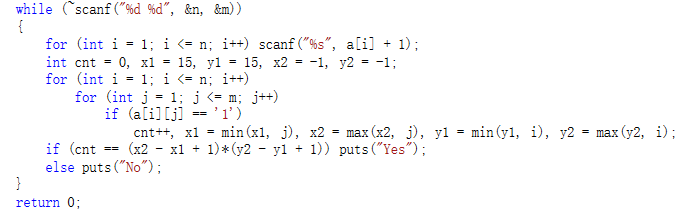
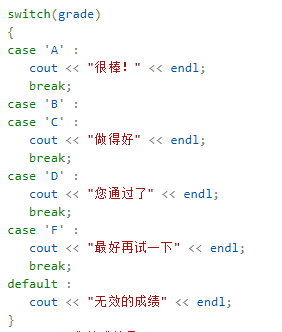
# 输入输出

ACM赛制：



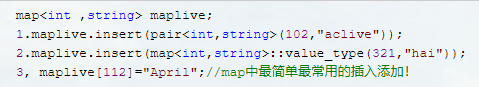
# 实际编程使用

## Switch

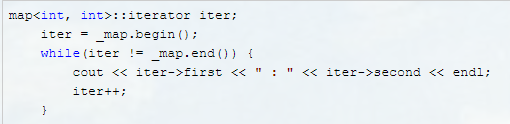


## Map

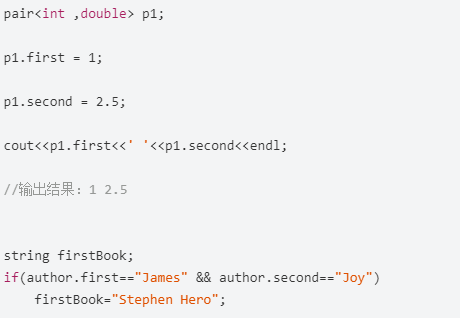
插入



遍历



## Pair



# Github上传代码步骤

git status

状态：

1：Untracked: 未跟踪, 此文件在文件夹中, 但并没有加入到git库, 不参与版本控制. 通过git add 状态变为Staged.

2：Modified: 文件已修改, 仅仅是修改, 并没有进行其他的操作.

3：deleted： 文件已删除，本地删除，服务器上还没有删除.

4：renamed

git add + 文件

git add -u + 路径：将修改过的被跟踪代码提交缓存

git add -A + 路径: 将修改过的未被跟踪的代码提交至缓存

git commit -m “注释部分”

git push

完成

# C++软件开发面试题

## 关键字用途

Static：

默认初始化为0；函数体内与函数体外的定义；类内定义；文件内定义变量或者函数名。

This：

Const:

Ifndef/define/endif

Extern “C”

struct/class

new/malloc

heap/stack

vector/list

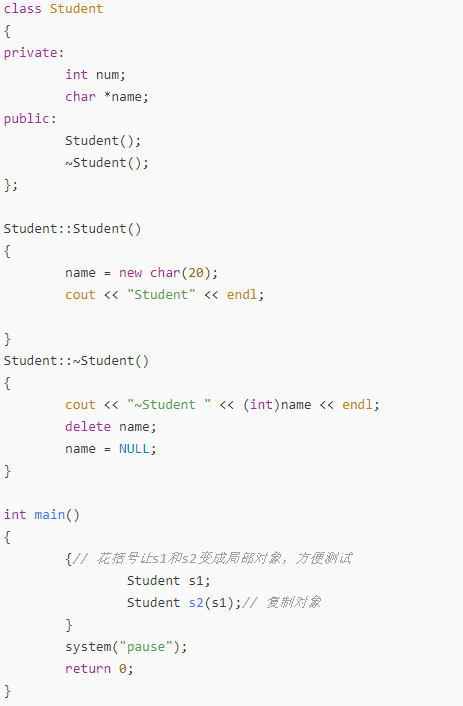
inline

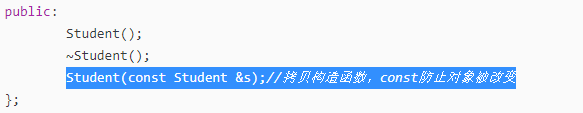
## 23种设计模式

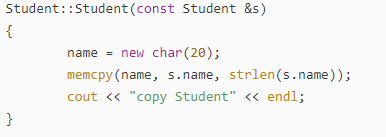
# 常见的面试问题

1. 深拷贝与浅拷贝的区别？

对于什么是浅拷贝，定义下列student类说明。

定义构造函数时为name申请内存，在析构函数中释放申请的内存；main函数中，定义s1对象时调用构造函数，作用域结束时调用析构函数；第二句会调用默认的拷贝构造函数，不会调用定义的构造函数，作用域结束时调用析构函数，释放name，这样就释放了两次，导致程序崩溃。上述过程就是浅拷贝。

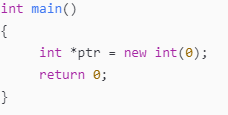
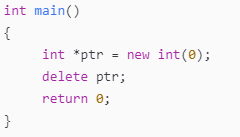
因此需要自定义拷贝构造函数，进行深拷贝，避免内存泄漏。



这样的过程就是深拷贝。

总结：浅拷贝只是对指针的拷贝，拷贝后两个指针指向同一个内存空间，深拷贝不但对指针进行拷贝，而且对指针指向的内容进行拷贝，经深拷贝后的指针是指向两个不同地址的指针。

扩展的问题：智能指针与完整的一个String类

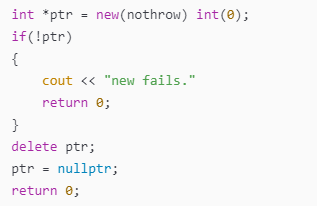
左图会导致内存泄漏；

右图虽然已经释放申请的内存，但ptr

成为野指针；需要ptr = nullptr;

如果new申请内存不成功（由于内存有限）

可以进行如下改进：

使用智能指针的三点原因：

智能指针能够帮助我们处理资源泄露问题；

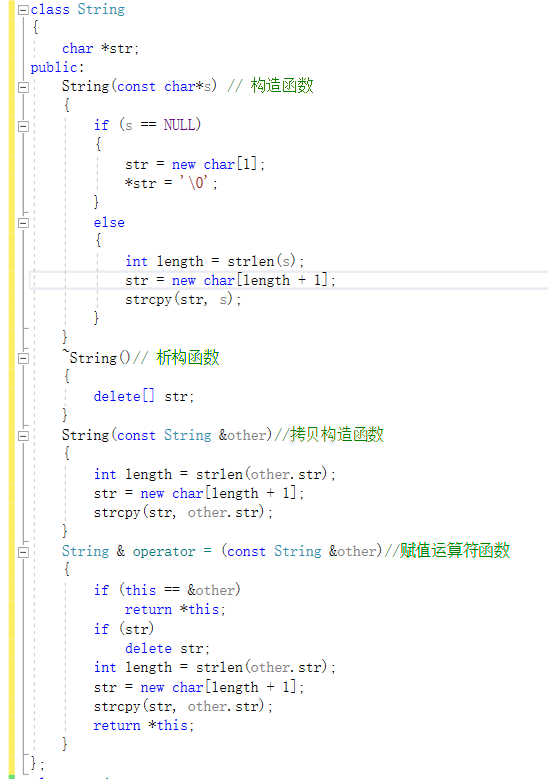
它也能够帮我们处理空悬指针的问题；

它还能够帮我们处理比较隐晦的由异常造成的资源泄露。

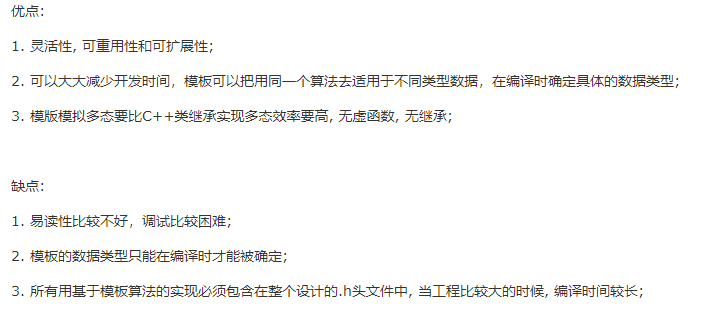
shared\_ptr：

unique\_ptr：

标准String类



1. 类模板的优缺点



1. 内存的几大区域

栈区：由编译器分配和释放，存放函数参数值，局部变量等

堆区：由程序员分配和释放，可能出现内存泄漏的问题，new

全局区：全局变量与静态变量；分为全局初始化区与全局未初始化区

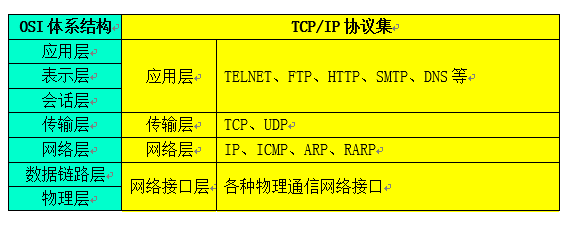
文字常量区：存放常量字符串

代码区：存档函数的二进制代码

1. 递归的后果

容易栈溢出，因为函数实参存于栈区

1. TCP/IP协议

网络接口层：物理层次的一些接口，电缆等。

网络层：提供独立于硬件的逻辑寻址,实现物理地址与逻辑地址的转换。

传输层：为网络提供了流量控制,错误控制和确认服务。

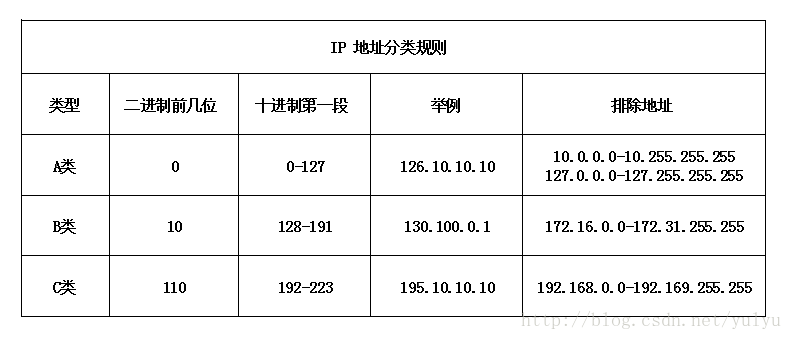
应用层：为网络排错,文件传输,远程控制和　Internet 操作提供具体的应用程序。

绝大部分 IP 地址属于以下几类

A 类地址：IP 地址的前 8 位代表网络 ID ，后 24 位代表主机 ＩＤ；

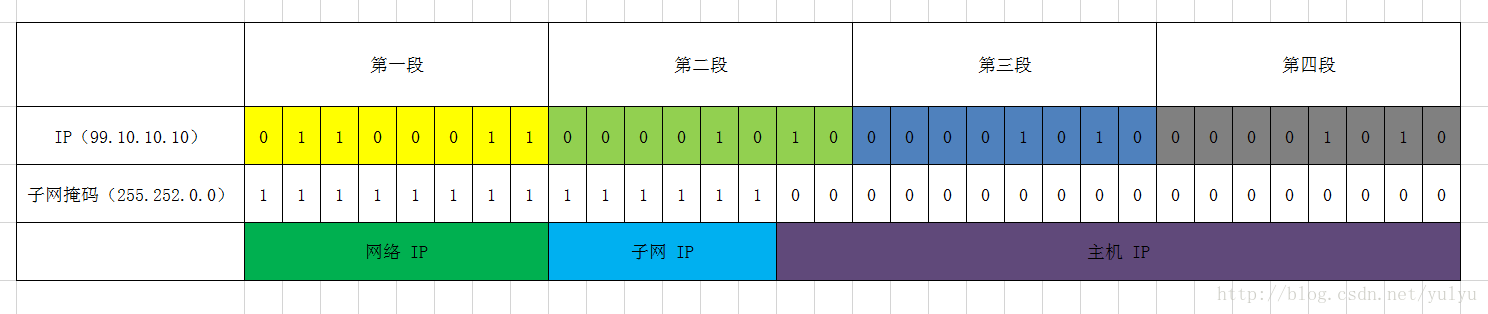
B 类地址：IP 地址的前 16 位代表网络 ID ，后 16 位代表主机 ＩＤ；

C 类地址：IP 地址的前 24 位代表网络 ID ，后 8 位代表主机 ＩＤ。



全是 0 的主机 ID 代表网络本身，比如说 IP 地址为 130.100.0.0 指的是网络 ID 为130.100 的 B 类地址；全是 1 的主机 ID 代表广播，是用于向该网络中的全部主机方法消息的。 IP 地址为 130.100.255.255 就是网络 ID 为 130.100 网络的广播地址（二进制 IP 地址中全是 1 ，转换为十进制就是 255 ）。

子网掩码的使用：



1. 面向对象的概念：封装，继承和多态
2. Vector和list的区别：vector类似于数组（随机访问），list类似于双向链表（插入和删除）
3. 软件测试方法：

白盒测试、黑盒测试、灰盒测试、静态测试、动态测试

白盒测试：你可以清楚盒子内部的东西以及里面是如何运作的，因此白盒测试需要你对系统内部的结构和工作原理有一个清楚的了解，并且基于这个知识来设计你的用例；

黑盒测试又叫功能测试：这是因为在黑盒测试中主要关注被测软件的功能实现，而不是内部逻辑。在黑盒测试中，被测对象的内部结构，运作情况对测试人员是不可见的，测试人员对被测产品的验证主要是根据其规格，验证其与规格的一致性；

灰盒测试：白盒测试和黑盒测试往往不是决然分开的，一般在白盒测试中交叉使用黑盒测试的方法，在黑盒测试中交叉使用白盒测试的方法。灰盒测试就是这类界于白盒测试和黑盒测试之间的测试；

静态测试：是一种不通过执行程序而进行测试的技术。它的关键功能是检查软件的表示和描述是否一致，没有冲突或者没有歧义；

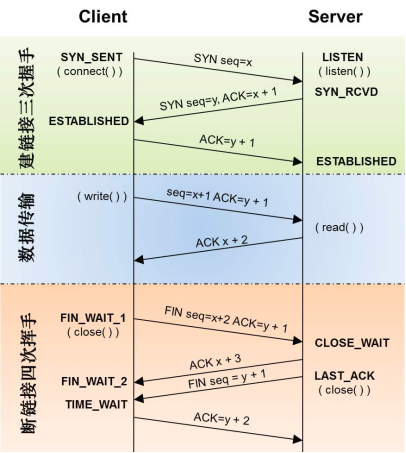
动态测试：包含了程序在受控的环境下使用特定的期望结果进行正式的运行。它显示了一个系统在检查状态下是正确还是不正确。

1. 进程与线程：

进程是对运行时程序的封装，是系统进行资源调度和分配的的基本单位，实现了操作系统的并发；

线程是进程的子任务，是CPU调度和分派的基本单位，用于保证程序的实时性，实现进程内部的并发；线程是操作系统可识别的最小执行和调度单位。每个线程都独自占用一个虚拟处理器：独自的寄存器组，指令计数器和处理器状态。每个线程完成不同的任务，但是共享同一地址空间（也就是同样的动态内存，映射文件，目标代码等等），打开的文件队列和其他内核资源。

1. 一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。线程依赖于进程而存在；
2. 进程在执行过程中拥有独立的内存单元，而多个线程共享进程的内存。（资源分配给进程，同一进程的所有线程共享该进程的所有资源。同一进程中的多个线程共享代码段（代码和常量），数据段（全局变量和静态变量），扩展段（堆存储）。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。）
3. 进程是资源分配的最小单位，线程是CPU调度的最小单位；
4. 系统开销： 由于在创建或撤消进程时，系统都要为之分配或回收资源，如内存空间、I／o设备等。因此，操作系统所付出的开销将显著地大于在创建或撤消线程时的开销。类似地，在进行进程切换时，涉及到整个当前进程CPU环境的保存以及新被调度运行的进程的CPU环境的设置。而线程切换只须保存和设置少量寄存器的内容，并不涉及存储器管理方面的操作。可见，进程切换的开销也远大于线程切换的开销。
5. 通信：由于同一进程中的多个线程具有相同的地址空间，致使它们之间的同步和通信的实现，也变得比较容易。进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信——需要进程同步和互斥手段的辅助，以保证数据的一致性。在有的系统中，线程的切换、同步和通信都无须操作系统内核的干预
6. 进程编程调试简单可靠性高，但是创建销毁开销大；线程正相反，开销小，切换速度快，但是编程调试相对复杂。
7. 进程间不会相互影响线程一个线程挂掉将导致整个进程挂掉
8. 进程适应于多核、多机分布；线程适用于多核
9. 三次握手四次挥手

三次握手：SYN同步请求，ACK确认

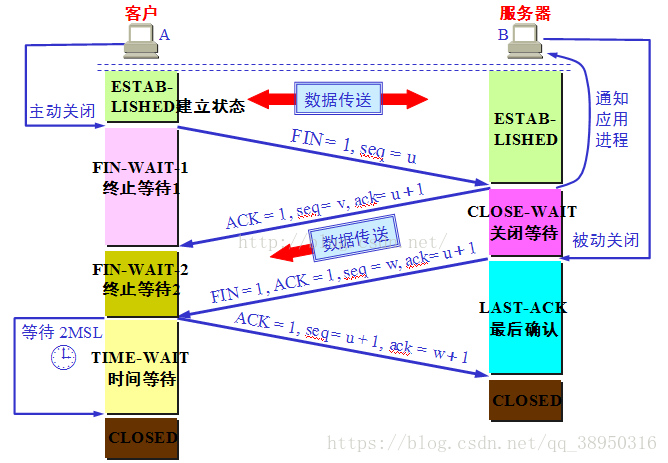
第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据.

四次挥手：FIN希望断开连接

1. 客户端进程发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入FIN-WAIT-1（终止等待1）状态。 TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。
2. 服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。
3. 客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）
4. 服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最后确认）状态，等待客户端的确认
5. 客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过2∗∗MSL（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。
6. 服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。
7. 树

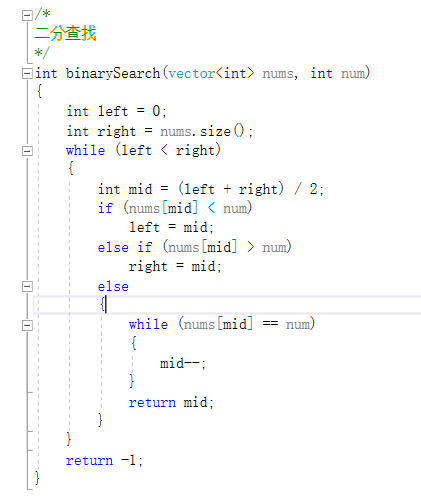
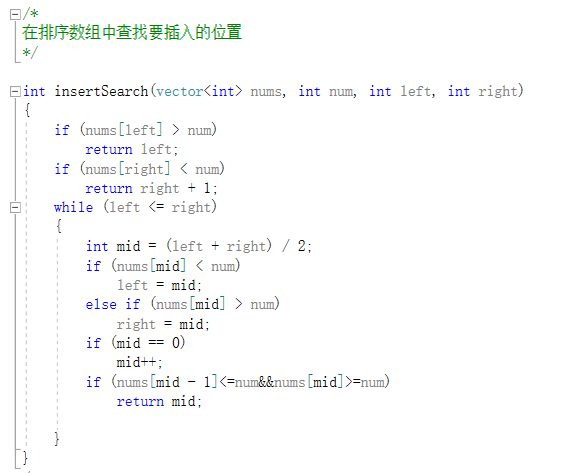
平衡二叉树（AVL树）：是一种特殊的二叉排序树。其左右子树都是平衡二叉树，且左右子树高度之差的绝对值不超过1。

红黑树：红黑树是一种二叉查找树，但在每个节点增加一个存储位表示节点的颜色，可以是红或黑（非红即黑）；每个节点非红即黑、根节点是黑的、每个叶节点（叶节点即树尾端NULL指针或NULL节点）都是黑的、如果一个节点是红色的，则它的子节点必须是黑色的、对于任意节点而言，其到叶子点树NULL指针的每条路径都包含相同数目的黑节点；

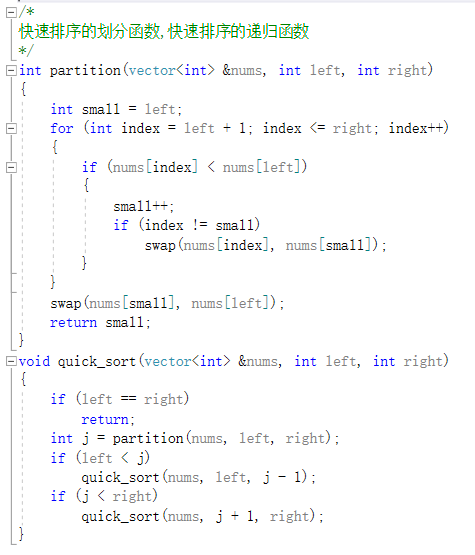
1. 哈夫曼编码

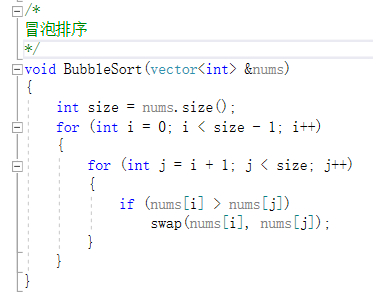
# 算法与数据结构部分

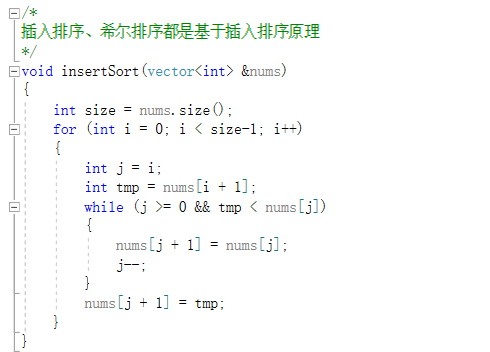
## 查找算法

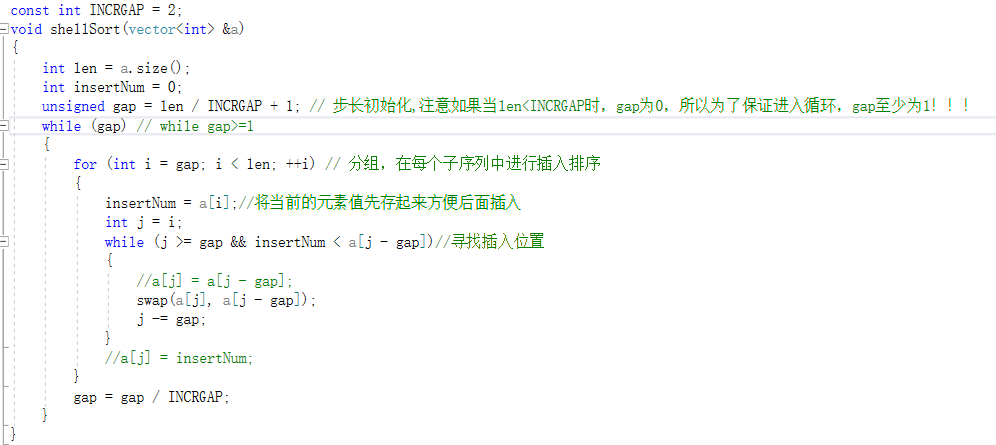
二分法以及二分接近查找

## 排序

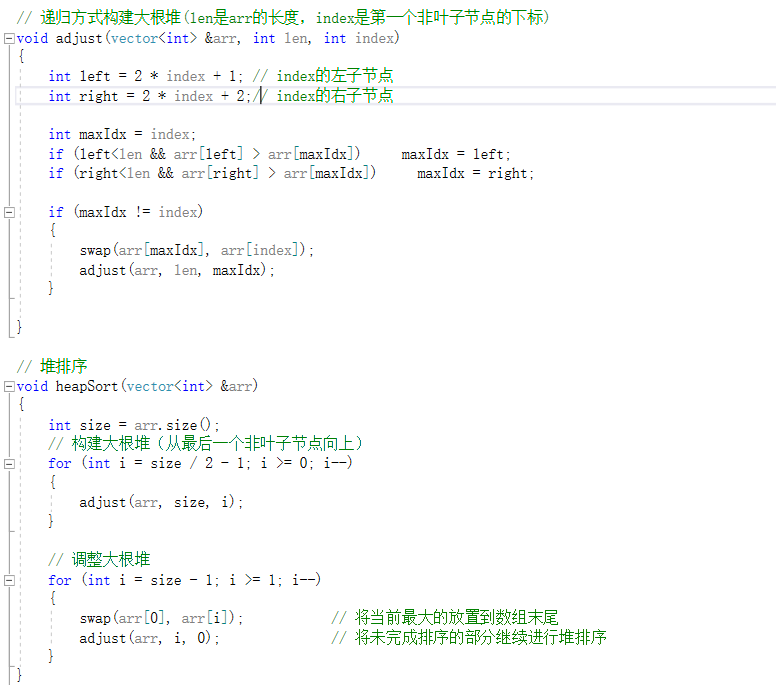


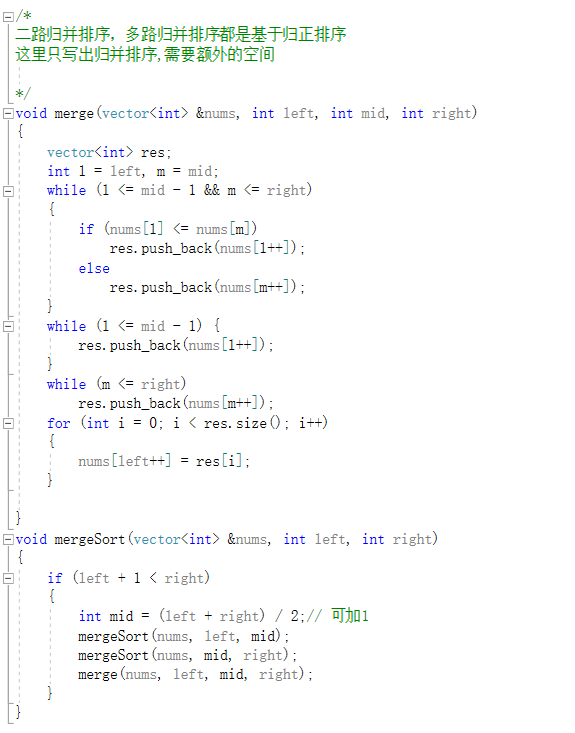












# 剑指offer

## **链表**

倒数第K个节点：

先走k-1步，再同步走，注意是否超出长度与k==0；

链表环的入口节点：

首先根据一快一慢指针找到环的节点个数，然后指针3首先走n步，指针4与指针3一起单步向前走；

反转链表：

首先定义pre = nullptr 然后遍历pNode，一定先记录下一节点，再去反转。

合并排序的链表：

判断两个头结点大小，作为新链表头结点，下一节点就用递归.很巧妙

## 树

树的子结构：输入二叉树A和B判断B是不是A的子结构

分为两步：第一步判断是否相等，再去遍历A,B查找是否是子结构。如果不相等，递归A的左子树与右子树。

求二叉树的镜像与判断二叉树的镜像

直接遍历，然后交换左右节点。判断是否是镜像二叉树与前面不同。则是判断该节点是否相同，若相同，将a->left 与b->right对比，a->right与b->left 对比。若不相同没直接返回false；

对称的二叉树：

层序打印二叉树：

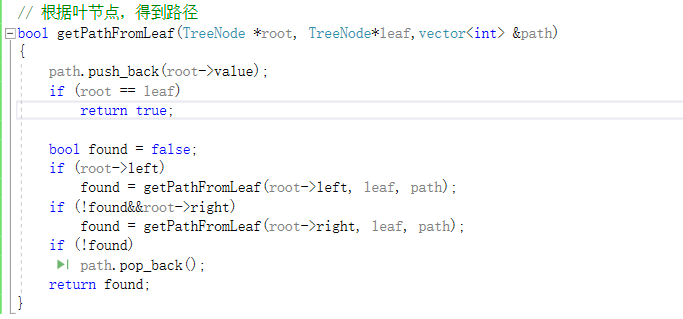
二叉树和为某值的路径：

序列化二叉树：

面试题68 树中两个节点的最低公共祖先

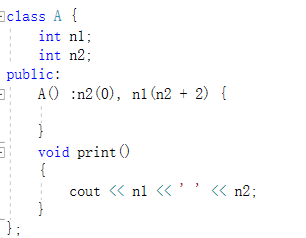
如果是二叉搜索树的情况：从根节点开始判断是否在两个叶节点值得中间，如果是，那么根节点就是最低公共祖先，如果不是遍历搜索（左右子树）。

如果不是二叉搜索树，甚至不是二叉树？看是否有指向父节点的指针？有则从叶节点遍历回到根节点，比较这两个数组，可以找到最低公共祖先。如果没有指向父节点的指针，从根节点开始，查找是否有这两个叶节点，（根节点当然是有的），再去找左子树与右子树（递归）。在借助辅助空间的情况下，计算出路径，找出直到有两个叶节点的路径。以下是根据叶节点得到路径的算法：



面试题67 把字符串转换成整数

首先需要注意的是：构造函数的初始化与声明的顺序有关，而与语句的前后无关A():n2(0),n1(n2+2)



Atoi用于把字符串转换成整数。实现string转int

需要注意的是：字符串是否合法，需要检查字符串是否为空，是否输入正负号，是否有字母等其他非法字符等。

面试题66 构建乘积数组

给定数组A[0,1,2,3,…,n-1]，构建乘积数组B[i] = A[0]\*A[1]\*…A[i-1]\*A[i+1]\*…\*A[n-1]。根据i可划分为C[i] = A[0]\*A[1]\*…A[i-1],C[i] = C[i-1]\*A[i-1]，D[i]= D[i+1]\*A[i+1].B[i] = C[i]\*D[i].我们可以遍历一次，计算出CD数组，从而再遍历一次计算数组B。

面试题65 不用加减乘除做加法：

加法类似于异或^（但是没有进位），此时需要标记处进位。

首先 sum = num1^num2(没有考虑进位的加法)

获取进位算法carry = （num1&&num2）<<1;

循环num1 = sum;num2 = carry（把进位考虑进去）

面试题64 求1+2+3+…+n

求和函数，构造函数（定义静态变量，定义n个对象）

面试题63 股票的最大利润（就是求一个数组的最大差值，可以找到最小值与最大值）

面试题62 圆圈中最后剩下的数字

0,1,2，…n-1排成圆圈，每次剔除第m个数字，求最后剩余的数字。（约瑟夫环的问题）

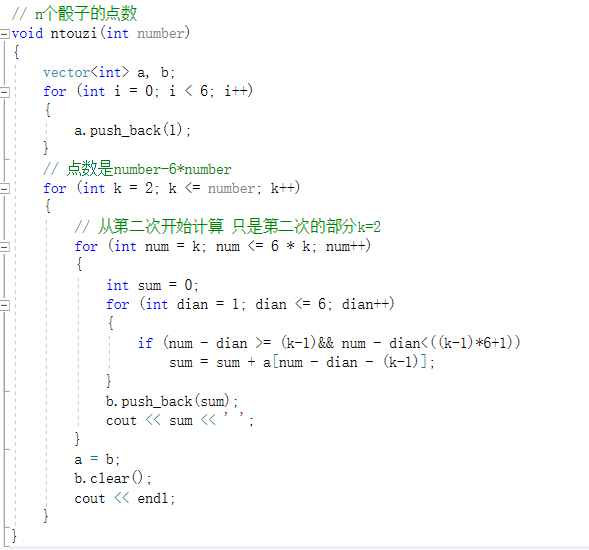
方法1：建立环形链表，每次删除第m个节点（需要自定义链表的删除节点函数）

方法2：由一定的规律关系：n=1,f(n,m) = 0;n>1 f(n,m) = (f(n-1,m)+m )%n;

面试题61 扑克牌中的顺子（简单）

面试题60 n个骰子掷出的点数之和为s的概率（是一道数学题）

第一次出现1-6的个数分别为1，第二次出现n的个数就是前一次（n-1,n-2,n-3,n-4,n-5,n-6）之和（考虑边界条件）



面试题59 队列的最大值

滑动窗口的最大值，基本就用蛮力法，那么队列的最大值呢？可以为队列添加一个成员函数，没增加一个变量，就自主去求最大值（当前最大值与添加值去比较）

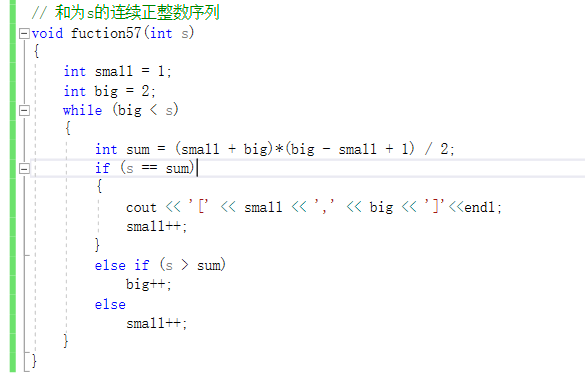
面试题58 翻转单词顺序

需要把每个单词保存下来，如何辨别是否是单词呢（空格 ，‘.’等）

面试题57 输入已排序数组，找到和为s的两个数字

可以定义两个索引，left与right，当num[left]+num[right]>s时，right--；否则left++；直到找到相等的值s。

问题2：和为s的连续正整数序列

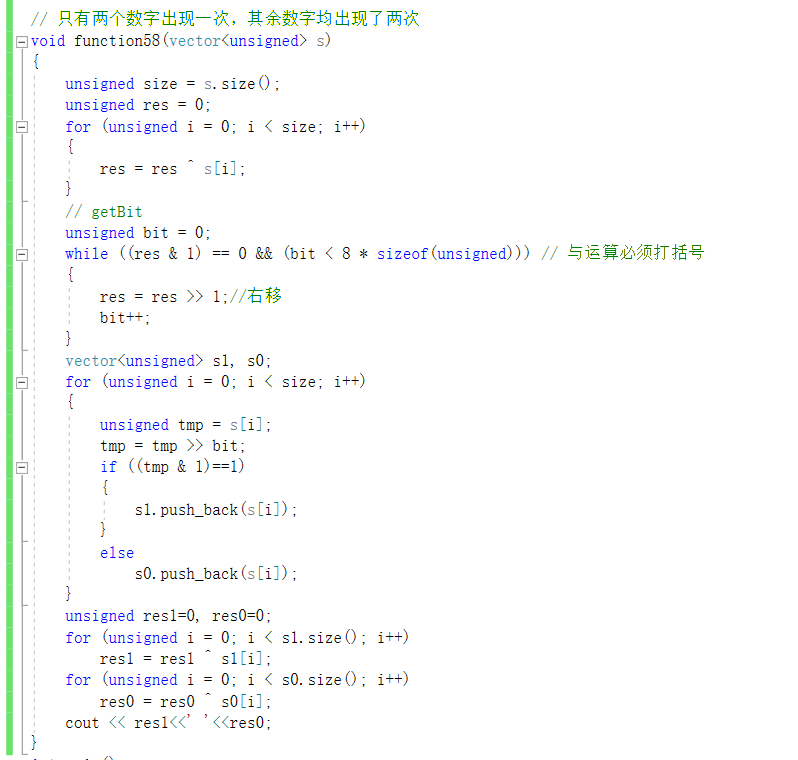


面试题56 一个整型数组中，只有两个数字出现一次，其余数字均出现了两次，找出只出现一次的两个数字。比较困难，两个数相同的话，那么异或是为0的

* 首先遍历，计算所有数字的异或（肯定不为0）
* 找出其中的第几位是1，再遍历数组，该位是1的分成一组，为0的分为一组，从而划分了两个数组，且每个数组中都有只出现一次的数。
* 两个数组分别异或，得到答案

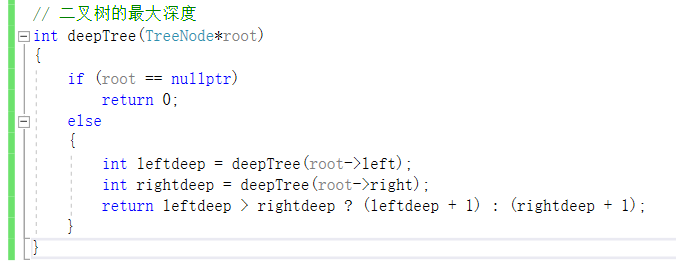
问题2，数组中只有一个数字出现一次，其余数字出现三次，找出来

思路：所有数字的第一位二进制相加，能被3整除（不能被3整除），需要有个掩码（1，10，100,1000，一共32位）



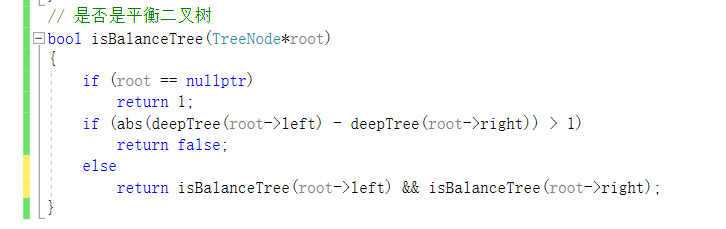
面试题55 二叉树的深度

用递归的方法去计算深度，代码如下



问题2：判断是不是平衡二叉树

关键点：左右子树的的深度不超过1，根据上面计算出的深度，但存在的缺陷就是一个节点可能遍历多次，可以采用后续遍历的方法



面试题 54 二叉搜索树的第K大节点 （中序遍历）

面试题53 在排序数组中查找数字

问题一：数字在排序数组中出现的次数，解题思路，首先是二分查找，当找到一个后，判断前一个是不是也是k，如果是right = mid。这样可以找到第一次出现k的地方，同理找到最后一次出现k的地方。

问题二：0~n-1缺失的数字，有n个数，但其中有一个数不在范围内，找出来，是递增的数组。二分查找，值与下标作对比。（要么是第一个，要么是最后一个，搞不懂这个题目）

面试题52 两个链表的第一次公共节点

思路：先遍历得到两个链表的长度，然后设置两个指针，让长链表先走（长度之差），然后遍历每一步两个指针比较，找到第一个公共节点。

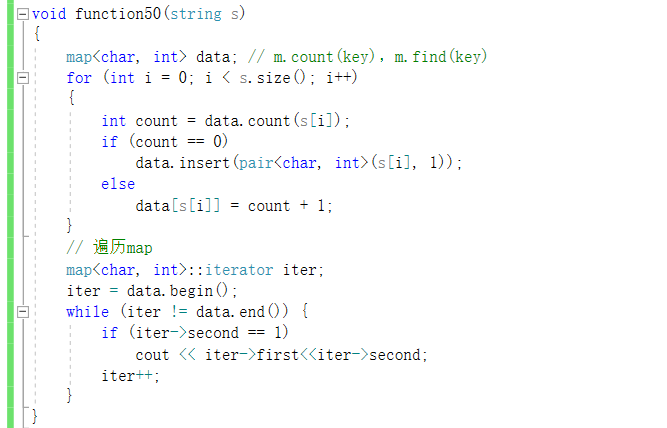
面试题51 数组中的逆序对

给定一个数组，找出其中的逆序对（前一个数字大于后面一个数字）

用归并排序的思想，难度较大。

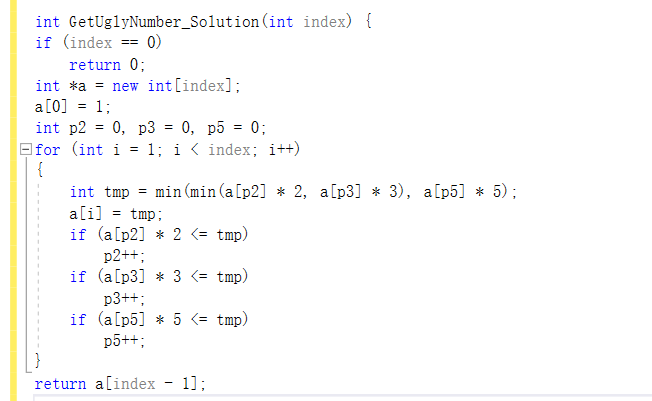
面试题50 第一次只出现一次的字符

字符串中只出现一次的字符，首先是n^2的解法。我们可以定义一个哈希表，Key是字符，Value是出现的次数，再去扫描。map,其实也可以用数组（ASCII码）代替Key。下图涉及map的使用。



面试题49 找到第1500个丑数

只包含因子2,3,5的数，需要看懂思路，记忆



面试题48 最长不包含重复字符串的子字符串

略

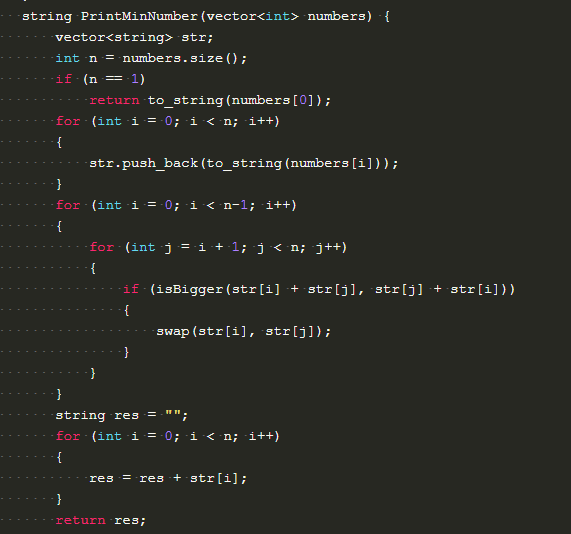
面试题47 矩阵左上角至至右下角最大值的路径（只能向下或者向右运动一步）

思路：保存最开始的值，然后逐步看右一步与往下一步的值，看哪个值大

面试题46 把数字翻译成字符串

面试题45 把数组拼成一个最小的数

类似于冒泡，根据is bigger进行排序，mn与nm

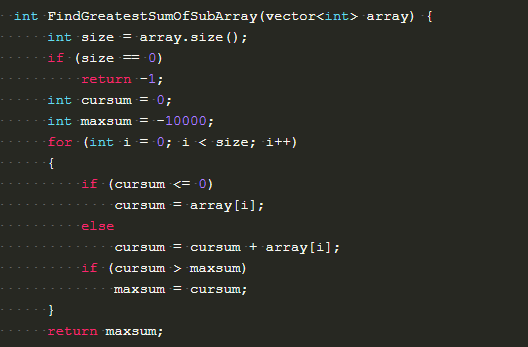


面试题44 略

面试题43 略

面试题42 连续子数组的最大和

F(i) 要么等于nums[i]，要么等于f[i-1] + nums[i];



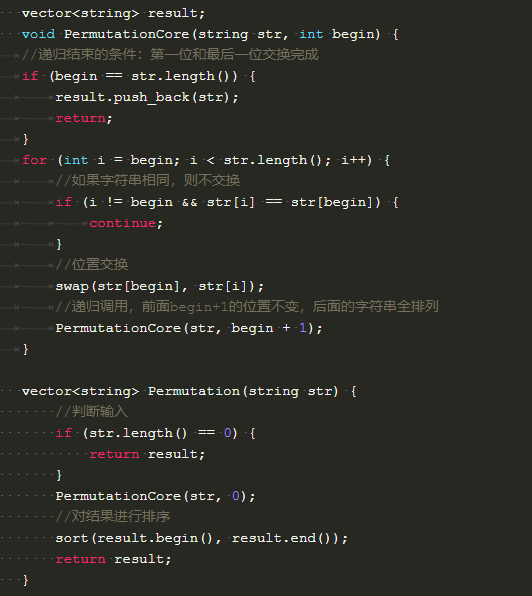
面试题41 略

面试题40 略

面试题39 略

面试题38 字符串的排列

思路：分成两部分，第一字符与剩余字符分别交换（注意均为不相同的字符），交换后，需要交换回来。

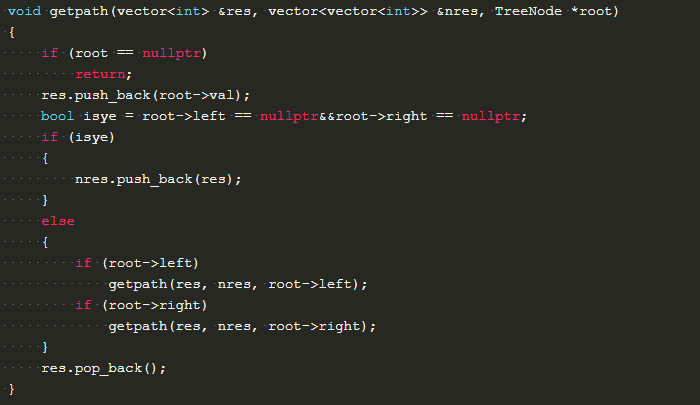


面试题37 序列化二叉树 略

面试题36 略

面试题35 略

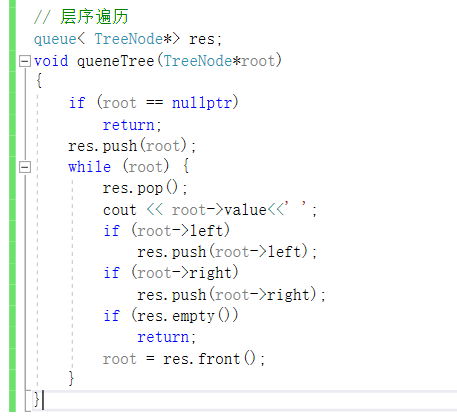
面试题34 二叉树和为某值的路径（重要）



面试题33 二叉搜索树的后序遍历

面试题32 层序遍历二叉树

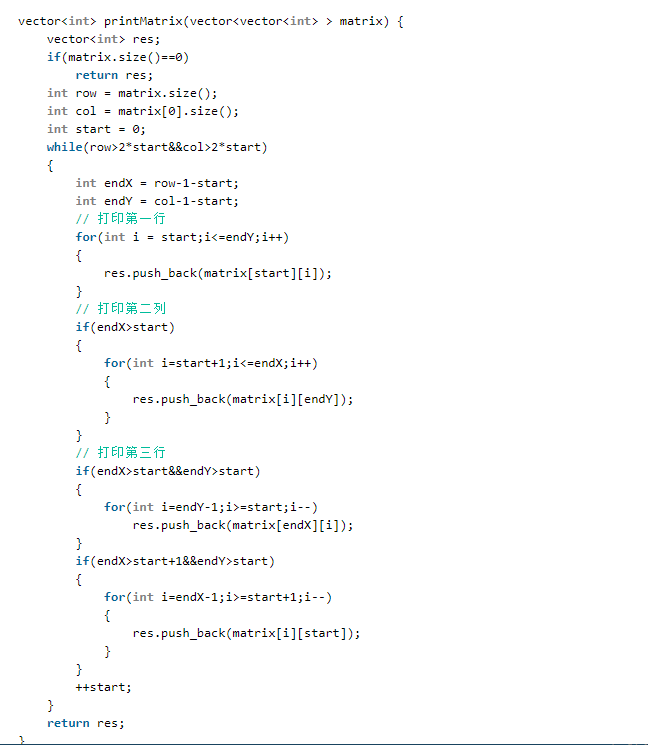
定义quene



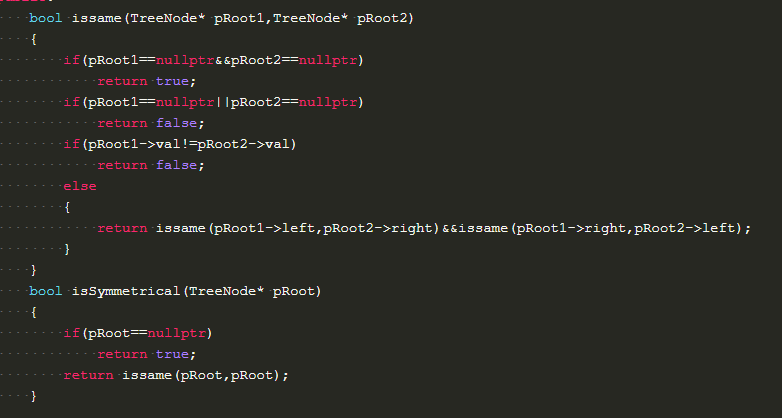
面试题31 略

面试题30 略

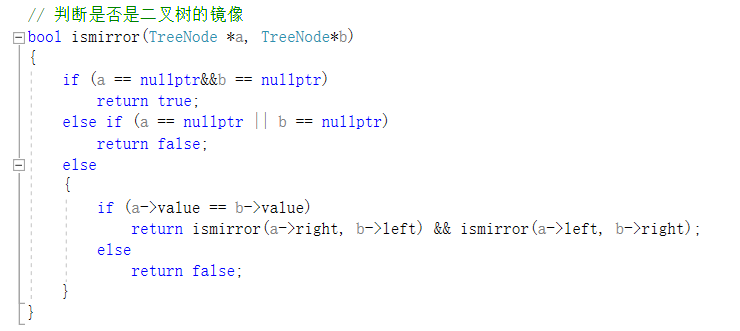
面试题29 顺时针打印矩阵



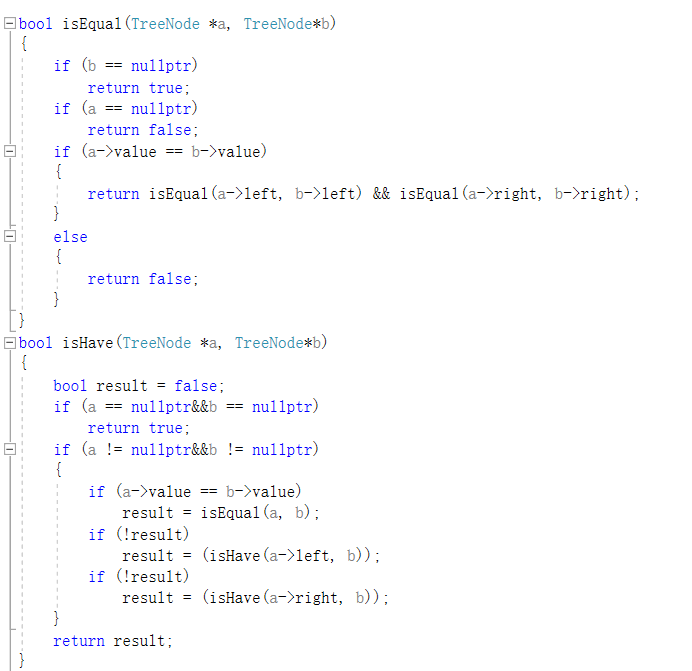
面试题28 是否是对称二叉树



面试题27 二叉树的镜像



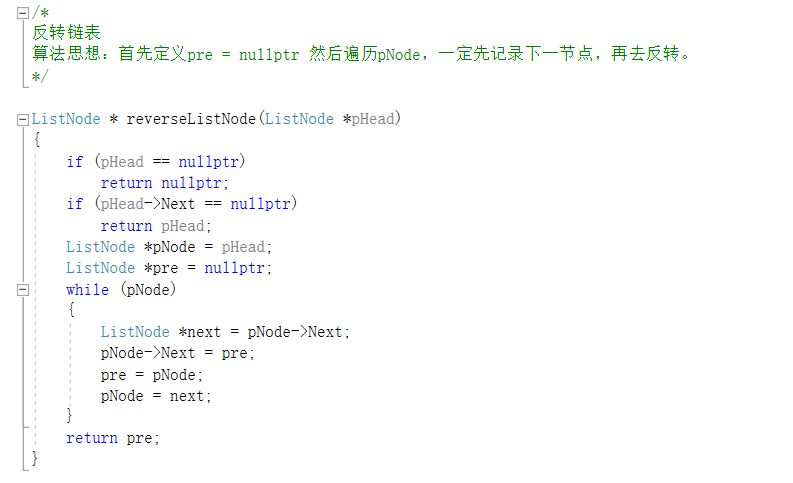
面试题26 判断是否是数的子结构



面试题25 合并排序的链表



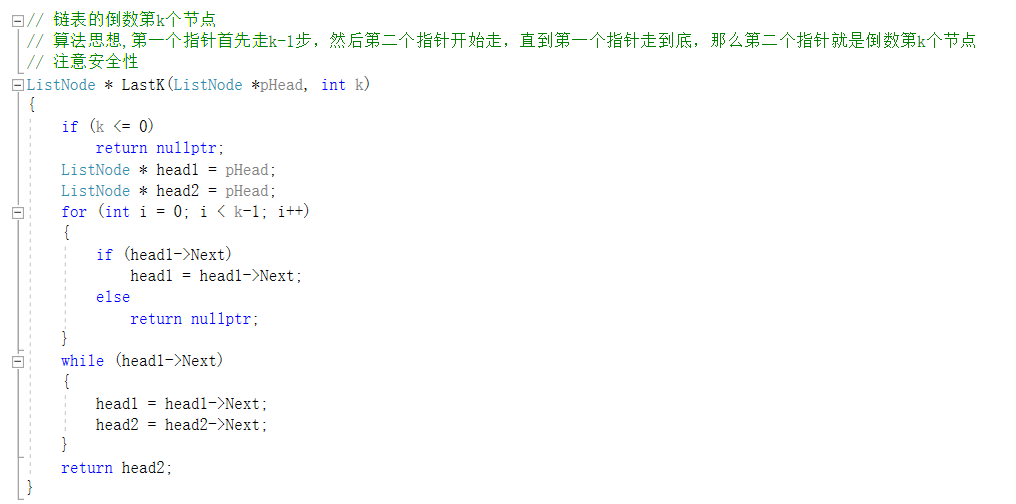
面试题24 反转链表



面试题23 链表环的入口节点

算法思想：首先根据一快一慢指针找到环的节点个数，然后指针3首先走n步，指针4与指针3一起单步向前走。

面试题22 链表倒数第k个节点



面试题21 略

面试题20 表示数值的字符串 略

面试题19 正则表达式的匹配 略

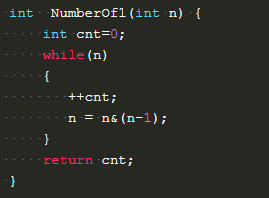
面试题18 删除链表的节点



面试题17 略

面试题16 略

面试题15 二进制中1的个数

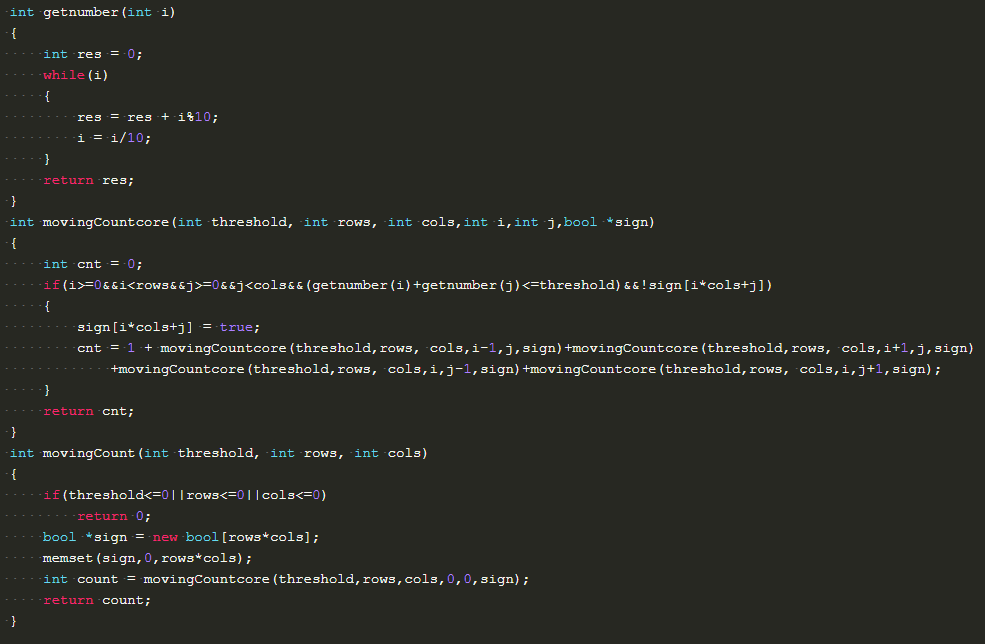


面试题14 剪绳子

贪婪算法

当剩下绳子长度>=5时，尽可能多的剪3，当剩余绳子为4时，剪出两个2。这样确保乘积最大

面试题13 机器人的运动范围



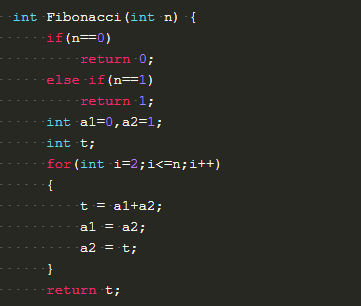
面试题12 矩阵的运动路径



上面两题都是回溯法的应用。

面试题11 略

面试题10 斐波那契数列



面试题9 略

面试题8 二叉树的下一个节点 分三种情况讨论

TreeLinkNode\* GetNext(TreeLinkNode\* pNode)

{

if(pNode==nullptr)

return nullptr;

if(pNode->next!=nullptr)

{

if(pNode->next->left==pNode)

{

if(pNode->right==nullptr)

return pNode->next;

else

{

TreeLinkNode \* leftnode = pNode->right;

while(leftnode->left!=nullptr)

{

leftnode = leftnode->left;

}

return leftnode;

}

}

else

{

if(pNode->right==nullptr)

{

TreeLinkNode \* father = pNode->next;

while(father->next!=nullptr&&father==father->next->right)

father = father->next;

if(father->next==nullptr)

return nullptr;

else

return father->next;

}

else

{

return pNode->right;

}

}

}

else

{

if(pNode->right==nullptr)

return nullptr;

else

{

TreeLinkNode \* leftnode = pNode->right;

while(leftnode->left!=nullptr)

{

leftnode = leftnode->left;

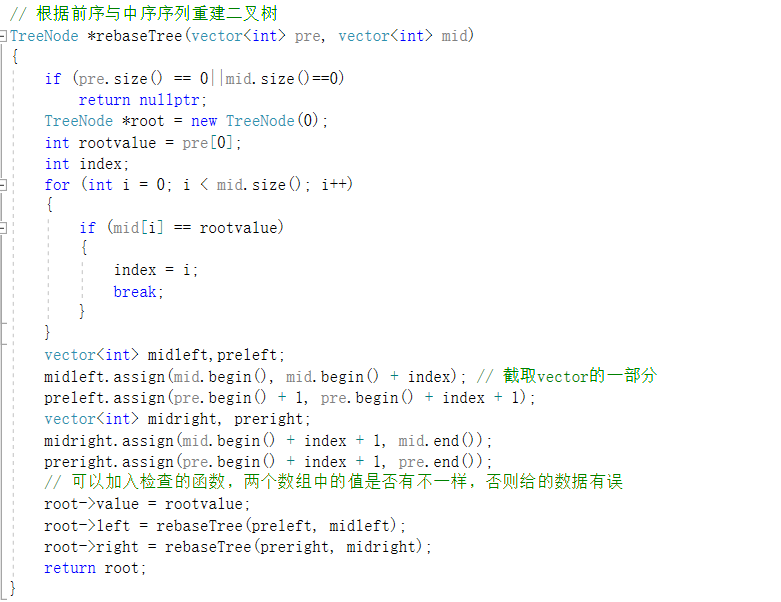
}

return leftnode;

}

}

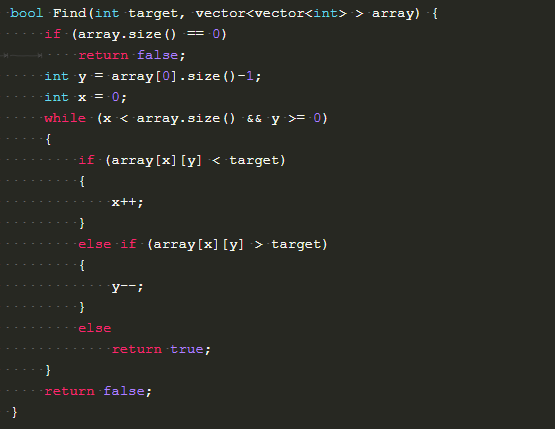
面试题7 重建二叉树



面试题6 从尾打印链表

面试题5 替换空格，从尾部替换

面试题4 二维数组的查找



面试题3 略

面试题2 略

面试题1 赋值运算符函数

String& String ::operator = (const String &str)

需要注意的点

# C++类知识

# 牛客网讨论区

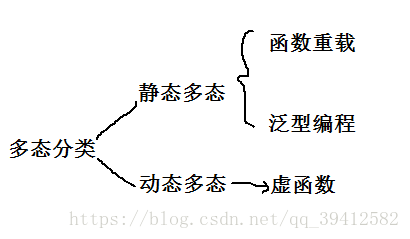
## 多态

多态是在不同继承关系的类对象，去调同一函数，产生了不同的行为，两个类里面都有一个函数且名字、参数、返回值均相同，然后我们通过调用函数来实现不同类对象完成不同的事件。

但是构成多态还有两个条件：

调用函数的对象必须是指针或者引用。

被调用的函数必须是虚函数，且完成了虚函数的重写。

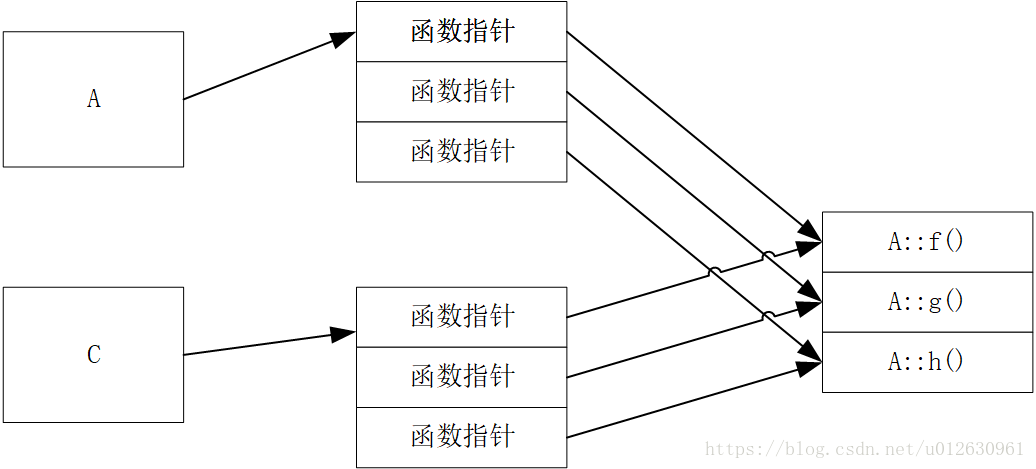


静态多态是编译器在编译期间完成的，编译器会根据实参类型来选择调用合适的函数，如果有合适的函数可以调用就调，没有的话就会发出警告或者报错

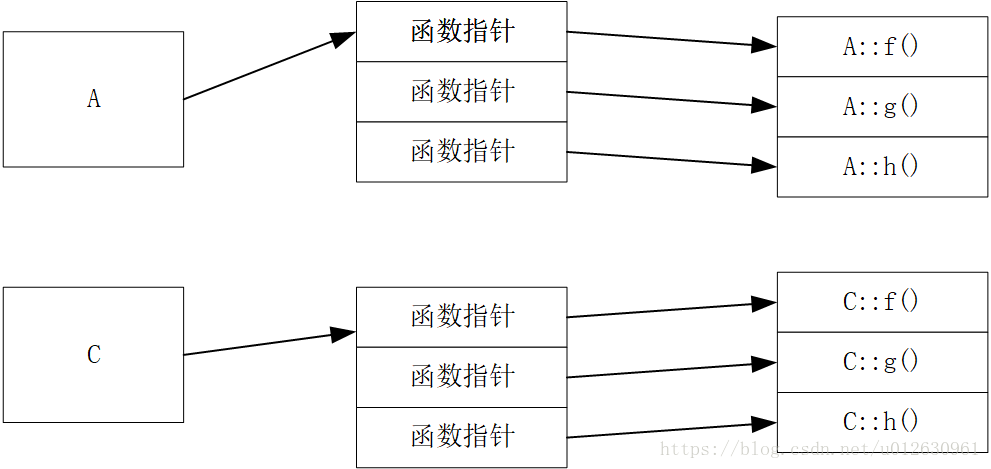
动态多态:它是在程序运行时根据基类的引用（指针）指向的对象来确定自己具体该调用哪一个类的虚函数。

在成员函数(必须为虚函数)的形参列表后面写上=0，则成员函数为纯虚函数，抽象类是指包括至少一个纯虚函数的类。当定义一个类里面如果存在虚函数，那么sizeof（classA） = 8,保存了虚函数表。

当派生类没有重写基类的虚函数



重写后就是这样的



## TCP与UDP

### 引言

网络协议是每个前端工程师都必须要掌握的知识，TCP/IP 中有两个具有代表性的传输层协议，分别是 TCP 和 UDP，本文将介绍下这两者以及它们之间的区别。

**想阅读更多优质文章请猛戳GitHub博客**

### 一、TCP/IP网络模型

计算机与网络设备要相互通信，双方就必须基于相同的方法。比如，如何探测到通信目标、由哪一边先发起通信、使用哪种语言进行通信、怎样结束通信等规则都需要事先确定。不同的硬件、操作系统之间的通信，所有的这一切都需要一种规则。而我们就把这种规则称为协议（protocol）。

TCP/IP 是互联网相关的各类协议族的总称，比如：TCP，UDP，IP，FTP，HTTP，ICMP，SMTP 等都属于 TCP/IP 族内的协议。

TCP/IP模型是互联网的基础，它是一系列网络协议的总称。这些协议可以划分为四层，分别为链路层、网络层、传输层和应用层。

* 链路层：负责封装和解封装IP报文，发送和接受ARP/RARP报文等。
* 网络层：负责路由以及把分组报文发送给目标网络或主机。
* 传输层：负责对报文进行分组和重组，并以TCP或UDP协议格式封装报文。
* 应用层：负责向用户提供应用程序，比如HTTP、FTP、Telnet、DNS、SMTP等。

在网络体系结构中网络通信的建立必须是在通信双方的对等层进行，不能交错。 在整个数据传输过程中，数据在发送端时经过各层时都要附加上相应层的协议头和协议尾（仅数据链路层需要封装协议尾）部分，也就是要对数据进行协议封装，以标识对应层所用的通信协议。接下去介绍TCP/IP 中有两个具有代表性的传输层协议----TCP 和 UDP。

### 二、UDP

UDP协议全称是用户数据报协议，在网络中它与TCP协议一样用于处理数据包，是一种无连接的协议。在OSI模型中，在第四层——传输层，处于IP协议的上一层。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

它有以下几个特点：

#### 1. 面向无连接

首先 UDP 是不需要和 TCP一样在发送数据前进行三次握手建立连接的，想发数据就可以开始发送了。并且也只是数据报文的搬运工，不会对数据报文进行任何拆分和拼接操作。

具体来说就是：

* 在发送端，应用层将数据传递给传输层的 UDP 协议，UDP 只会给数据增加一个 UDP 头标识下是 UDP 协议，然后就传递给网络层了
* 在接收端，网络层将数据传递给传输层，UDP 只去除 IP 报文头就传递给应用层，不会任何拼接操作

#### 2. 有单播，多播，广播的功能

UDP 不止支持一对一的传输方式，同样支持一对多，多对多，多对一的方式，也就是说 UDP 提供了单播，多播，广播的功能。

#### 3. UDP是面向报文的

发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付IP层。UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。因此，应用程序必须选择合适大小的报文

#### 4. 不可靠性

首先不可靠性体现在无连接上，通信都不需要建立连接，想发就发，这样的情况肯定不可靠。

并且收到什么数据就传递什么数据，并且也不会备份数据，发送数据也不会关心对方是否已经正确接收到数据了。

再者网络环境时好时坏，但是 UDP 因为没有拥塞控制，一直会以恒定的速度发送数据。即使网络条件不好，也不会对发送速率进行调整。这样实现的弊端就是在网络条件不好的情况下可能会导致丢包，但是优点也很明显，在某些实时性要求高的场景（比如电话会议）就需要使用 UDP 而不是 TCP。

从上面的动态图可以得知，UDP只会把想发的数据报文一股脑的丢给对方，并不在意数据有无安全完整到达。

#### 5. 头部开销小，传输数据报文时是很高效的。

UDP 头部包含了以下几个数据：

* 两个十六位的端口号，分别为源端口（可选字段）和目标端口
* 整个数据报文的长度
* 整个数据报文的检验和（IPv4 可选 字段），该字段用于发现头部信息和数据中的错误

因此 UDP 的头部开销小，只有八字节，相比 TCP 的至少二十字节要少得多，在传输数据报文时是很高效的

### 三、TCP

当一台计算机想要与另一台计算机通讯时，两台计算机之间的通信需要畅通且可靠，这样才能保证正确收发数据。例如，当你想查看网页或查看电子邮件时，希望完整且按顺序查看网页，而不丢失任何内容。当你下载文件时，希望获得的是完整的文件，而不仅仅是文件的一部分，因为如果数据丢失或乱序，都不是你希望得到的结果，于是就用到了TCP。

TCP协议全称是传输控制协议是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的RFC 793定义。TCP 是面向连接的、可靠的流协议。流就是指不间断的数据结构，你可以把它想象成排水管中的水流。

#### 1. TCP连接过程

如下图所示，可以看到建立一个TCP连接的过程为（三次握手的过程）:

**第一次握手**

客户端向服务端发送连接请求报文段。该报文段中包含自身的数据通讯初始序号。请求发送后，客户端便进入 SYN-SENT 状态。

**第二次握手**

服务端收到连接请求报文段后，如果同意连接，则会发送一个应答，该应答中也会包含自身的数据通讯初始序号，发送完成后便进入 SYN-RECEIVED 状态。

**第三次握手**

当客户端收到连接同意的应答后，还要向服务端发送一个确认报文。客户端发完这个报文段后便进入 ESTABLISHED 状态，服务端收到这个应答后也进入 ESTABLISHED 状态，此时连接建立成功。

这里可能大家会有个疑惑：为什么 TCP 建立连接需要三次握手，而不是两次？这是因为这是为了防止出现失效的连接请求报文段被服务端接收的情况，从而产生错误。

#### 2. TCP断开链接

TCP 是全双工的，在断开连接时两端都需要发送 FIN 和 ACK。

**第一次握手**

若客户端 A 认为数据发送完成，则它需要向服务端 B 发送连接释放请求。

**第二次握手**

B 收到连接释放请求后，会告诉应用层要释放 TCP 链接。然后会发送 ACK 包，并进入 CLOSE\_WAIT 状态，此时表明 A 到 B 的连接已经释放，不再接收 A 发的数据了。但是因为 TCP 连接是双向的，所以 B 仍旧可以发送数据给 A。

**第三次握手**

B 如果此时还有没发完的数据会继续发送，完毕后会向 A 发送连接释放请求，然后 B 便进入 LAST-ACK 状态。

**第四次握手**

A 收到释放请求后，向 B 发送确认应答，此时 A 进入 TIME-WAIT 状态。该状态会持续 2MSL（最大段生存期，指报文段在网络中生存的时间，超时会被抛弃） 时间，若该时间段内没有 B 的重发请求的话，就进入 CLOSED 状态。当 B 收到确认应答后，也便进入 CLOSED 状态。

#### 3. TCP协议的特点

* 面向连接

面向连接，是指发送数据之前必须在两端建立连接。建立连接的方法是“三次握手”，这样能建立可靠的连接。建立连接，是为数据的可靠传输打下了基础。

* 仅支持单播传输

每条TCP传输连接只能有两个端点，只能进行点对点的数据传输，不支持多播和广播传输方式。

* 面向字节流

TCP不像UDP一样那样一个个报文独立地传输，而是在不保留报文边界的情况下以字节流方式进行传输。

* 可靠传输

对于可靠传输，判断丢包，误码靠的是TCP的段编号以及确认号。TCP为了保证报文传输的可靠，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的字节发回一个相应的确认(ACK)；如果发送端实体在合理的往返时延(RTT)内未收到确认，那么对应的数据（假设丢失了）将会被重传。

* 提供拥塞控制

当网络出现拥塞的时候，TCP能够减小向网络注入数据的速率和数量，缓解拥塞

* TCP提供全双工通信

TCP允许通信双方的应用程序在任何时候都能发送数据，因为TCP连接的两端都设有缓存，用来临时存放双向通信的数据。当然，TCP可以立即发送一个数据段，也可以缓存一段时间以便一次发送更多的数据段（最大的数据段大小取决于MSS）

### 四、TCP和UDP的比较

#### 1. 对比

|  | **UDP** | **TCP** |
| --- | --- | --- |
| 是否连接 | 无连接 | 面向连接 |
| 是否可靠 | 不可靠传输，不使用流量控制和拥塞控制 | 可靠传输，使用流量控制和拥塞控制 |
| 连接对象个数 | 支持一对一，一对多，多对一和多对多交互通信 | 只能是一对一通信 |
| 传输方式 | 面向报文 | 面向字节流 |
| 首部开销 | 首部开销小，仅8字节 | 首部最小20字节，最大60字节 |
| 适用场景 | 适用于实时应用（IP电话、视频会议、直播等） | 适用于要求可靠传输的应用，例如文件传输 |

#### 2. 总结

* TCP向上层提供面向连接的可靠服务 ，UDP向上层提供无连接不可靠服务。
* 虽然 UDP 并没有 TCP 传输来的准确，但是也能在很多实时性要求高的地方有所作为
* 对数据准确性要求高，速度可以相对较慢的，可以选用TCP

**给大家推荐一个好用的BUG监控工具Fundebug，欢迎免费试用！**

### 参考文章与书籍

* 计算机网络简明教程及仿真实验
* TCP和UDP的区别
* 前端面试之道
* 一篇文章看明白 TCP/IP，TCP，UDP，IP，Socket 之间的关系
* 深入理解计算机网络
* TCP和UDP协议的特点和区别详解

## 红黑树

[jianshu.com/p/e136ec79235c](https://www.jianshu.com/p/e136ec79235c)

## C与C++区别

## 内存管理

<https://www.cnblogs.com/learning-zjx/p/10645659.html>

# 项目经历介绍

科大讯飞AI开发者方言种类识别：

pcm原始的音频信号，经过HCopy处理后，提取FB特征；

构建LSTM-Linear网络模型，长短时记忆网络，训练模型；

测试pcm作为输入，经过训练好的模型后，输出10维度结果，找到最大值，则是最佳识别结果。

学过的课程：矩阵论，数理统计，面向对象的程序设计，工业现场，光电检测原理，机器人学等

高通上海实习：

ISP流程：坏点矫正，lens shading 矫正，bayer降噪，Demosaic， 色彩矫正，gamma矫正，降噪与锐化等。

AF算法与tuning

武汉小弦：

图片->灰度化->LSD直线检测（斜率一致，长度条件去除干扰直线） 膨胀腐蚀，获取矩形的连通域，精定位，根据获取的坐标，精确查找边界。从而得到条形码区域。

由于条形码确定了物流单使用哪一个模板，标定每个模板下的地址、三段码等区域。经过仿射变换计算出地址的区域。三段码通过tesseract – ocr库识别。

地址区域需要经过分割，分割出的图片进行识别，c++再去调用python的结果。

电工基地：

本科阶段学习：PCB制作，Verilog HDL module语言。

全自动激光打孔机：

Opc与上位机的通讯。PLC程序编写。