1、在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

**class Solution {**

**public:**

**bool Find(int target, vector<vector<int> > array) {**

**int i=array.size()-1;**

**int j=0;**

**while((i>=0) && (j<array[0].size())){//用<=会发生内存溢出的错误，莫名其妙**

**if(array[i][j]==target) return true;**

**if(array[i][j]>target){i--;continue;}**

**if(array[i][j]<target){j++;continue;}**

**}**

**return false;**

**}**

**};**

**2.** 请实现一个函数，将一个字符串中的空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

**class Solution {**

**public:**

**void replaceSpace(char \*str,int length) {**

**//遍历一边字符串找出空格的数量**

**if(str==NULL||length<0)**

**return ;**

**int i=0;**

**int oldnumber=0;//记录以前的长度**

**int replacenumber=0;//记录空格的数量**

**while(str[i]!='\0')**

**{**

**oldnumber++;**

**if(str[i]==' ')**

**{**

**replacenumber++;**

**}**

**i++;**

**}**

**int newlength=oldnumber+replacenumber\*2;//插入后的长度**

**if(newlength>length)//如果计算后的长度大于总长度就无法插入**

**return ;**

**int pOldlength=oldnumber; //注意不要减一因为隐藏个‘\0’也要算里**

**int pNewlength=newlength;**

**while(pOldlength>=0&&pNewlength>pOldlength)//放字符**

**{**

**if(str[pOldlength]==' ') //碰到空格就替换**

**{**

**str[pNewlength--]='0';**

**str[pNewlength--]='2';**

**str[pNewlength--]='%';**

**}**

**else //不是空格就把pOldlength指向的字符装入pNewlength指向的位置**

**{**

**str[pNewlength--]=str[pOldlength];**

**}**

**pOldlength--; //不管是if还是elsr都要把pOldlength前移**

**}**

**}**

**};**

**3.** 输入一个链表，从尾到头打印链表每个节点的值。

**/\*\***

**\* struct ListNode {**

**\* int val;**

**\* struct ListNode \*next;**

**\* ListNode(int x) :**

**\* val(x), next(NULL) {**

**\* }**

**\* };**

**\*/**

**class Solution {**

**public:**

**vector<int> printListFromTailToHead(ListNode\* head) {**

**ListNode\* h;**

**vector<int> result;**

**stack<struct ListNode\*> nodes;//用栈来存储每个节点**

**while(head != NULL){**

**nodes.push(head);**

**head = head->next;**

**}**

**while(!nodes.empty()){**

**h = nodes.top();**

**result.push\_back(h->val);**

**nodes.pop();**

**}**

**return result;**

**}**

**};**

**或者:**

**struct ListNode {**

**int val;**

**struct ListNode \*next;**

**ListNode(int x) :**

**val(x), next(NULL) {**

**}**

**};**

**class Solution{**

**public:**

**vector<int> printListFromTailToHead(struct ListNode\* head){**

**vector<int> result;**

**struct ListNode\* pNode=head;**

**while(pNode!=NULL){**

**result.push\_back(pNode->val);**

**pNode=pNode->next;**

**}**

**reverse(result.begin(),result.end());//applying reverse()**

**return result;**

**}**

**};**

**4.** 输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

**/\*\***

**\* Definition for binary tree**

**\* struct TreeNode {**

**\* int val;**

**\* TreeNode \*left;**

**\* TreeNode \*right;**

**\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}**

**\* };**

**\*/**

**class Solution {**

**public:**

**TreeNode\* reConstructBinaryTree(vector<int> pre,vector<int> vin) {**

**int in\_size = vin.size();//获得序列的长度**

**if (in\_size == 0)**

**return NULL;**

**//分别存储先序序列的左子树，先序序列的右子树，中序序列的左子树，中序序列的右子树**

**vector<int> pre\_left, pre\_right, in\_left, in\_right;**

**int val = pre[0];//先序遍历第一个位置肯定是根节点node,取其值**

**//新建一个树结点，并传入结点值**

**TreeNode\* node = new TreeNode(val);//root node is the first element in pre**

**//p用于存储中序序列中根结点的位置**

**int p = 0;**

**for (p; p < in\_size; ++p){**

**if (vin[p] == val) //Find the root position in in**

**break; //找到即跳出for循环**

**}**

**for (int i = 0; i < in\_size; ++i){**

**if (i < p){**

**//建立中序序列的左子树和前序序列的左子树**

**in\_left.push\_back(vin[i]);//Construct the left pre and in**

**pre\_left.push\_back(pre[i + 1]);//前序第一个为根节点,+1从下一个开始记录**

**}**

**else if (i > p){**

**//建立中序序列的右子树和前序序列的左子树**

**in\_right.push\_back(vin[i]);//Construct the right pre and in**

**pre\_right.push\_back(pre[i]);**

**}**

**}**

**//取出前序和中序遍历根节点左边和右边的子树**

**//递归，再对其进行上述所有步骤，即再区分子树的左、右子子数，直到叶节点**

**node->left = reConstructBinaryTree(pre\_left, in\_left);**

**node->right = reConstructBinaryTree(pre\_right, in\_right);**

**return node;**

**}**

**};**

**5.** 用两个栈来实现一个队列，完成队列的Push和Pop操作。 队列中的元素为int类型。

**class Solution**

**{**

**//思路：用stack1来实现队列的push操作**

**//用stack2来实现队列的pop操作，当stack2为空时**

**//将stack1的数据全部压入stack2，等待队列的pop操作。**

**public:**

**void push(int node) {**

**stack1.push(node);**

**}**

**int pop() {**

**int result;**

**if(stack2.empty()){**

**while(!stack1.empty()){**

**stack2.push(stack1.top());**

**stack1.pop();**

**}**

**}**

**result = stack2.top();**

**stack2.pop();**

**//stack1的第一个元素出去了，恢复stack和stack2**

**while(!stack2.empty()){**

**stack1.push(stack2.top());**

**stack2.pop();**

**}**

**return result;**

**}**

**private:**

**stack<int> stack1;**

**stack<int> stack2;**

**};**

**6.** 把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。 输入一个非递减排序的数组的一个旋转，输出旋转数组的最小元素。 例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。 NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

**class Solution {**

**public:**

**int minNumberInRotateArray(vector<int> rotateArray) {**

**if(rotateArray.size()==0)**

**return 0;**

**int i;**

**for(i=0;i<rotateArray.size()-1;i++){**

**if(rotateArray[i+1] < rotateArray[i])**

**return rotateArray[i+1];**

**}**

**return 0;**

**}**

**};**

**7.** 大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项。

n<=39

**class Solution {**

**public:**

**int Fibonacci(int n) {**

**int f[40];**

**f[0]=0;**

**f[1]=1;**

**int i=0;**

**for(i=2;i<=n;i++){**

**f[i] = f[i-1]+f[i-2];**

**}**

**return f[n];**

**}**

**};**

**8.** 一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

**/\***

**可以考虑，小青蛙每一步跳跃只有两种选择：一是再跳一级阶梯到达第 i 级阶梯，此时小青蛙处于第 i-1 级阶梯；或者再跳两级阶梯到达第 i 级阶梯，此时小青蛙处于第 i-2 级阶梯。**

**于是，i 级阶梯的跳法总和依赖于前 i-1 级阶梯的跳法总数f(i-1)和前 i-2 级阶梯的跳法总数f(i-2)。因为只有两种可能性，所以，f(i)=f(i-1)+f(i-2);**

**依次类推，可以递归求出n级阶梯跳法之和。**

**\*/**

**class Solution {**

**public:**

**int jumpFloor(int number) {**

**int f[number];**

**f[0]=0;**

**f[1]=1;**

**f[2]=2;**

**int i=0;**

**for(i=3;i<=number;i++){**

**f[i] = f[i-1]+f[i-2];**

**}**

**return f[number];**

**}**

**};**

**9.** 一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

**class Solution {**

**public:**

**int jumpFloorII(int number) {**

**if (number <= 0)**

**return -1;**

**if (number == 1)**

**return 1;**

**return 2 \* jumpFloorII(number - 1);**

**}**

**};**

**10.** 我们可以用2\*1的小矩形横着或者竖着去覆盖更大的矩形。请问用n个2\*1的小矩形无重叠地覆盖一个2\*n的大矩形，总共有多少种方法？

**class Solution {**

**public:**

**int rectCover(int number) {**

**int f[1000000];**

**f[0]=0;**

**f[1]=1;**

**f[2]=2;**

**if(number>=3)**

**for(int i=3;i<=number;i++){**

**f[i] = f[i-1]+f[i-2];**

**}**

**return f[number];**

**}**

**};**

**11、**输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。