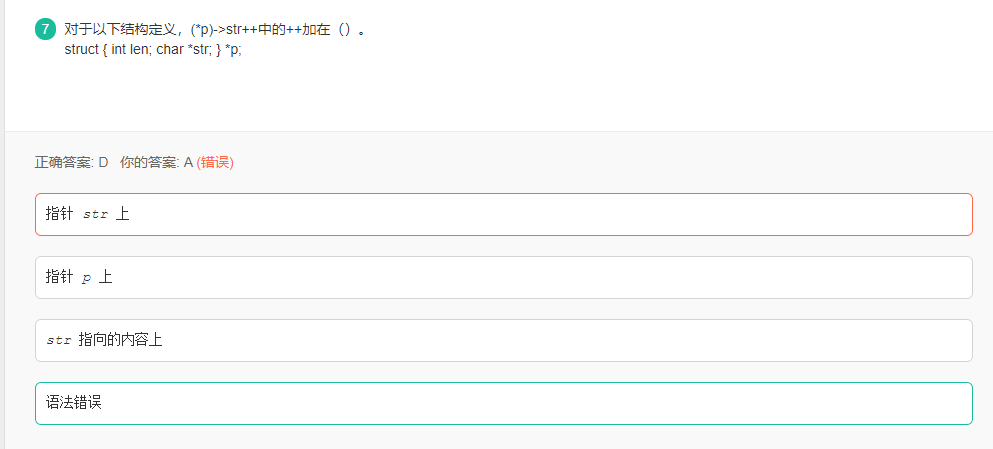
解析：注意Base构造函数的访问权限是protected，说明在类外除继承方式，无法实例化对象。

## 指针，引用



解析：

引用只是一个别名，是已有变量的别名。Void类型没有分配内存，而引用必须是另一个固定内存变量的别名，所以引用不能是void类型。同时注意，引用必须初始化。



解析：

p是指针，可以写成p->str或(\*p).str，而(\*p)->str会造成语法错误。



解析：

注意，pa改变了a[0]的值之后，改变了指向。

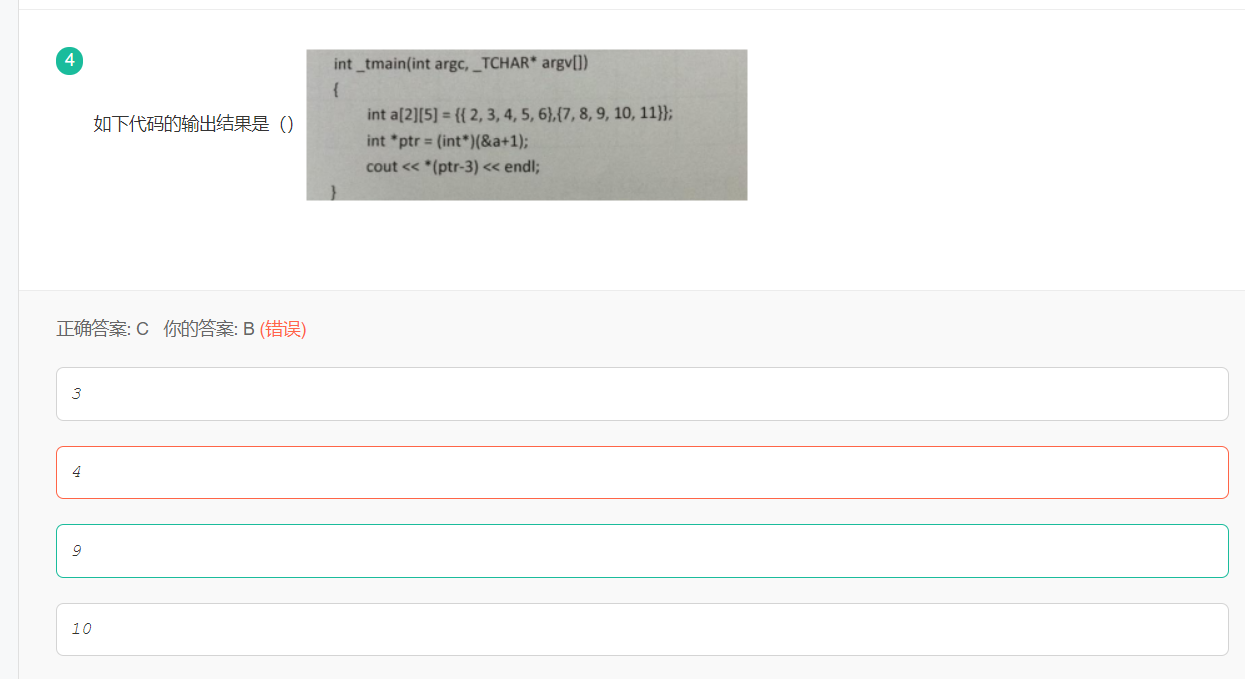


解析：

a+1才表示数据首元素地址+1，即指到6的位置。而&a+1的偏移量是整个数组的长度，指到了10后一位。然后往前移动两个单位，就到了9。



解析：()是初始化，[ ]是定义多个对象。所以p仅仅是指向一个int值。



解析：对于数组名来说，\*是往里解析一层，&是向外增加一层。所以&a+1，这里的1加的是整个数组的跨度。再减3就得到了9。

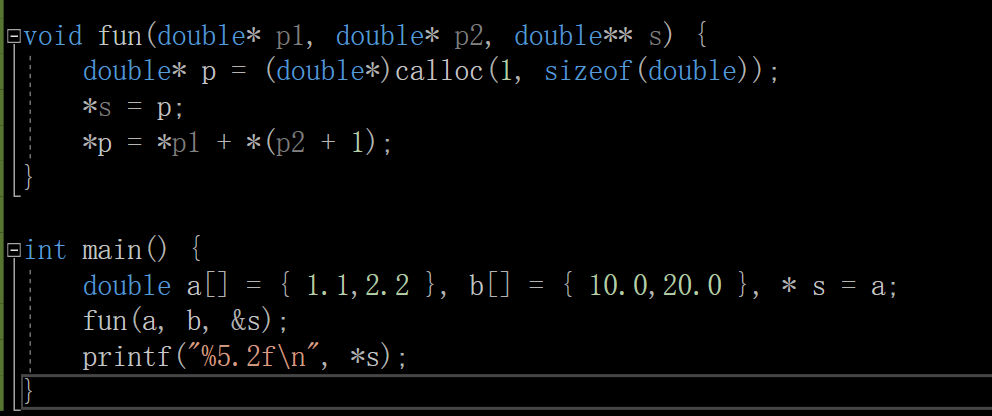


解析：1.10

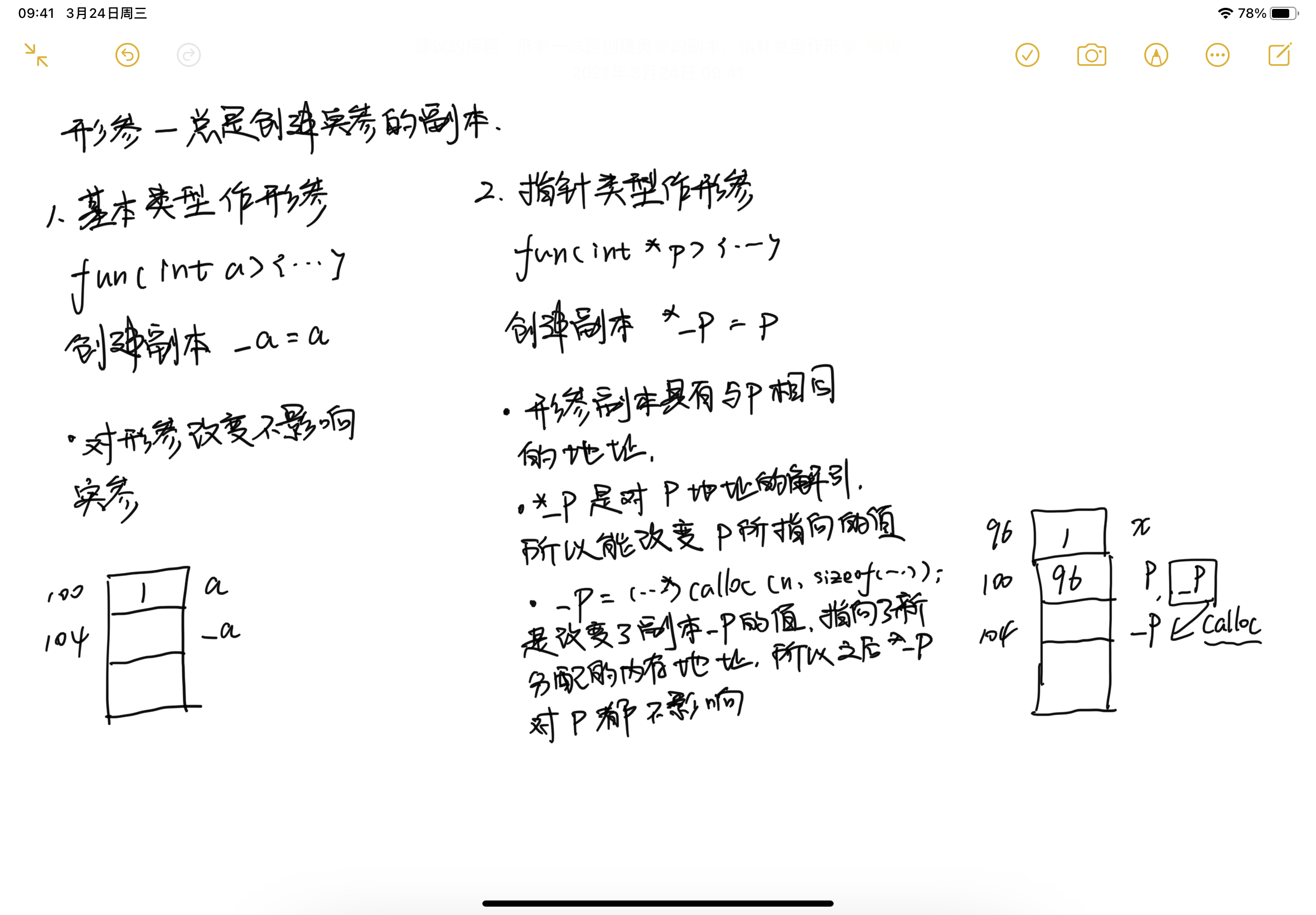
关键：要理解形参以及指针的意思（指针是传值的）

形参为基本数据类型，那么就是创建副本，形参为指针，也是创建一个副本，副本的值就是指针的地址，所以在函数中对形参解引能够改变main中指针所指向的值。但是fun中的calloc，其实是对副本指针\_s重新分配地址，然后在重新分配的地址上赋值。这样因为之前并没有解引\_s，所以对main中的s并不影响。

如果要在fun中使用calloc改变\*s的值，那么需要用到二级指针，改变s的指向，使其指向新分配的内存。



关于形参



## 四，字符串



解析：注意字符串常量不能直接对数组名进行赋值，但是可以初始化。D中第二个直接赋值也是不行的。



解析：八进制表示，区别sizeof和strlen

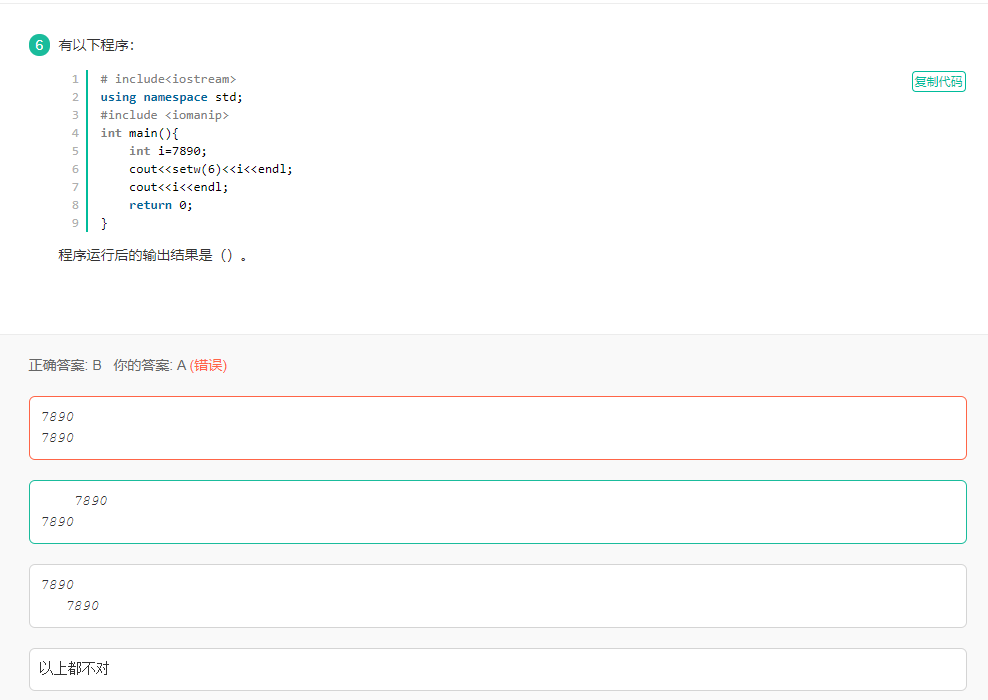
sizeof计算的是变量整体占用内存的大小，包括了字符串末尾的‘\0’。而strlen是遇到结束符就停止。\065表示八进制数。

## 其他



解析：

《effective c++》里面有一模一样的话。。



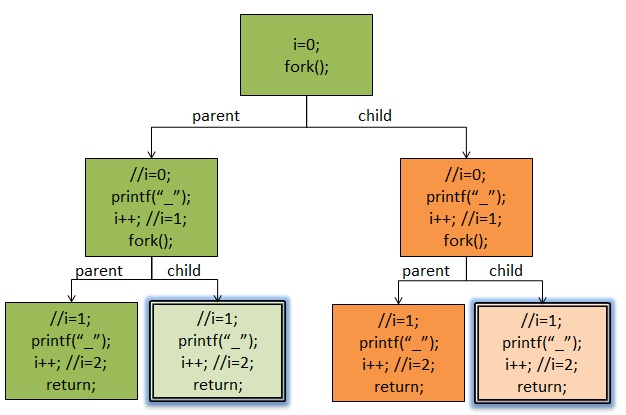
setw(int)作用是根据输入的int值，保证输出的位数，默认**右对齐**（可用setiosflags()修改），并用空格（可用setfill()修改）在左侧填充不足的位数。



解析：

fork()系统调用是Unix下以自身进程创建子进程的系统调用，有如下的特性：

1. 一次调用，两次返回，如果返回为0，则是子进程；如果返回值>0，则是父进程（返回值是子进程的pid）
2. 在fork()调用处，整个父进程空间会原样地复制到子进程中，包括指令，变量值，程序调用栈，环境变量，缓冲区等。



该题的程序一共调用6次printf()方法，但是**由于printf语句有buffe**r但又没有”\n”，所以不会刷新缓冲区，**子进程又复制了父进程的输出缓冲区**，所以每个子进程都会输出2个“-”。所以最终输出了8个”-”



解析：

方法重载的定义：同一个类或与他的派生类中，方法名相同，而参数列表不同的方法。其中参数列表不同指的是参数的类型，数量，类型的顺序这三种至少有一种不同。

\*方法重载与下列无关：

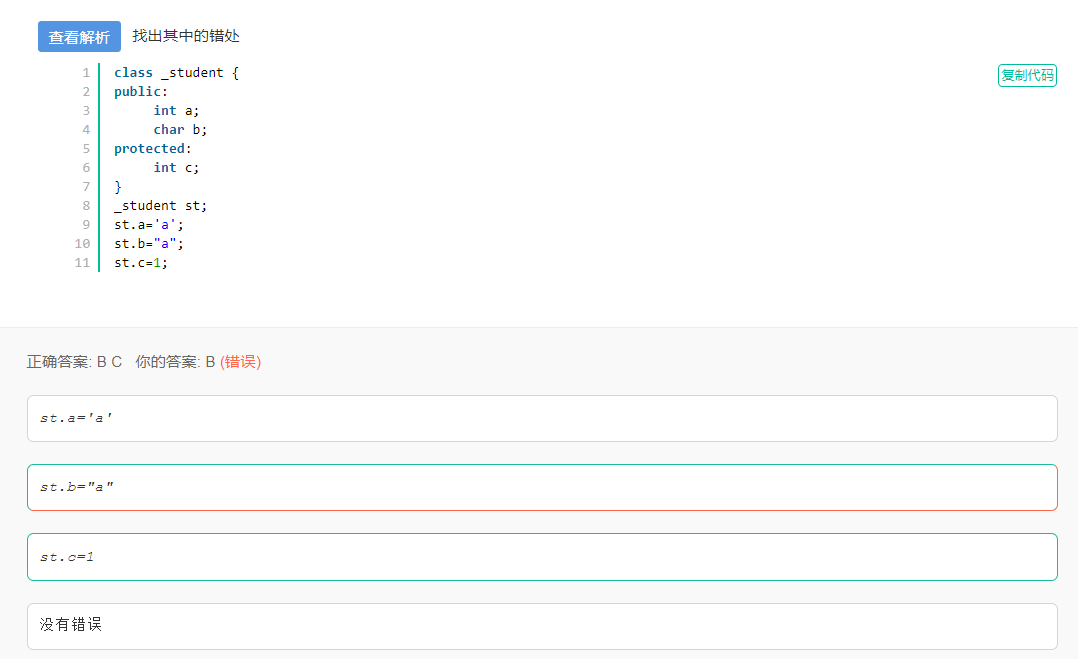
与返回值类型无关；与访问修饰符无关

注意：构造方法也可以重载



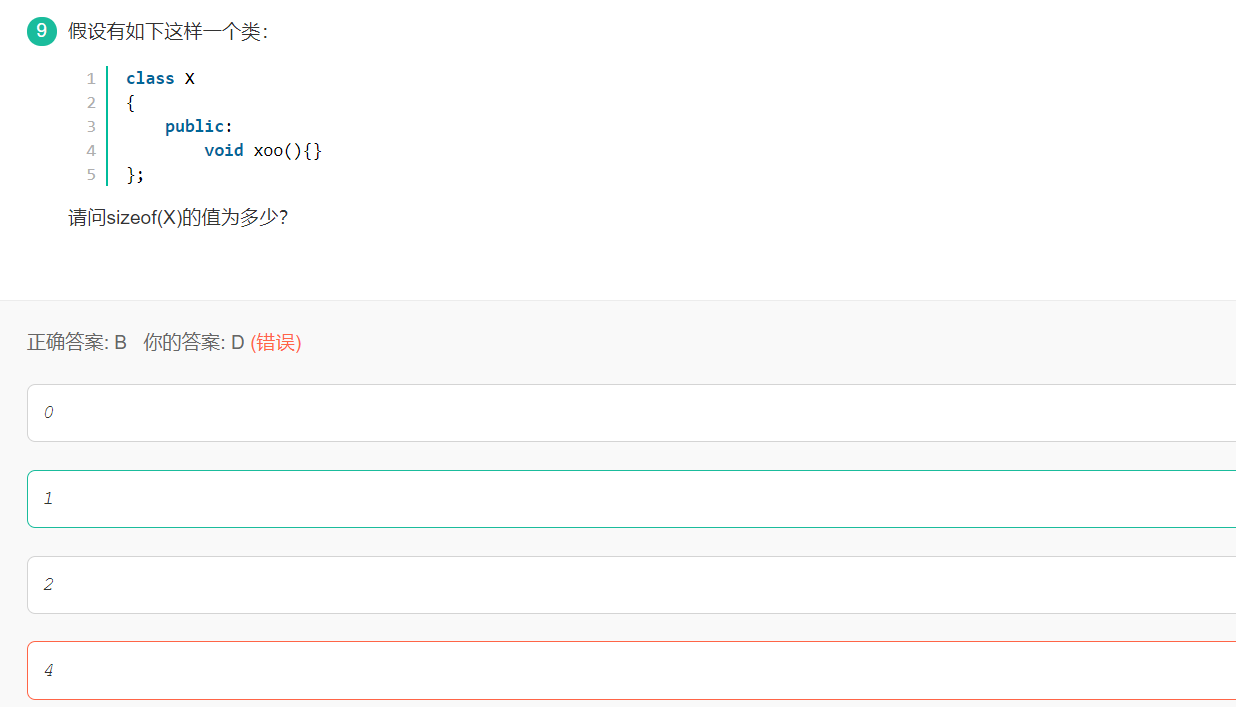
解析：

逻辑运算符两侧运算对象的值如果是0，则表示假，非0就表示真，而不管类型是什么。



注意：

protected成员不能直接在外部访问。



解析：

C++标准规定类的大小不为0，空类（没有非静态数据成员，没有虚函数也没有虚基类）的大小为1

**类的sizeof大小**一般是类中的所有成员的sizeof大小之和，这个就不用多说。

不过有几点需要注意：

1）类大小遵循结构体的对齐原则

2）当类中含有虚成员函数的时候，在类中隐藏了一个指针，该指针指向虚函数表。

3）当类中没有任何成员变量，也没有虚函数的时候，该类的大小为1

4）类大小与普通数据成员有关，与普通成员函数，静态成员函数，静态数据成员无关



解析：24。

32位机器表示4字节对齐。那么1-8分配给\_x1，9分配给\_x2，13-16分配给\_x3，17-18分给\_x4，静态变量\_x5不需要管。但是，结构体的总大小需要为最大成员大小的整数倍。所以\_x4将占17-24。总大小为24字节。

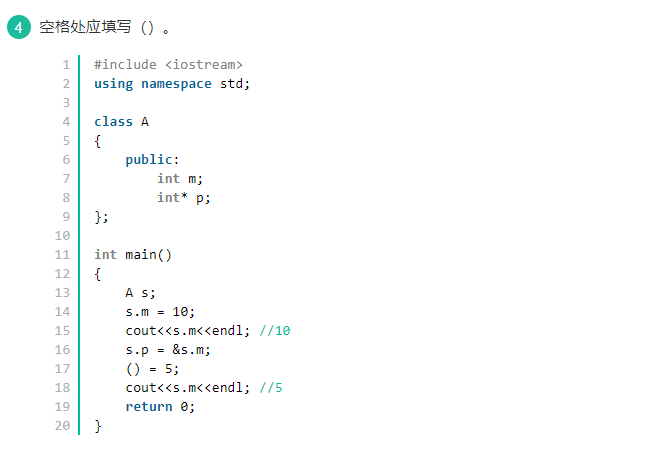


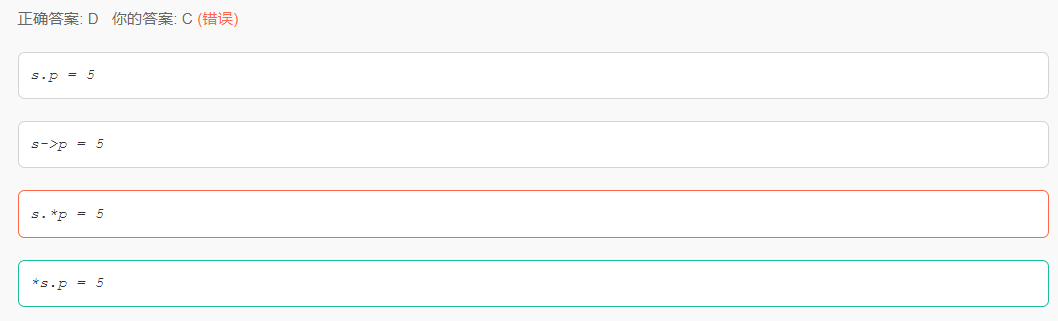
解析：1，1，4，4，4，4

根据上一题其实可以得出前四个，最后nLenC，nLenCObject为4是因为：

对于普通继承，派生类和基类共享虚函数指针，

派生类对象的存储空间=基类存储空间+派生类特有的非static数据成员的空间。由于t\_classA为空类，t\_classB和t\_classC共享虚函数指针，因此nLenC和nLencCObject的值为4.





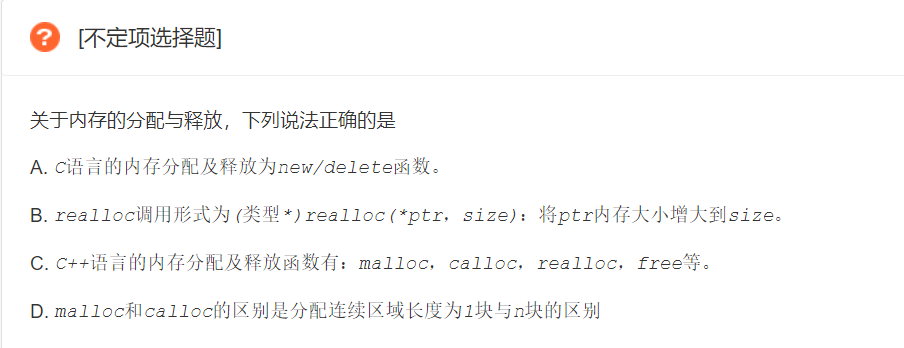
解析：

.的优先级比\*高



解析：

重载：函数名相同，且参数列表可区分。



解析：B D

c语言的标准内存分配函数有：malloc,calloc,realloc,free等。

c++的内存分配运算符为new/delete，注意new/delete是运算符，不是函数。



解析： 答案是11

变量初始化的顺序是其声明的顺序，跟初始化列表中的顺序无关。



解析：

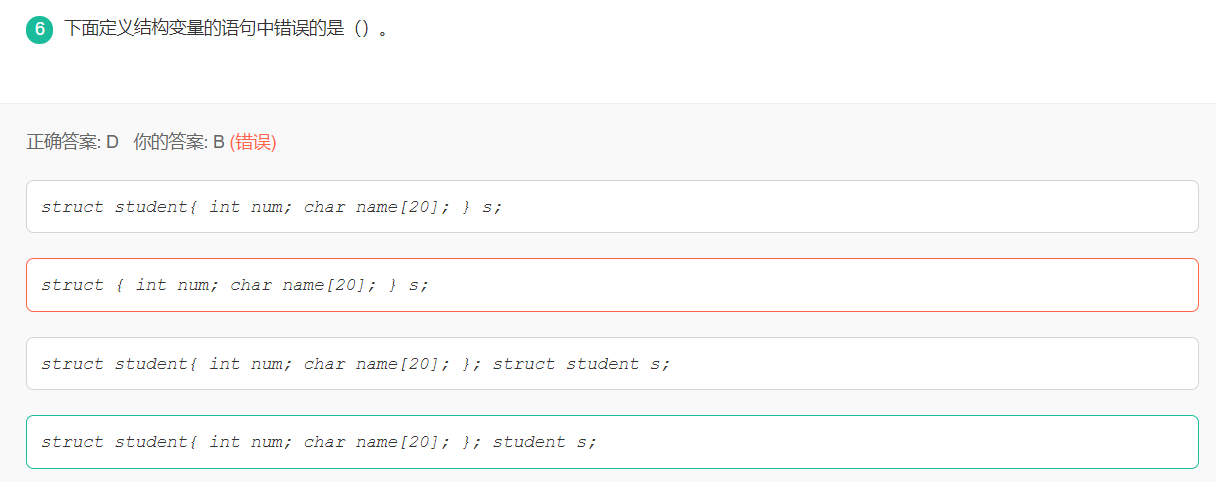
关于迭代器失效的情况：<https://www.cnblogs.com/fnlingnzb-learner/p/9300073.html>

（1）序列式容器（如vector, deque），序列式容器就是数组式容器。**删除当前的iterator会使后面所有元素的iterator都失效**。这是因为vetor,deque使用了连续分配的内存，删除一个元素导致后面所有的元素会向前移动一个位置。所以不能使用erase(iter++)的方式，还好**erase方法可以返回下一个有效的iterator**。【但是注意，deque是size改变后，所有元素迭代器都失效】

（2）对于关联容器(如map, set,multimap,multiset)，删除当前的iterator，**仅仅会使当前的iterator失效**，只要在erase时，递增当前iterator即可。这是因为map之类的容器，使用了红黑树来实现，插入、删除一个结点不会对其他结点造成影响。erase迭代器只是被删元素的迭代器失效，但是**返回值为void**，所以要采用erase(iter++)的方式删除迭代器。

（3）对于链表式容器(如list)，删除当前的iterator，**仅仅会使当前的iterator失效**，这是因为list之类的容器，使用了链表来实现，插入、删除一个结点不会对其他结点造成影响。只要在erase时，递增当前iterator即可，并且erase方法可以返回下一个有效的iterator。

//自己还没弄太明白orz



解析：D

结构体定义的三种方式：

1. 先构造，后定义
2. 构造的同时定义
3. 构造同时定义，省略类型名，这样就不能定义新变量。

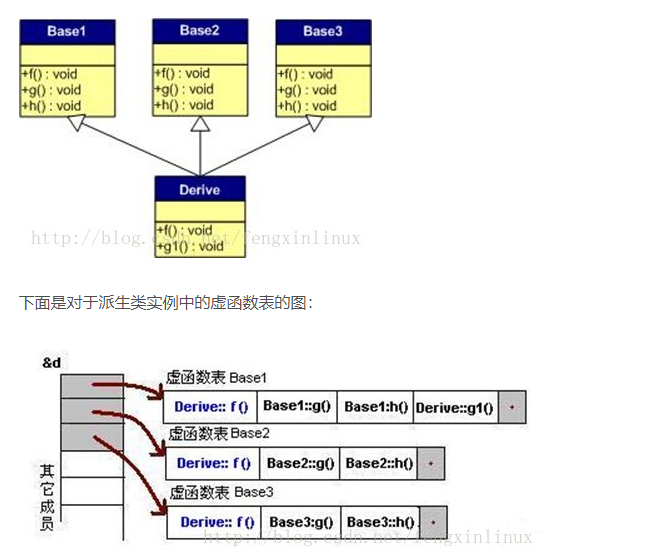
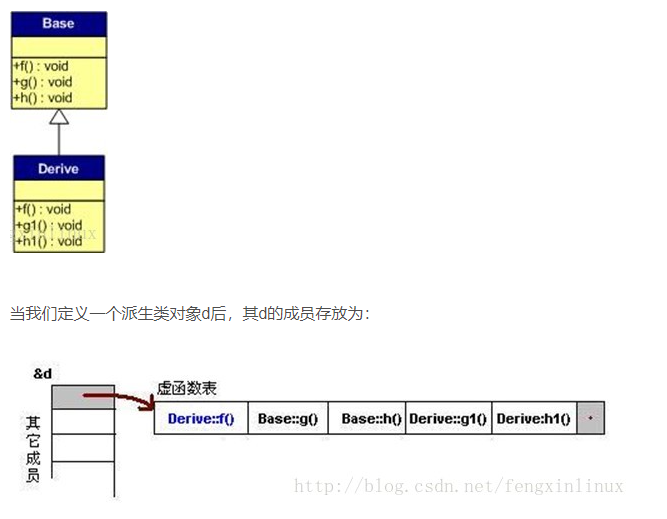
这题中D选项在C语言中不支持。

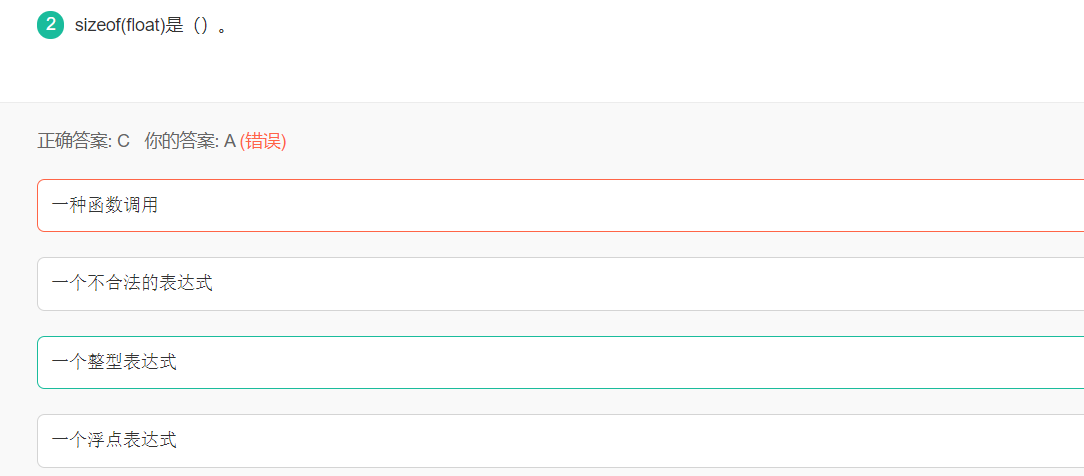
**关于虚函数的讲解**：https://blog.csdn.net/fengxinlinux/article/details/72836199

虚函数（Virtual Function）是通过一张虚函数表（Virtual Table）来实现的。编译器必需要保证虚函数表的指针存在于对象实例中最前面的位置（这是为了保证正确取到虚函数的偏移量）。

每当创建一个包含有虚函数的类或从包含有虚函数的类派生一个类时，编译器就会为这个类创建一个虚函数表（VTABLE）保存该类所有虚函数的地址，其实这个VTABLE的作用就是保存自己类中所有虚函数的地址，可以把VTABLE形象地看成一个函数指针数组，这个数组的每个元素存放的就是虚函数的地址。在每个带有虚函数的类 中，编译器秘密地置入一指针，称为v p o i n t e r（缩写为V P T R），指向这个对象的V TA B L E。 当构造该派生类对象时，其成员VPTR被初始化指向该派生类的VTABLE。所以可以认为VTABLE是该类的所有对象共有的，在定义该类时被初始化；而VPTR则是每个类对象都有独立一份的，且在该类对象被构造时被初始化。

单继承时，子类，基类共享虚函数表。而多继承时，派生类有多少个虚基类就有多少个虚函数表的指针。





解析：

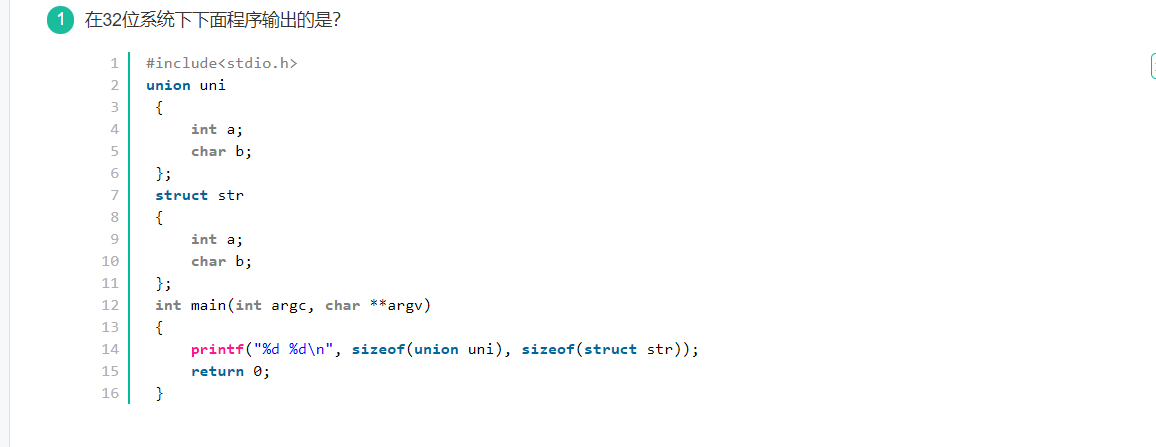
sizeof是操作符，在编译阶段就得到结果，并替换成一个整数。



解析：

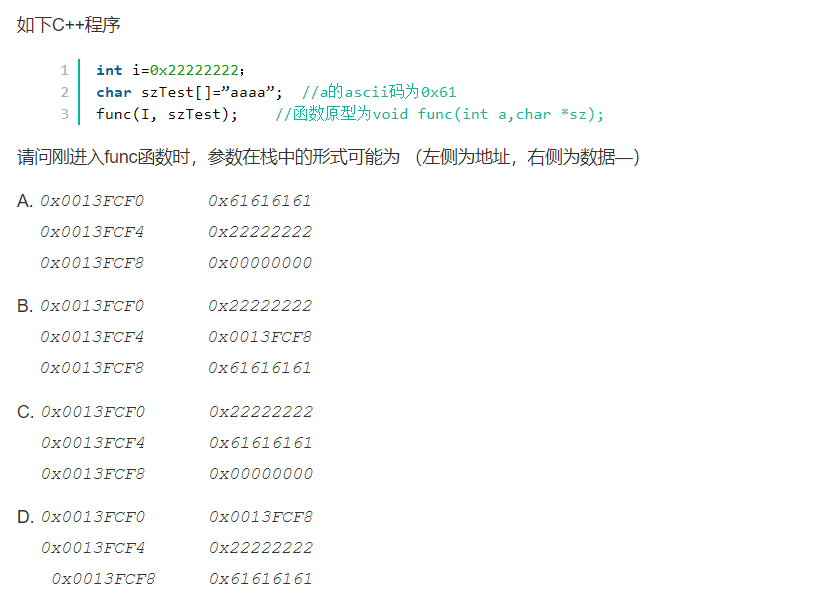
B选项，std::sort先采用快排，若递归深度过大，则转用堆排，与选取中值无关。

D选项，std::vector可以调用shrink\_to\_fit()归还多余空间，所以错误



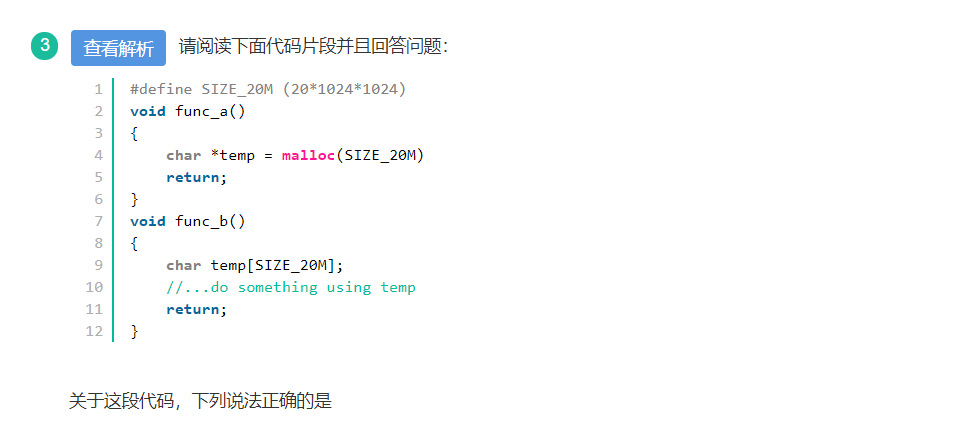
解析：4 8

Union的大小**等于**其所占空间最大的成员的大小。这些成员存放在同一地址开始的内存单元中。也就是使用的**覆盖**技术，这多个变量互相覆盖，而这种使几个不同的变量共占同一段内存的结构。



解析：D

注意：（1）栈从内存高端向低端生长；（2）函数调用时入栈顺序：函数返回地址->参数（从右到左入栈）。D中第一行是函数开始地址，不是返回地址。



解析：注意区分两段程序的内存分配是在哪个空间进行的。

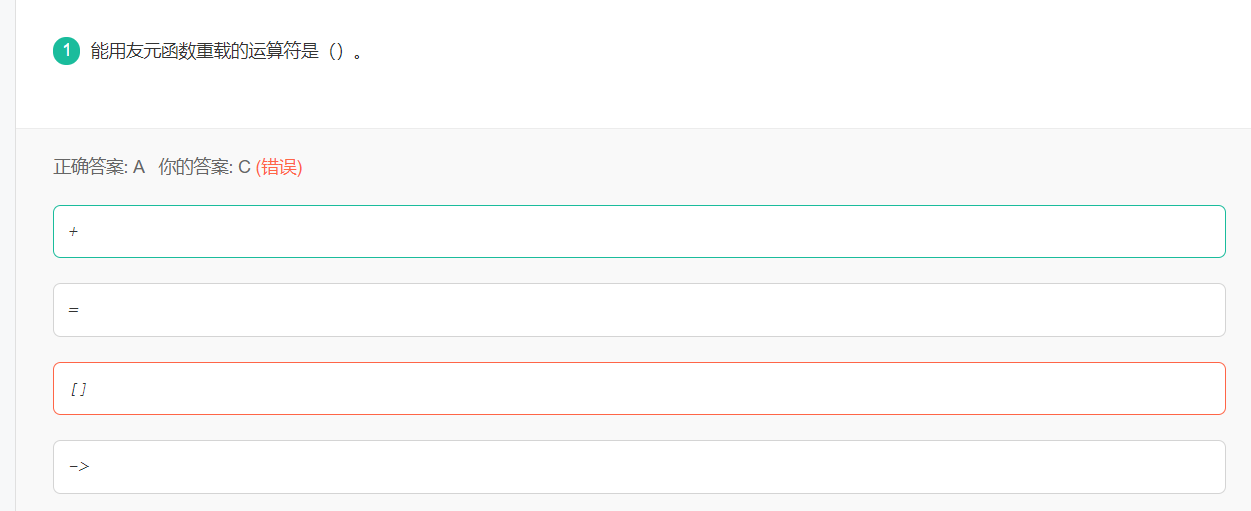
1. func\_a是在堆上分配内存空间，效率低于func\_b在栈中分配空间。栈上分配空间效率更高。
2. 栈的大小通常是2M，分配太多栈空间程序会崩溃。
3. func\_b中没有动态分配的空间，不存在内存泄漏问题。
4. 都不会自动初始化为0，只有全局变量和static变量才会初始化为0

详见：<https://blog.csdn.net/baidu_37964071/article/details/81428139>

栈与堆的区别。



解析：c++中除了5个运算符其他的都可以重载，这5个是：（1）类属关系运算符“.”；（2）成员指针运算符“.\*”；（3）作用域运算符“::”；（4）sizeof运算符；（5）三目运算符“?:”。



解析：C++规定=，[ ]，()，->这四个运算符只能被重载为类的非静态成员函数，其他的可以被友元重载，主要是因为其他的运算符重载函数都会根据参数类型或数目进行精确匹配，这四个不具有这种检查的功能，用友元定义就会出错。



解析：如果同时在类中，对于函数名相同的const函数与非const函数能够构成重载。



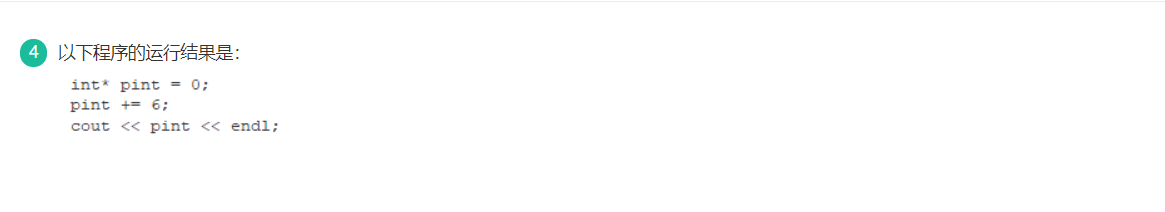
解析：

嵌套的函数指针从返回值往前套。

int (\*f)(int) … ①

f=>(\*fp)(int,int) … ②

把1，2式合并就可以得到结果。



解析：

pint最开始指向地址0，然后加6是加6个int的长度。如果int长度为4，那么pint的地址将为24。



解析：

shared\_ptr的引用计数本身是线程安全的且无锁的，但对象的读写不是，因为 shared\_ptr 有两个数据成员，读写操作不能原子化。 shared\_ptr 的线程安全级别和内建类型、标准库容器、std::string 一样，即：

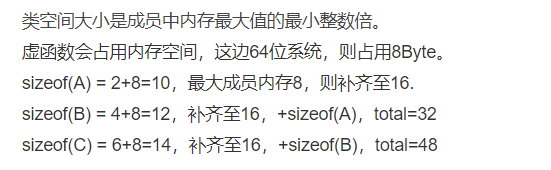
• 一个 shared\_ptr 对象实体可被多个线程同时读取；

• 两个 shared\_ptr 对象实体可以被两个线程同时写入，“析构”算写操作；

• 如果要从多个线程读写同一个 shared\_ptr 对象，那么需要加锁。

请注意，以上是 shared\_ptr 对象本身的线程安全级别，不是它管理的对象的线程安全级别

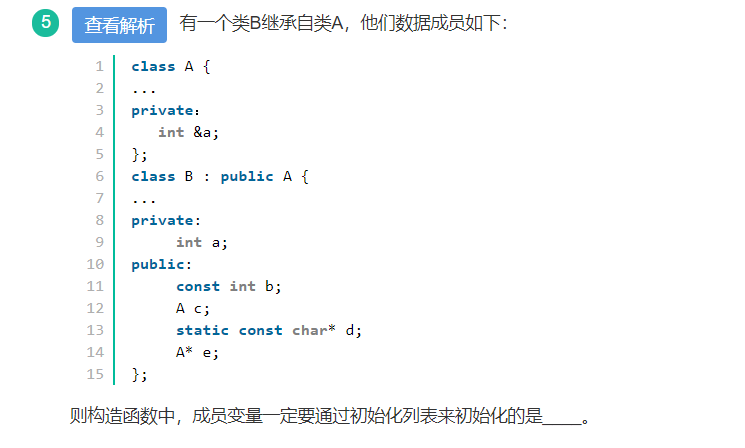
解析：



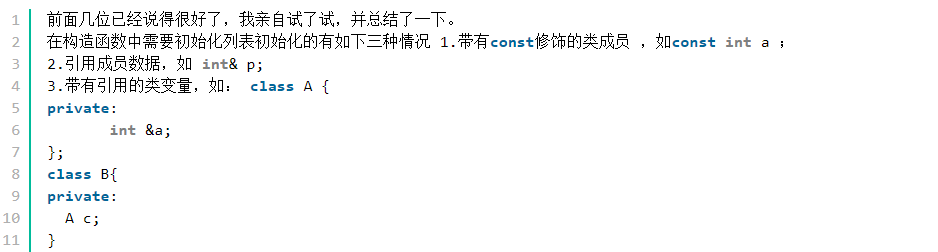


解析：

关于B选项，非const静态数据成员要在类外定义和初始化。而const静态数据成员必须在类内进行初始化。



解析：b，c



关于类成员初始化列表，参考：<https://www.runoob.com/w3cnote/cpp-construct-function-initial-list.html>。

初始化列表与构造函数内进行的区别是：初始化列表是显式的初始化，而构造函数内是对成员的赋值，而不是初始化。

对于内置类型，初始化和赋值没有什么很大的区别。但是对于非内置类型成员变量，为避免两次构造，推荐使用初始化列表来构造。但是有时候，必须使用带初始化列表的构造函数：

1. 成员类型是没有默认构造函数的类。
2. Const成员或引用类型的成员。因为const对象和引用类型只能初始化，不能对它们赋值。

同时，需要注意，初始化列表的初始化顺序是类中成员的声明顺序，而不是初始化列表中的出现顺序。



解析：

B，例如Matrix m; m = new Matrix(); 就涉及到复制构造

C，要保证唯一ID，若不自定义拷贝构造那么将会产生重复ID。同时，拷贝构造函数也不能处理静态成员（如，计数器）

一个简单的原则是：如果你需要一个非空的析构函数，那么通常情况下也需要定义一个拷贝构造函数。

参考：

* <https://blog.csdn.net/weixin_41066529/article/details/89673877>
* <https://blog.csdn.net/lwbeyond/article/details/6202256>



解析：

X86机器是小端存储，即低位字节存储在内存低地址处。依次是：0x34, 0x12, 0xcd …



解析：按照优先级的顺序，应该先计算加法，再进行左移。

参考，运算符优先级：<https://blog.csdn.net/yuliying/article/details/72898132>



解析：输出为short

考点：explicit构造函数的作用。

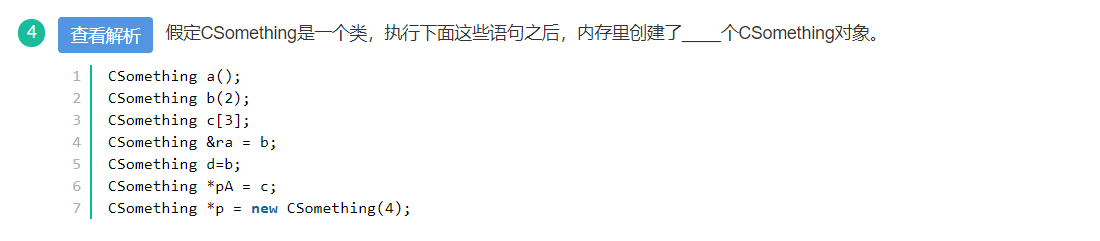
注意：1，接受一个参数的构造函数允许使用赋值语句隐式调用来初始化对象；

2，explicit构造函数只能被显式调用。

Test a = n; 需要有参构造函数。但却并不是显示调用的，所以不能用explicit Test(int)，因为explicit说明必须显示调用。编译器在所有带参构造函数中选择合适的，所以n被转换为short。



解析：#pragma pack(2) 表示强制2字节对齐；typedef说明只是函数指针的别名，不是自己的指针，不需要计算大小。枚举类型实例color占4字节。所以最终是4+14+4=22。

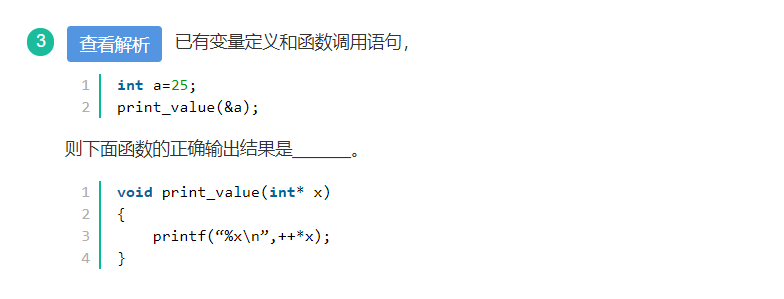


解析：6个。

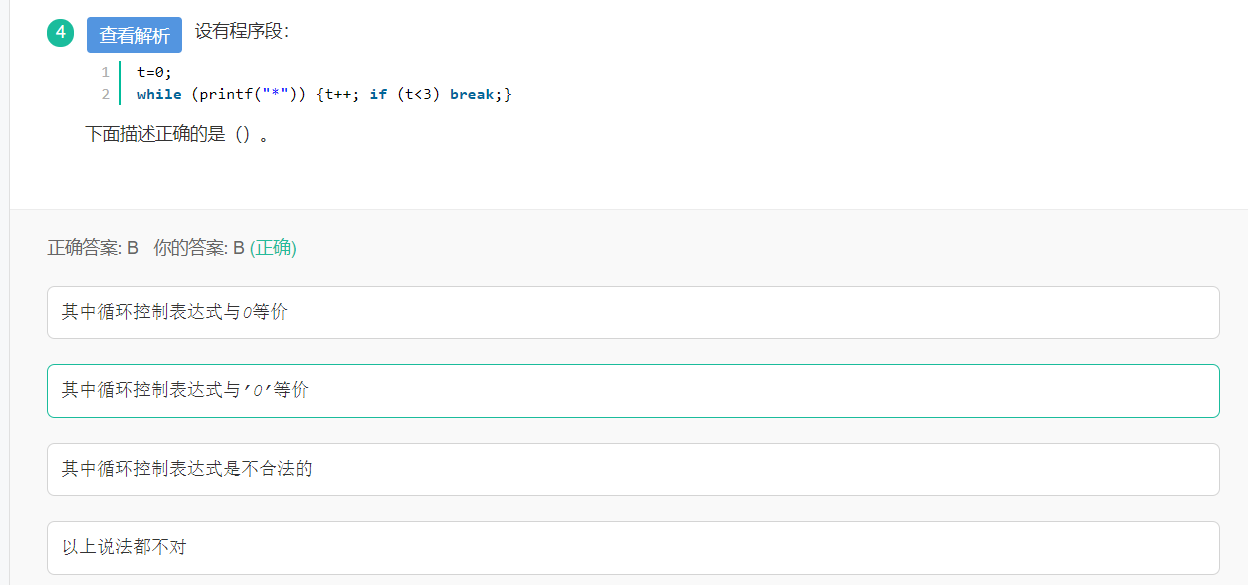
注意:

CSomething a(); 是一个函数的声明。 CSomething a; 与 CSomething a(0)；才是构造。

然后c[3] 对象数组。



解析：1a。 %x表示以十六进制输出。



解析：printf具有返回值。返回值为输出内容的长度。



解析：区分动态联编和静态联编。

如果是通过对象名调用虚函数（如，point.shapeName()），在编译阶段就能确定调用的是哪一个类的虚函数，所以属于静态联编。

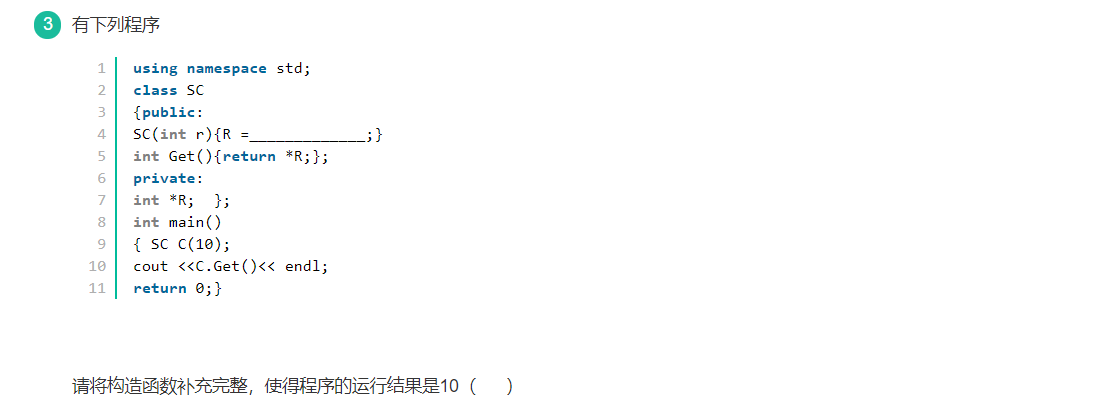
如果是通过基类指针调用虚函数（如，pt->shapeName()），在编译阶段无法从语句本身确定调用的是哪一个类的虚函数，只有在运行时，pt指向某一类对象后，才能确定调用的是哪一个类的虚函数，故为动态联编。

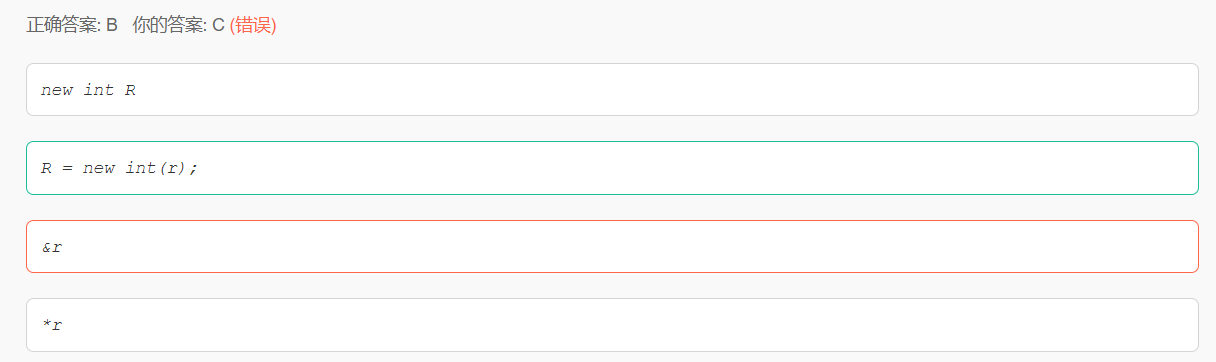


解析：<https://www.nowcoder.com/test/question/done?tid=41694760&qid=36607#summary>

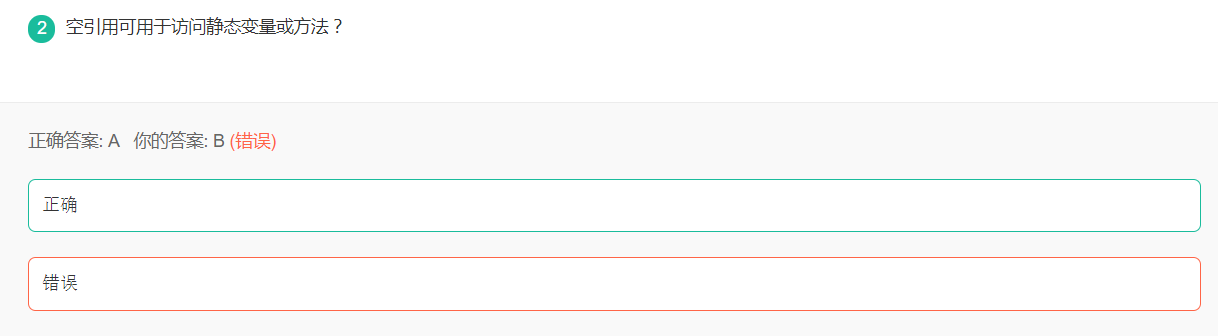
char data[0]; 是C99标准中加入的柔性数组。表示申请一个长度为0的数组（只能放在结构体最后面），使得这个结构体是可变长的。对于编译器来说，长度为0的数组并不占用空间，因为数组名只是一个偏移量，表示一个不可修改的地址常量，但是对于这个数组的大小，可以动态分配。

char data[1]; 也可以表示柔性数组，但是是占用sizeof(char)\*1的空间的。





解析：r是形参，临时变量，生命周期结束后r被释放，其内存地址不安全，其值也不确定。



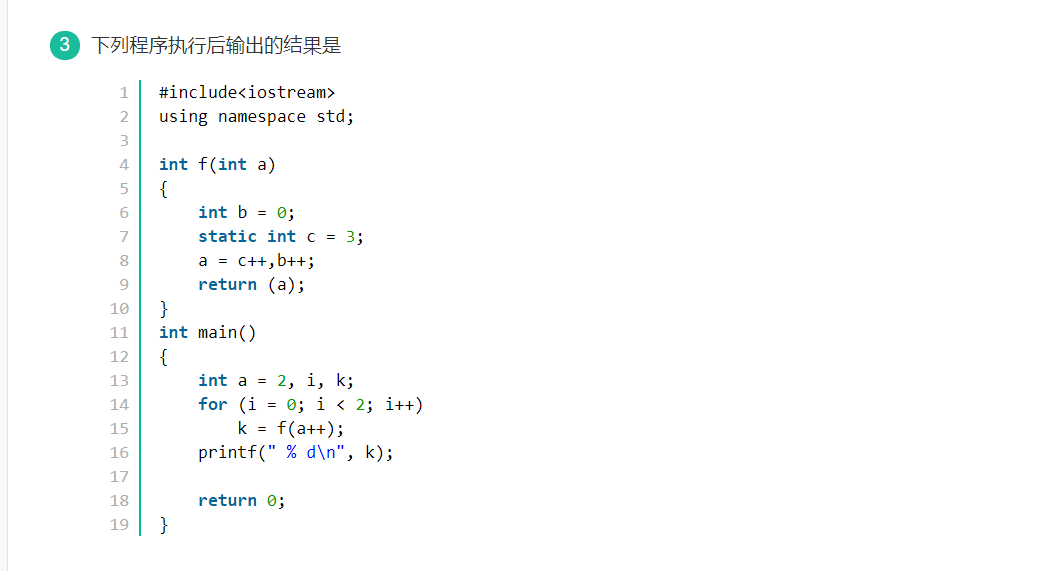
解析：

很多情况下，“空引用”确实可以工作，以致于“引用不能为空”的忠告，被嘲笑为形式主义，仅仅是标准制定者的耸人听闻。

所引用对象是否为null是无关紧要的，因为访问静态方法不需要实例对象。

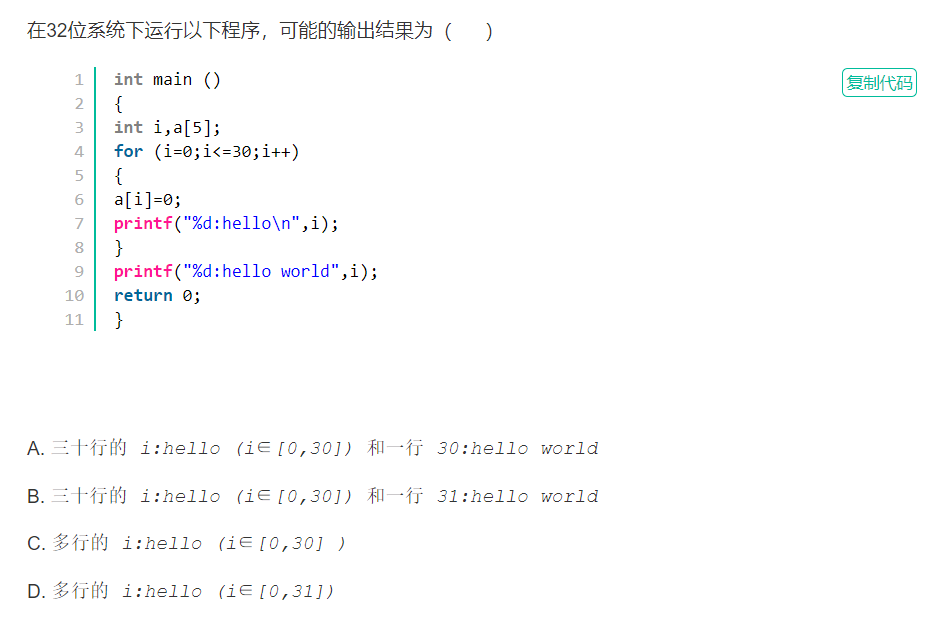


解析：判断表示的是什么还是知道，但是这里又是错在了对优先级的理解。题目中int \*ptr[4]。优先级应该是[]大于\*，那么读过来就是 ptr is an array of 4 pointers to int. 对应就是D选项。



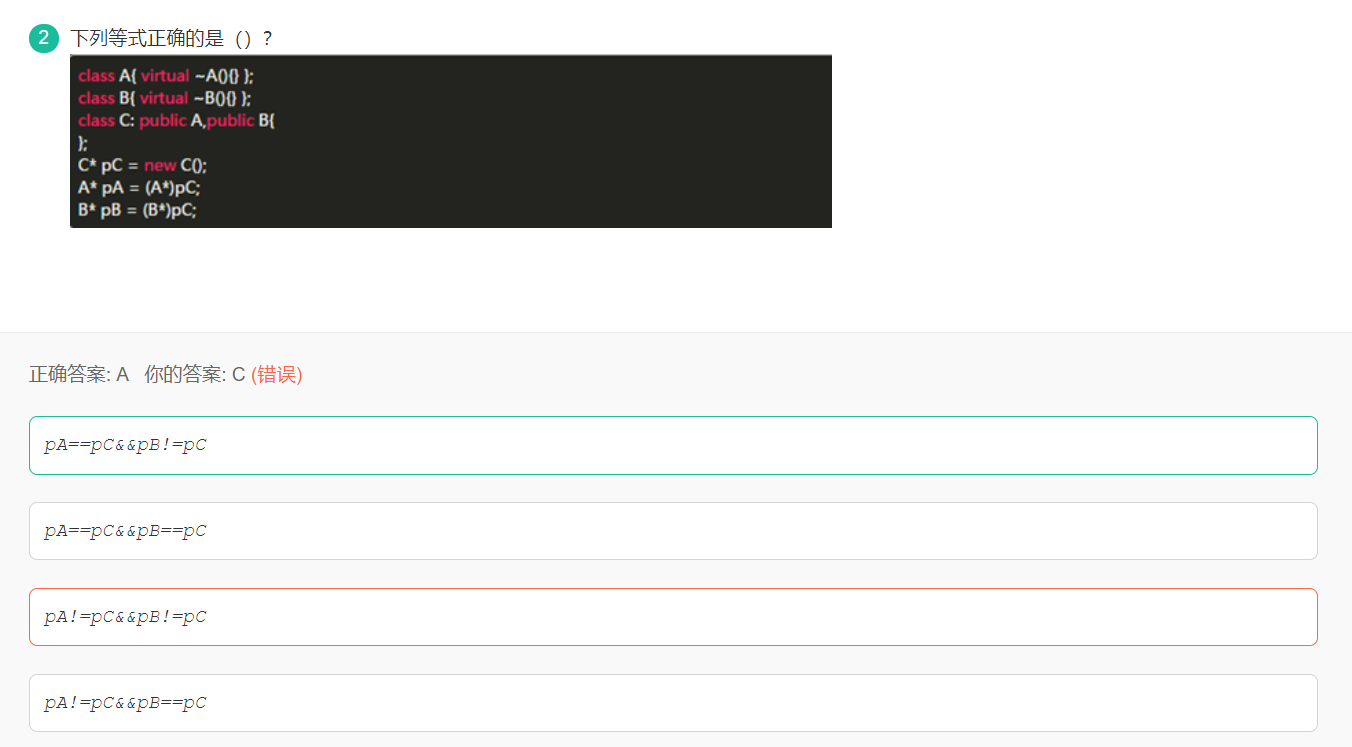
解析：答案为4。

逗号运算符优先级最低，且从左到右进行运算。同时注意c为static局部变量，退出函数时作用域结束，但是生命周期并没有结束。



解析：栈中是从高地址指向低地址，也就是先为高地址分配空间。i的位置与a[6]重合，当for循环到a[6]时，赋值a[6]=0，也会把i也赋值为0，重新循环，导致死循环。

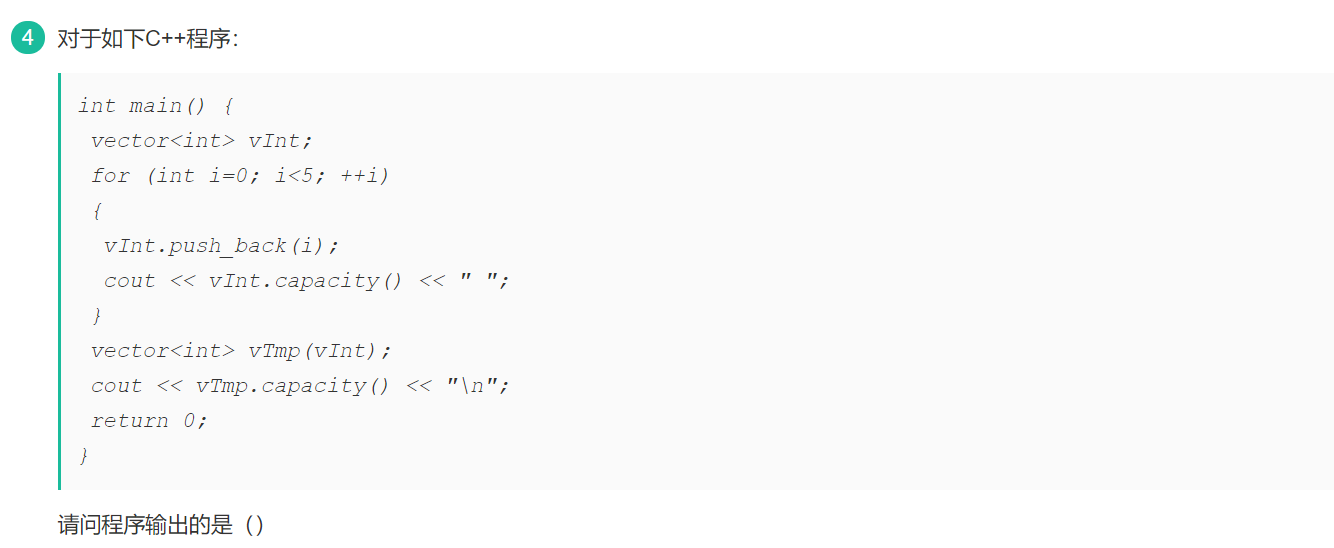
如果将i，a[5]定义顺序改变就不会导致死循环了。



解析：

在多继承中，派生类和被继承的第一个基类的地址相同，和第二个基类的地址不同

见：<https://www.nowcoder.com/test/question/done?tid=42552548&qid=112874#summary>



解析：1 2 4 4 8 5

Vector自动扩容，通常是当不能容纳新元素时，扩容为原来的2倍。但是初始化时size与capacity相等·。

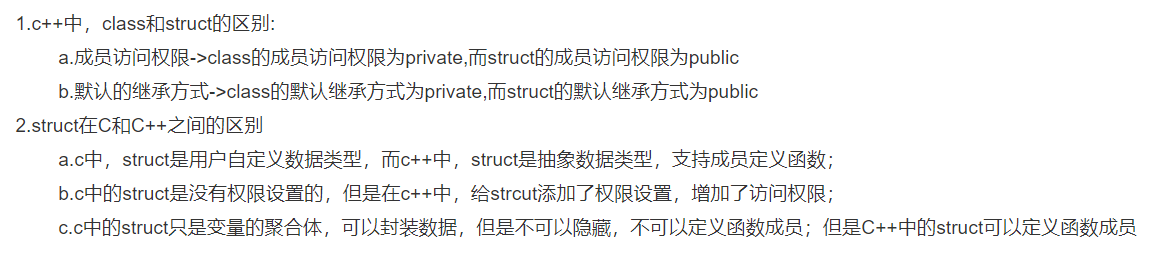
在构造函数不要调用虚函数。在基类构造的时候，虚函数是非虚，不会走到派生类中，既是采用的静态绑定。

在析构函数中也不要调用虚函数。在析构的时候会首先调用子类的析构函数，析构掉对象中的子类部分，然后在调用基类的析构函数析构基类部分，如果在基类的析构函数里面调用虚函数，会导致其调用已经析构了的子类对象里面的函数，这是非常危险的。

<https://www.nowcoder.com/test/question/done?tid=42704343&qid=7621#summary>

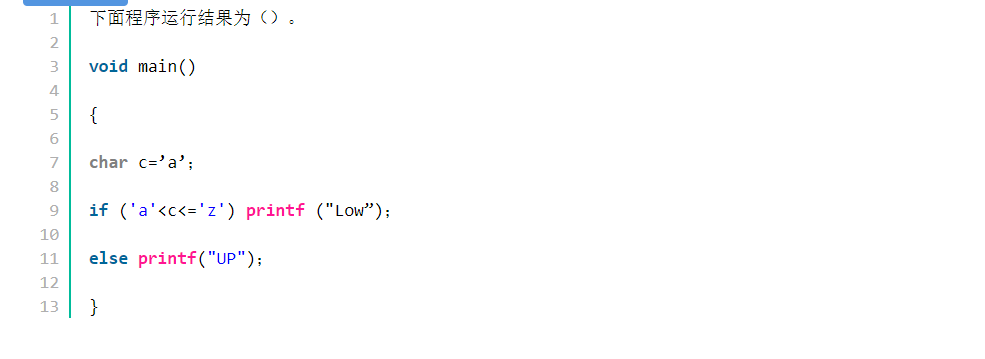


解析：



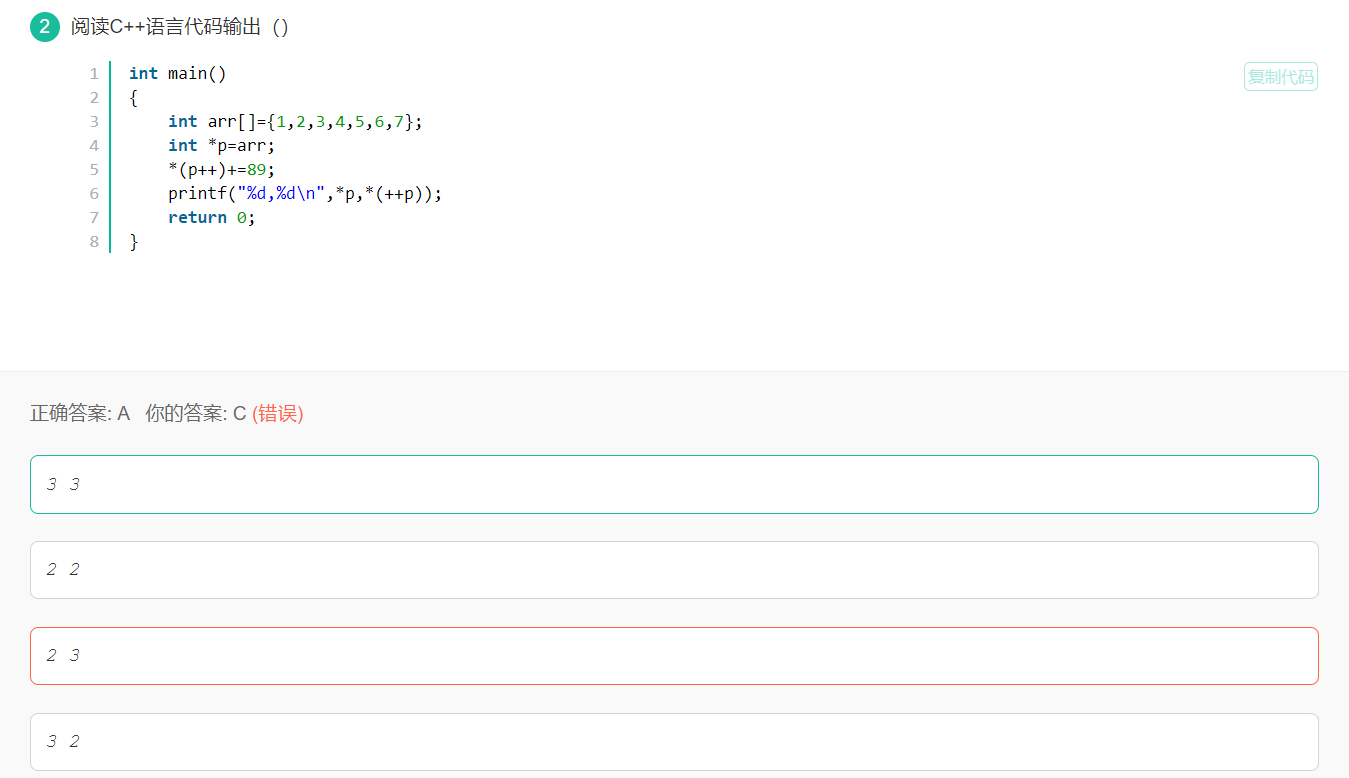


解析：shape中有纯虚函数，是抽象类，抽象类不能实例化。而虚函数只有在指针或引用的情况下起作用。



解析：LOW

关系运算符，< <= 是左结合的。注意，只有单目运算符，位运算符，三目运算符是右结合，其余都是左结合运算。



解析：printf从右到左编译，从左到右输出，符合栈模型。