**基础知识**

**C++基础**

1. 面试官看重的是代码的调试能力和编程习惯
2. 除了完成基本功能，还要考虑错误输入、边界处理，很重要。
3. 面试官给提问环节，最好问 和 求职部门相关的项目和技术，不要向非HR问工资，也不要问自己的面试表现怎么样？
4. A a = 10; 执行了两步：先将10隐式转换为A(10)； 再调用拷贝构造函数将 A a = A（10）； <https://blog.csdn.net/twdlll/article/details/78302349>
5. 类A的复制构造函数为A(A other)这样是不行的，因为是传值调用，那么调用这个拷贝构造函数的时候：形参用拷贝构造函数给实参赋值，这样就会循环调用拷贝构造函数，导致栈溢出。
6. 重载运算符：返回值 class::operator = ( const参数&),如果一个运算符函数是成员函数，那么它的左侧运算对象绑定到掩式的this指针上，因此，成员运算符函数的参数比运算符运算对象少一个。如重载赋值运算符：MyClass& MyClass::operator=( const MyClass&);
7. 复制构造函数和赋值运算符：

都是将一个对象的值复制给另一个对象。

赋值运算符是将对象的值复制给一个已经存在的实例；复制构造函数是用传入的对象构造一个新的对象。二者主要看是否有新对象产生。复制构造函数的定义：Class(const Class& );

复制构造函数：string str = “000”;

赋值运算符： str = str1;

1. C++规定：初始化只在进入构造函数体之前，进入构造函数体之后就是赋值了；

如：ClassA(int num, string str): age(num), name(string){} 是初始化，如果是ClassA(int num, string str){age = num; name = str;}就是赋值了。

1. 因为不能拷贝数组，所以数组形参会自动转化为指针。所以：

Void print（int \*）；

Void print(int[]);

Void print(int[10]); //这三个虽然形式不同，但是都是传入的指针类型

1. 数组和指针：当声明一个数组时，数组名也是一个指针，该指针指向数组的第一个元素。我们可以使用指针来访问数组。如：

Int data1[] = {1,2,3,4};

Int data2\* = data1;

GetSize(data1); //函数GetSize的声明为GetSize( int data[]);

如果sizeof（data1），等于16；而sizeof（data2），等于4； 将数组传入函数中，那么此时data1是指向数组第一个元素的指针。

上述例子访问数组的第三个元素可以：

1. data1[2]
2. data1 + 2;
3. data2 + 2;
4. 从函数返回数组，返回的是数组的指针，局部变量需声明为static：

<https://www.runoob.com/cplusplus/cpp-return-arrays-from-function.html>

1. Sizeof是关键字，用于获取类或数据类型的大小。
2. 为了节省内存，C/C++把常量字符串放到单独的一个内存区域，将此字符串赋值给不同的指针时，是指向相同的区域。用数组时则有所不同。

Char str1[] = “hello, world”;

Char str2[] = “hello , world”;

Char \* str3 = “hello,world”;

Char \* str4 = “hello, world”;

Str1和str2是两个不同的数组， str3和str4指向相同的区域。

**其他基础知识**

1. 单例设计模式：

<https://blog.csdn.net/qq_44735907/article/details/117292159>

1. 互斥锁

#include<mutex>

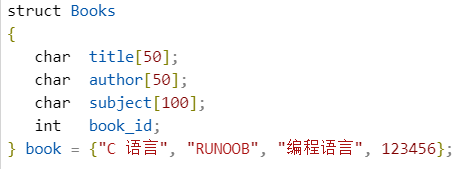
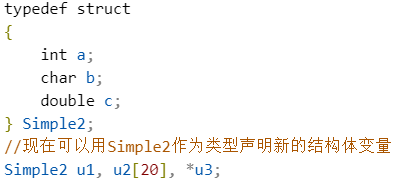
Std::mutex mtx;

Mtx.lock(); mtx.unlock(); mtx.try\_lock();

1. UML图

<https://blog.csdn.net/u010648555/article/details/54379051>

1. C语言结构体



1. 锁：

配合博文：<https://blog.csdn.net/drdairen/article/details/73554843>

互斥锁：持有者使用，其他只能睡眠

自旋锁：持有者使用，其他循环等待

信号量：PV操作

顺序锁：

读写锁：

1. Tcp协议规定了端口号，IP协议规定了ip地址， Mac协议规定了Mac地址

**刷题总结**

1. 从尾向前打印链表

方法一：使用std :: reverse

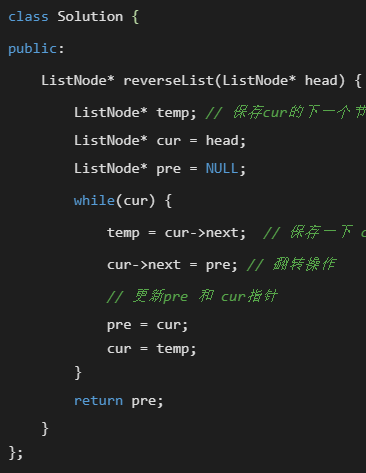
方法二：参考后序遍历,

方法三：反转链表后输出

ListNode\* cur; cur -> next = head; //这是错的，因为cur只进行了声明，没有进行定义：ListNode\* cur = new ListNode(0); 才是对的

1. 反转链表：

方法一：双指针，迭代的时候维护前一个结点和当前结点，记住下一个结点就行，下一次结点不用维护，也就是说只更新前一个结点和当前结点。



**Temp = cur -> next; 是保存**

**Cur->next = pre; 是将cur的next指向pre**

方法二：递归

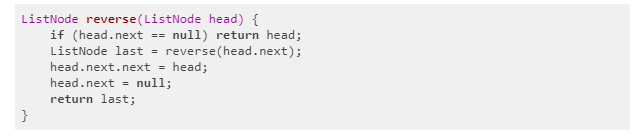
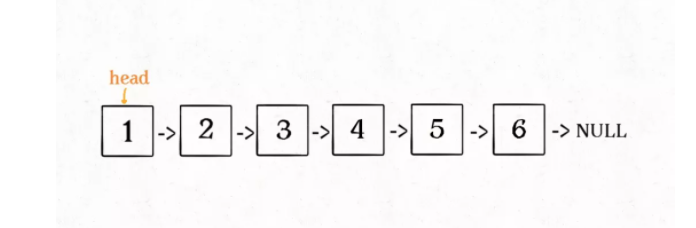
**递归怎么学？用递归反转链表来学！！！**

**递归最重要的就是搞明白递归函数的定义！！！**

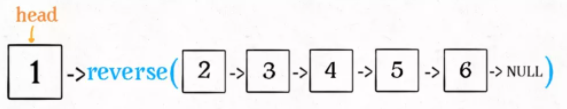
[**https://mp.weixin.qq.com/s?\_\_biz=MzAxODQxMDM0Mw==&mid=2247484467&idx=1&sn=beb3ae89993b812eeaa6bbdeda63c494&chksm=9bd7fa3baca0732dc3f9ae9202ecaf5c925b4048514eeca6ac81bc340930a82fc62bb67681fa&scene=21#wechat\_redirect**](https://mp.weixin.qq.com/s?__biz=MzAxODQxMDM0Mw==&mid=2247484467&idx=1&sn=beb3ae89993b812eeaa6bbdeda63c494&chksm=9bd7fa3baca0732dc3f9ae9202ecaf5c925b4048514eeca6ac81bc340930a82fc62bb67681fa&scene=21#wechat_redirect)

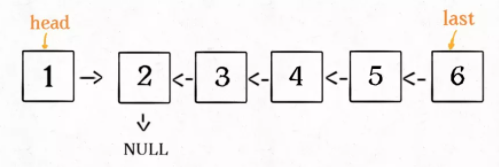
**递归反转链表：**

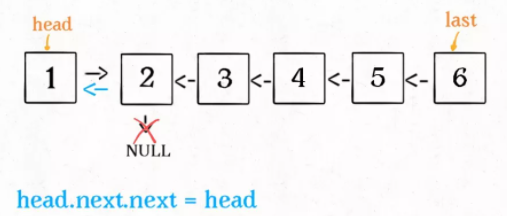
**原链表**

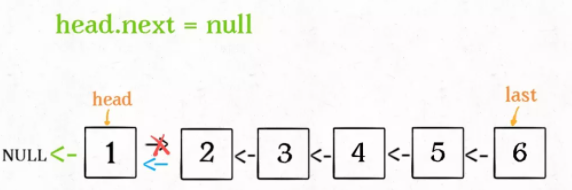


**反转下一个节点后**







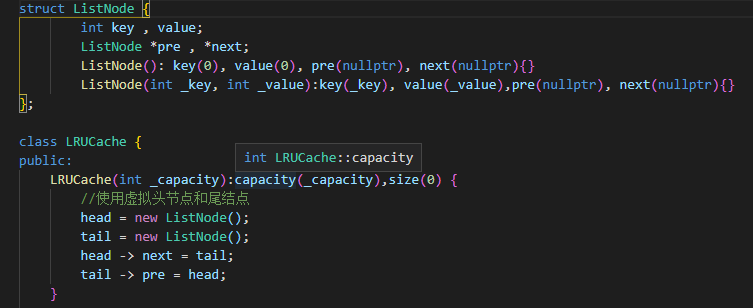


1. //有序无重复数组，使用二分法； 二分区间为【left, right】,则while（left **<=** right）且right = mid -1 或 left = mid +1; 面试老老实实用int，别用size\_t；；；旋转数组找最小数：二分！！！二分查找如果有target，跟target比，如果没有target，那么就和端点比
2. 层序遍历就是得到每个子树的父结点，二叉树就是递归！！！！如根据前序遍历和中序遍历结果重建二叉树就是 将根节点找出来，再分为左右子树分别重建
3. 给链表中插入一个节点，注意判断插入的是不是头节点
4. 涂长方形问题，可以竖着涂，也可以横着涂，竖着涂就像跳台阶跳一级，横着涂就像跳台阶跳两级，那么就可以了，牛客剑指10题
5. 用栈实现队列：一个用于push, 另一个用于pop。用队列实现栈，一个队列用于push, 当要pop的时候就把除了最后一个元素移到另一个里去。可以用que.size得到队列中还剩多少元素。
6. 二分法，快速排序，归并排序都要会！！！！
7. 旋转数组的最小数，已经排序了，用二分，不是相等，没有target，没有target的时候一般和端点相比较
8. 如果是无符号数，右移左边添加的是0，如果是有符号位，右移添加的是1，如何判断某一位是否是1，与对应位为1的数做与运算，
9. 树的子结构：JZ17！！！; 判断一颗树是不是另一个树的子结构 ：两个return有东西：
10. 顺时针打印矩阵：JZ19; 只剩一行或一列的特殊情况要考虑！！！
11. 构造回文串：大小写字母判断 ‘A’<ch<’Z’ || ‘a’<ch<’z’ ; 将一个数字变为对应的ascii码： char ch = 70;就行了
12. 剑指21题，判断数组是否越界的条件要放在前面，否则发生越界错误：判断入栈，出栈顺序是否正确，用辅助栈！！

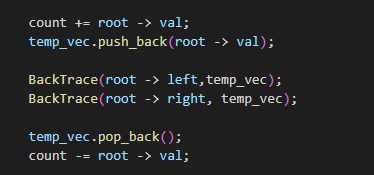


1. Set的插入方法：set.insert(vec.begin(), vec,end())或set.insert({1,2,3})接受迭代器或列表初始化。 关联容器接受的查找算法set.find(); 或find(set.begin(), set.end(), nums);
2. 剑指offer里的回溯和递归：JZ13机器人的运动范围+JZ12矩阵的路径:VISITED数组标记；return 1 + DFS + DFS！！！
3. JZ15统计数中1的个数：使用右移，如果负数则会导致错误，应该使用左移！！且指定flag为unsigned int 类型！！！！
4. JZ16,求数的幂：递归和迭代，如果幂次是负数，要注意处理；另外递归和迭代两个版本。递归就是return myPow(x\*x, n/2); 迭代就是x^n = (x^2)^n/2 = ((x^2)^2)^n/4;那么就可以从x=x\*x ; n = n/2; 那么下次迭代x和n都被更新了，如果有奇数次幂，那么就用res \*=x; 将奇数次幂的单独记录。
5. LRU(least recently used)页面置换算法：使用哈希加双向链表的格式，哈希表用来存储key对应的结点。Unordered\_map<int, ListNode\*>cache; 哈希表插入元素：cache[key] = node ; 或cache.insert({key,node}); 改变取出键key对应的结点并改变值： node = cache【key】; node -> value = value;

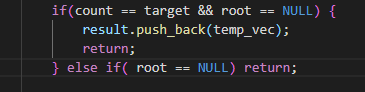
双向链表的定义和初始化：



1. 反转链表：ListNode \* temp = NULL; temp -> next = head; 这里出错！！不能用空指针指向一个结点。
2. JZ33二叉搜索树的后序遍历？判断一个数组是不是二叉搜索树的后序遍历？用最后一个元素将数组分为左右子树数组，然后递归判断是不是二叉树的后序遍历
3. JZ34二叉树中和为某一值的路径，找出二叉树中从跟结点到叶子结点的路径和为target的路径集合。回溯算法：***二叉树的回溯不太一样！！！！***出现问题，二叉树做选择后取消选择的位置！！！



结束条件：下面这样写会导致重复集合



1. JZ35复杂链表的复制。有一个random指针可能指向任何一个结点。复制可以使用哈希表【原结点，新结点】hash[原结点]就得到了新节点，那么hash[源节点]-》 next= hash[原结点->next]!!!!

第二种方法：构建新链表：原结点->新结点->原结点->新结点… 那么原结点的->random->next 就是新结点的random。

1. 二叉搜索树与双向链表。将二叉搜索树改为双向链表。声明pre和head两个私有成员，用pre记录前一个结点，head记录头结点，不断更新pre，最后的时候pre就等于尾结点了，



1. 动态规划： （1）就是穷举，列出所有可能的状态，从这些状态中选一个最好的，这个状态也能代表整体的最好，这就是最优子结构；（2）同时，动态规划要有重叠的子问题，能够递推，