最近在看《剑指offer》，过程中遇到了一些问题和注意点，所以接下来的几期我会将这些内容记录下来。今天是第一期.

本期关键词：摘录面试软知识，c++ const, 函数引用传递和指针传递。

面试软知识（第一章）

面试官说：

初级程序员: 算法和数据结构； 高级程序员：专业技能和项目经验

对公司的近况，项目情况要有所了解；准备好问题问面试官。

面试三种形式：

①电话面试

②共享桌面面试

* 思考清楚开始编码
* 良好代码命名和缩进对齐习惯
* 进行单元测试 （对定义的函数）
* 调试能力（断点，单步跟踪，查看内存，分析调用栈）

③现场面试

* 准备几个问题问面试官

面试三个环节：

①行为面试

* 项目经验（参与、负责； 了解、熟悉、精通）

着重突出自己的贡献：

系统设计：介绍系统架构的特点；

软件开发：基于什么工具在哪个平台应用了哪些技术

软件测试： 手工测试还是自动化测试；白盒测试还是黑盒测试（注1）

* 掌握的技能
* 为什么“跳槽”

②技术面试

* 扎实的基础知识（编程语言，数据结构，算法）
* 高质量的代码（鲁棒稳定，边界条件，特殊输入）

eg.写一段代码将字符串转换成整数

需要考虑哪些坑？先想一想，往下滑会有参考

。

。

。

。

。

。

。

。

。

。

。

。

。

字符串中有非数字字符和正负号

最大正整数和最小负整数的以及溢出，

字符串不能转换成数字时错误处理？

传入空指针?

* 清晰的思路

二叉搜索树转换成排序的双向链表

* 优化效率的能力
* 优秀综合能力

迁移能力（什么是丑数？计算第1500个丑数）

③应聘者提问环节

* 问与应聘者职位和项目相关的问题

注1：<https://blog.csdn.net/ningmeng2010/article/details/8114530>

说了这么多，不去实践也只是纸上谈兵，所以鼓励大家多去走一走流程，相信一定能比坐在这里空想，学到的更多。



问题：

1. p8 c++ 成员变量的初始化顺序

如下程序：



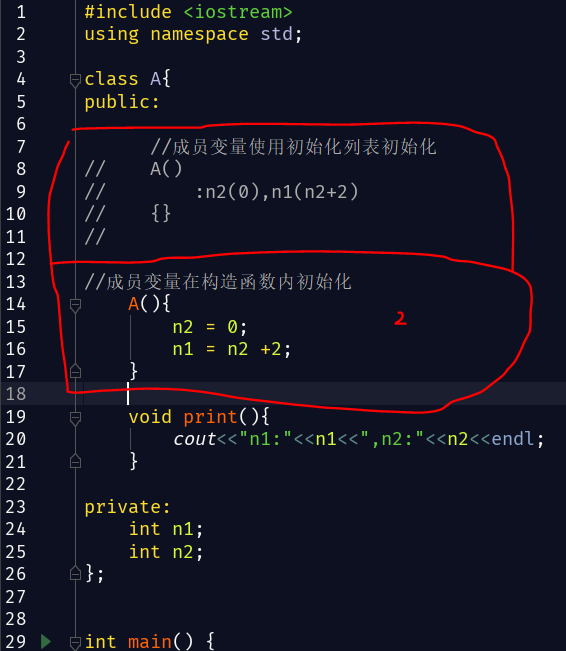
你以为：n1:2  ,  n2:0  

实际上：



成员变量（n1,n2）使用初始化列表初始化时，与构造函数中初始化成员列表的顺序无关，只与定义成员变量的顺序有关。

下面成员变量在构造函数内部初始化，不使用初始化列表初始化了



结果：

n1:2  ,  n2:0

如果不使用初始化列表初始化，在构造函数内初始化时，此时与成员变量在构造函数中的位置有关。

（为什么我的IDE两次结果都是一样的，（小声。。））

另： 类的const成员常量必须在初始化列表中初始化。

类的static成员变量，不能在类中初始化。（ **静态成员**是类所有的对象的**共享**的成员，而不是某个对象的成员。它在对象中不占用存储空间，这个属性为整个类所共有，**不属于任何一个具体对象**。所以静态成员不能在类的内部初始化，比如声明一个学生类，其中一个成员为学生总数，则这个变量就应当声明为静态变量，应该根据实际需求来设置成员变量。）

总结：

1.成员变量在使用初始化列表初始化时，与构造函数中初始化成员列表的顺序无关，只与定义成员变量的顺序有关。

2.如果不使用初始化列表初始化，在构造函数内初始化时，此时与成员变量在构造函数中的位置有关。

3.类中const成员常量必须在构造函数初始化列表中初始化。

4.类中static成员变量，只能在类外初始化(同一类的所有实例共享静态成员变量)。

详情参见博客<https://blog.csdn.net/no_sying_nothing/article/details/51958547>

2.如果函数需要传入一个指针，什么时候需要加const，将const放在什么位置有什么区别？

（这一部分比较绕，我多读了几遍并加上了自己的理解，如果不对，还请大佬指正！orz））

回答问题之前首先让我们来弄清楚cosnt 是干什么的？

C++ const 允许指定一个语义约束，编译器会强制实施这个约束，允许程序员告诉编译器某值是保持不变的。如果在编程中确实有某个值保持不变，就应该明确使用const，这样可以获得编译器的帮助。

const与指针的关系分为两种：const修饰的指针和指向const对象的指针，二者const的位置不相同。

* 指向const 对象的指针（const 位于“\*”前 ）

对于一个const对象，必须用一个指向const的指针的指向它。因为由于const的修饰，对象无法改变。

比如下面的const对象ival，指针ptrVal（这个指针是指向const的指针！）指向它，所以ival 对象无法改变。

const int ival = 1;

const int \*ptrVal = &ival

而对于一个指向const对象的指针，可以指向任意一个对象

 int \*ptr = &val; 将val的地址赋值给ptr,地址只是个值，所以ptr也不知道自己指的对象是不是const.

将一个地址赋值给指向const的指针（ptrVal），见下面语句第二行

int ival = 1;

const int \*ptrVal = &ival;

注意，ival并非const变量，所以可以通过赋值改变ival的值。而ptrVal指向的是“自以为是”const的对象，所以无法通过\*ptrVal改变ival的值。

* const修饰的指针（const在”后”）

int \* const ptr;

上述声明了一个const类型的指针，即指针本身是一个常量，不能修改。如何理解？ 指针本身的值是一串地址，即这个地址值不能被修改，也就是说指针只能指向这个地方，不能指向其他地方。

但指针所指向的地址的内容并不属于指针本身的值，所以其指向的内容可以改变。

int ival = 1;

int \*const ptr = &ival;

\*ptr = 2; //yes！ptr指向地址的内容可以边

int ivalTwo = 11;

ptr = &ivalTwo; //no! ptr本身不能变（不能指向别处咯）

3.函数传入值参数、引用参数、指针参数有什么区别？什么时候需要为传入的引用参数加上const？

* 值传递

函数调用过程中产生一个临时副本，函数对传入参数的操作是对副本的操作，不改变实参的值，无需const保护

虽然值传递的保护参数不被修改很好，但产生临时副本，若传入的是一个对象，因为对象需要构造、复制、析构等操作，增加副本效率不高。

void foo1(int x);

void foo2(ClassType instance); //副本instance开销较大

下面的const保护毫无意义

void foo1(const int x);

void foo2(const ClassType instance);

* 引用传递

传入实参，降低开销。引用即本身，不额外产生副本

void foo1(int &x);

void foo2(ClassType &ref);

注意：引用可以通过函数改变传入的参数来改变实参。这对与实参来说，比较危险，这时候需要通过const修饰来保护传入的引用不被修改。

void foo1(const int &x);

void foo2(const ClassType &ref);

对于基本内部类型（int ,char，double…），不存在对象构造等操作，下面两种保护参数不被修改的效率一样

void foo1(int x);

void foo1(const int &x);

* 指针传递

指针传递在保护参数时不被修改上和引用传递一样（需要const）,但指针传递还可以扩大接受参数的范围。

void foo1(const ClassType \*ptr);前面讲到，ptr指向了一个自以为是的const类型的对象。传入的对象不一定是const修饰，即可以const对象，也可以非const对象。

反之，若没有const修饰函数的形参，只能传入非const对象。

注意，无论指针传递传入的是否是const对象，都无法通过指针去修改这个对象，与上面不能修改的是一致的。且，cosnt只能修饰一个输入参数；输出参数无论是引用还是指针传递，都不能用const修饰。

参考<https://www.jb51.net/article/82269.htm>



好了今天的分享时间就到这里了，欢迎批评欢迎建议。

**我也是搬运工，知识共享，跪求大佬纠错，表情包随便取。**

**只在公众号(Darring的象牙塔)和微博（@张丹颖呀）不同时发布。**