1.栈和队列的共同特点是（只允许在端点处插入和删除元素）

4.栈通常采用的两种存储结构是（线性存储结构和链表存储结构）

5.下列关于栈的叙述正确的是（D）

     A.栈是非线性结构B.栈是一种树状结构C.栈具有先进先出的特征D.栈有后进先出的特征

6.链表不具有的特点是（B）A.不必事先估计存储空间       B.可随机访问任一元素

C.插入删除不需要移动元素      D.所需空间与线性表长度成正比

7.用链表表示线性表的优点是（便于插入和删除操作）

8.在单链表中，增加头结点的目的是（方便运算的实现）

9.循环链表的主要优点是（从表中任一结点出发都能访问到整个链表）

10.线性表L＝（a1,a2,a3,……ai,……an），下列说法正确的是（D）

     A.每个元素都有一个直接前件和直接后件   B.线性表中至少要有一个元素

     C.表中诸元素的排列顺序必须是由小到大或由大到小

     D.除第一个和最后一个元素外，其余每个元素都有一个且只有一个直接前件和直接后件

11.线性表若采用链式存储结构时，要求内存中可用存储单元的地址（D）

A.必须是连续的 B.部分地址必须是连续的C.一定是不连续的 D.连续不连续都可以

12.线性表的顺序存储结构和线性表的链式存储结构分别是（随机存取的存储结构、顺序存取的存储结构）

13.树是结点的集合，它的根结点数目是（有且只有1）

14.在深度为5的满二叉树中，叶子结点的个数为（31）

15.具有3个结点的二叉树有（5种形态）

16.设一棵二叉树中有3个叶子结点，有8个度为1的结点，则该二叉树中总的结点数为（13）

17.已知二叉树后序遍历序列是dabec，中序遍历序列是debac，它的前序遍历序列是（cedba）

18.已知一棵二叉树前序遍历和中序遍历分别为ABDEGCFH和DBGEACHF，则该二叉树的后序遍历为（DGEBHFCA）

19.若某二叉树的前序遍历访问顺序是abdgcefh，中序遍历访问顺序是dgbaechf，则其后序遍历的结点访问顺序是（gdbehfca）

20.数据库保护分为：安全性控制、完整性控制、并发性控制和数据的恢复。

1. 在计算机中，算法是指（解题方案的准确而完整的描述）

2.在下列选项中，哪个不是一个算法一般应该具有的基本特征（无穷性）

说明：算法的四个基本特征是：可行性、确定性、有穷性和拥有足够的情报。

3. 算法一般都可以用哪几种控制结构组合而成（顺序、选择、循环）

4.算法的时间复杂度是指（算法执行过程中所需要的基本运算次数）

5. 算法的空间复杂度是指（执行过程中所需要的存储空间）

6. 算法分析的目的是（分析算法的效率以求改进）

7. 下列叙述正确的是（C）

A．算法的执行效率与数据的存储结构无关

B．算法的空间复杂度是指算法程序中指令（或语句）的条数

C．算法的有穷性是指算法必须能在执行有限个步骤之后终止

D．算法的时间复杂度是指执行算法程序所需要的时间

8.数据结构作为计算机的一门学科，主要研究数据的逻辑结构、对各种数据结构进行的运算，以及（数据的存储结构）

9. 数据结构中，与所使用的计算机无关的是数据的（C）

A．存储结构   B．物理结构     C．逻辑结构     D．物理和存储结构

10. 下列叙述中，错误的是（B）

A．数据的存储结构与数据处理的效率密切相关

B．数据的存储结构与数据处理的效率无关

C．数据的存储结构在计算机中所占的空间不一定是连续的

D．一种数据的逻辑结构可以有多种存储结构

11. 数据的存储结构是指（数据的逻辑结构在计算机中的表示）

12. 数据的逻辑结构是指（反映数据元素之间逻辑关系的数据结构）

13. 根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为（线性结构和非线性结构）

14. 下列数据结构具有记忆功能的是（C）A．队列B．循环队列C．栈D．顺序表

15. 下列数据结构中，按先进后出原则组织数据的是（B）

A．线性链表   B．栈            C．循环链表        D．顺序表

16. 递归算法一般需要利用（队列）实现。

17. 下列关于栈的叙述中正确的是（D）A．在栈中只能插入数据B．在栈中只能删除数据

C．栈是先进先出的线性表            D．栈是先进后出的线性表

20. 由两个栈共享一个存储空间的好处是（节省存储空间，降低上溢发生的机率）

21. 应用程序在执行过程中，需要通过打印机输出数据时，一般先形成一个打印作业，将其存放在硬盘中的一个指定（队列）中，当打印机空闲时，就会按先来先服务的方式从中取出待打印的作业进行打印。

22.下列关于队列的叙述中正确的是（C）A．在队列中只能插入数据 B．在队列中只能删除数据   C．队列是先进先出的线性表            D．队列是先进后出的线性表

23.下列叙述中，正确的是（D）A．线性链表中的各元素在存储空间中的位置必须是连续的

B．线性链表中的表头元素一定存储在其他元素的前面 C．线性链表中的各元素在存储空间中的位置不一定是连续的，但表头元素一定存储在其他元素的前面 D．线性链表中的各元素在存储空间中的位置不一定是连续的，且各元素的存储顺序也是任意的

24.下列叙述中正确的是（A）A．线性表是线性结构      B．栈与队列是非线性结构

C．线性链表是非线性结构                                 D．二叉树是线性结构

25. 线性表L＝（a1,a2,a3,……ai,……an），下列说法正确的是（D）

A．每个元素都有一个直接前件和直接后件      B．线性表中至少要有一个元素

C．表中诸元素的排列顺序必须是由小到大或由大到小D．除第一个元素和最后一个元素外，其余每个元素都有一个且只有一个直接前件和直接后件

26.线性表若采用链式存储结构时，要求内存中可用存储单元的地址（连续不连续都可以）

27. 链表不具有的特点是（B）A．不必事先估计存储空间            B．可随机访问任一元素

C．插入删除不需要移动元素            D．所需空间与线性表长度成正比

28. 非空的循环单链表head的尾结点（由p所指向），满足（p->next=head）

29.与单向链表相比，双向链表的优点之一是（更容易访问相邻结点）

30. 在（D）中，只要指出表中任何一个结点的位置，就可以从它出发依次访问到表中其他所有结点。A．线性单链表            B．双向链表            C．线性链表            D．循环链表

31. 以下数据结构属于非线性数据结构的是（C）A．队列      B．线性表C．二叉树      D．栈

32.树是结点的集合，它的根结点数目是（有且只有1）

33.具有3个结点的二叉树有（5种形态）

34. 在一棵二叉树上第8层的结点数最多是（128）注：2K-1

35. 在深度为5的满二叉树中，叶子结点的个数为（16）注：2n-1

36. 在深度为5的满二叉树中，共有（31）个结点。注：2n－1

37.设一棵完全二叉树共有699个结点，则在该二叉树中的叶子结点数为（350）

说明：完全二叉树总结点数为N，若N为奇数，则叶子结点数为（N+1）/2；若N为偶数，则叶子结点数为N/2。

38. 设有下列二叉树，对此二叉树中序遍历的结果是（B）

A．ABCDEF

B．DBEAFC

C．ABDECF

D．DEBFCA

39.已知二叉树后序遍历序列是dabec，中序遍历序列debac，它的前序遍历序列是（cedba）

40. 已知一棵二叉树前序遍历和中序遍历分别为ABDEGCFH和DBGEACHF，则该二叉树的后序遍历为（DGEBHFCA）

41.若某二叉树的前序遍历访问顺序是abdgcefh，中序遍历访问顺序是dgbaechf，则其后序遍历的结点访问顺序是（gdbehfca）

42. 串的长度是（串中所含字符的个数）

43.设有两个串p和q，求q在p中首次出现位置的运算称做（模式匹配）

44. N个顶点的连通图中边的条数至少为（N-1）

45.N个顶点的强连通图的边数至少有（N）

46.对长度为n的线性表进行顺序查找，在最坏情况下所需要的比较次数为（N）

47. 最简单的交换排序方法是（冒泡排序）

48.假设线性表的长度为n，则在最坏情况下，冒泡排序需要的比较次数为（n(n-1)/2）

49. 在待排序的元素序列基本有序的前提下，效率最高的排序方法是（冒泡排序）

50. 在最坏情况下，下列顺序方法中时间复杂度最小的是（堆排序）

51. 希尔排序法属于（插入类排序）

52. 堆排序法属于（选择类排序）

53. 在下列几种排序方法中，要求内存量最大的是（归并排序）

54. 已知数据表A中每个元素距其最终位置不远，为节省时间，应采用（直接插入排序）

55. 算法的基本特征是可行性、确定性、有穷性   和拥有足够的情报。

1.一个算法通常由两种基本要素组成：一是对数据对象的运算和操作，二是算法的控制结构。

1. 算法的复杂度主要包括时间复杂度和空间复杂度。

2. 实现算法所需的存储单元多少和算法的工作量大小分别称为算法的空间复杂度和时间复杂度。

3.所谓数据处理是指对数据集合中的各元素以各种方式进行运算，包括插入、删除、查找、更改等运算，也包括对数据元素进行分析。

4.数据结构是指相互有关联的数据元素的集合。

5.数据结构分为逻辑结构与存储结构，线性链表属于存储结构。

6.数据结构包括数据的逻辑结构和数据的存储结构。

7. 数据结构包括数据的逻辑结构、数据的存储结构以及对数据的操作运算。

8.数据元素之间的任何关系都可以用前趋和后继关系来描述。

9.数据的逻辑结构有线性结构和非线性结构两大类。

10.常用的存储结构有顺序、链接、索引等存储结构。

11. 顺序存储方法是把逻辑上相邻的结点存储在物理位置   相邻的存储单元中。

12. 栈的基本运算有三种：入栈、退栈与读栈顶元素。

13. 队列主要有两种基本运算：入队运算与退队运算。

14. 在实际应用中，带链的栈可以用来收集计算机存储空间中所有空闲的存储结点，这种带链的栈称为可利用栈。

15.栈和队列通常采用的存储结构是链式存储和顺序存储   。

16.当线性表采用顺序存储结构实现存储时，其主要特点是逻辑结构中相邻的结点在存储结构中仍相邻。

17. 循环队列主要有两种基本运算：入队运算与退队运算。每进行一次入队运算，队尾指针就进1 。

18.当循环队列非空且队尾指针等于对头指针时，说明循环队列已满，不能进行入队运算。这种情况称为上溢   。

19.当循环队列为空时，不能进行退队运算，这种情况称为下溢。

20. 在一个容量为25的循环队列中，若头指针front=16，尾指针rear=9，则该循环队列中共有 18 个元素。注：当rear<front时，元素个数＝总容量－（front－rear）；

当rear>front时，元素个数＝rear－front。

1.判断链表是否存在环型链表问题：判断一个链表是否存在环，例如下面这个链表就存在一个环：

例如N1->N2->N3->N4->N5->N2就是一个有环的链表，环的开始结点是N5这里有一个比较简单的解法。设置两个指针p1，p2。每次循环p1向前走一步，p2向前走两步。直到p2碰到NULL指针或者两个指针相等结束循环。如果两个指针相等则说明存在环。

struct link

{

    int data;

    link\* next;

};

bool IsLoop(link\* head)

{

    link\* p1=head, \*p2 = head;

     if (head ==NULL || head->next ==NULL)

     {

          return false;

     }

    do{

        p1= p1->next;

        p2 = p2->next->next;

    } while(p2 && p2->next && p1!=p2);

     if(p1 == p2)

          return true;

     else

          return false;

}

2,链表反转单向链表的反转是一个经常被问到的一个面试题，也是一个非常基础的问题。比如一个链表是这样的： 1->2->3->4->5 通过反转后成为5->4->3->2->1。最容易想到的方法遍历一遍链表，利用一个辅助指针，存储遍历过程中当前指针指向的下一个元素，然后将当前节点元素的指针反转后，利用已经存储的指针往后面继续遍历。源代码如下：

struct linka {

     int data;

     linka\* next;

};

void reverse(linka\*& head)

{

     if(head ==NULL)

          return;

     linka\*pre, \*cur, \*ne;

     pre=head;

     cur=head->next;

     while(cur)

     {

          ne = cur->next;

          cur->next = pre;

          pre = cur;

          cur = ne;

     }

     head->next = NULL;

     head = pre;

}

还有一种利用递归的方法。这种方法的基本思想是在反转当前节点之前先调用递归函数反转后续节点。源代码如下。不过这个方法有一个缺点，就是在反转后的最后一个结点会形成一个环，所以必须将函数的返回的节点的next域置为NULL。因为要改变head指针，所以我用了引用。算法的源代码如下：

linka\* reverse(linka\* p,linka\*& head)

{

     if(p == NULL || p->next == NULL)

     {

          head=p;

          return p;

     }

     else

     {

          linka\* tmp = reverse(p->next,head);

          tmp->next = p;

          return p;

     }

}

3,判断两个数组中是否存在相同的数字给定两个排好序的数组，怎样高效得判断这两个数组中存在相同的数字？

这个问题首先想到的是一个O(nlogn)的算法。就是任意挑选一个数组，遍历这个数组的所有元素，遍历过程中，在另一个数组中对第一个数组中的每个元素进行binary search。用C++实现代码如下：

bool findcommon(int a[],int size1,int b[],int size2)

{

     int i;

     for(i=0;i<size1;i++)

     {

          int start=0,end=size2-1,mid;

          while(start<=end)

          {

               mid=(start+end)/2;

               if(a[i]==b[mid])

                    return true;

               else if (a[i]<b[mid])

                    end=mid-1;

               else

                    start=mid+1;

          }

     }

     return false;

}

后来发现有一个 O(n)算法。因为两个数组都是排好序的。所以只要一次遍历就行了。首先设两个下标，分别初始化为两个数组的起始地址，依次向前推进。推进的规则是比较两个数组中的数字，小的那个数组的下标向前推进一步，直到任何一个数组的下标到达数组末尾时，如果这时还没碰到相同的数字，说明数组中没有相同的数字。

bool findcommon2(int a[], int size1, int b[], int size2)

{

     int i=0,j=0;

     while(i<size1 && j<size2)

     {

          if(a[i]==b[j])

                return true;

          if(a[i]>b[j])

               j++;

          if(a[i]<b[j])

               i++;

     }

     return false;

}

4,最大子序列问题：

给定一整数序列A1， A2，... An （可能有负数），求A1~An的一个子序列Ai~Aj，使得Ai到Aj的和最大

例如：

整数序列-2, 11, -4, 13, -5, 2, -5, -3, 12, -9的最大子序列的和为21。

对于这个问题，最简单也是最容易想到的那就是穷举所有子序列的方法。利用三重循环，依次求出所有子序列的和然后取最大的那个。当然算法复杂度会达到O(n^3)。显然这种方法不是最优的，下面给出一个算法复杂度为O(n)的线性算法实现，算法的来源于Programming Pearls一书。在给出线性算法之前，先来看一个对穷举算法进行优化的算法，它的算法复杂度为O(n^2)。其实这个算法只是对对穷举算法稍微做了一些修改：其实子序列的和我们并不需要每次都重新计算一遍。假设Sum(i, j)是A[i] ... A[j]的和，那么Sum(i, j+1) = Sum(i, j) + A[j+1]。利用这一个递推，我们就可以得到下面这个算法：

int max\_sub(int a[],int size)

{

     int i,j,v,max=a[0];

     for(i=0;i<size;i++)

     {

          v=0;

          for(j=i;j<size;j++)

          {

               v=v+a[j];//Sum(i, j+1) = Sum(i, j) + A[j+1]

               if(v>max)

                    max=v;

          }

     }

     return max;

}

那怎样才能达到线性复杂度呢？这里运用动态规划的思想。先看一下源代码实现：

int max\_sub2(int a[], int size)

{

     int i,max=0,temp\_sum=0;

     for(i=0;i<size;i++)

     {

          temp\_sum+=a[i];

          if(temp\_sum>max)

               max=temp\_sum;

          else if(temp\_sum<0)

               temp\_sum=0;

     }

     return max;

}

6,按单词反转字符串并不是简单的字符串反转，而是按给定字符串里的单词将字符串倒转过来，就是说字符串里面的单词还是保持原来的顺序，这里的每个单词用空格分开。例如：

Here is www.mianwww.com.cn

经过反转后变为：

如果只是简单的将所有字符串翻转的话，可以遍历字符串，将第一个字符和最后一个交换，第二个和倒数第二个交换，依次循环。其实按照单词反转的话可以在第一遍遍历的基础上，再遍历一遍字符串，对每一个单词再反转一次。这样每个单词又恢复了原来的顺序。

char\* reverse\_word(const char\* str)

{

     int len = strlen(str);

     char\* restr = new char[len+1];

     strcpy(restr,str);

     int i,j;

     for(i=0,j=len-1;i<j;i++,j--)

     {

          char temp=restr[i];

          restr[i]=restr[j];

          restr[j]=temp;

     }

     int k=0;

     while(k<len)

     {

          i=j=k;

          while(restr[j]!=' ' && restr[j]!='' )

               j++;

          k=j+1;

          j--;

          for(;i<j;i++,j--)

          {

               char temp=restr[i];

               restr[i]=restr[j];

               restr[j]=temp;

          }

     }

     return restr;

}

如果考虑空间和时间的优化的话，当然可以将上面代码里两个字符串交换部分改为异或实现。

例如将

          char temp=restr[i];

          restr[i]=restr[j];

          restr[j]=temp;

改为

          restr[i]^=restr[j];

           restr[j]^=restr[i];

          restr[i]^=restr[j];

7,字符串反转我没有记错的话是一道MSN的笔试题，网上无意中看到的，拿来做了一下。题目是这样的，给定一个字符串，一个这个字符串的子串，将第一个字符串反转，但保留子串的顺序不变。例如：

输入：第一个字符串: "This is mianwww's Chinese site: http://www.mianwww.com.cn/cn"

子串: "mianwww"

输出： "nc/nc.moc.mianwww.www//:ptth :etis esenihC s'mianwww si sihT"

一般的方法是先扫描一边第一个字符串，然后用stack把它反转，同时记录下子串出现的位置。然后再扫描一遍把记录下来的子串再用stack反转。我用的方法是用一遍扫描数组的方法。扫描中如果发现子串，就将子串倒过来压入堆栈。

最后再将堆栈里的字符弹出，这样子串又恢复了原来的顺序。源代码如下：

#include <iostream>

#include <cassert>

#include <stack>

using namespace std;

//reverse the string 's1' except the substring 'token'.

const char\* reverse(const char\* s1, const char\* token)

{

       assert(s1 && token);

       stack<char> stack1;

       const char\* ptoken = token, \*head = s1, \*rear = s1;

       while (\*head != '')

       {

              while(\*head!= '' && \*ptoken == \*head)

              {

                 ptoken++;

                 head++;

              }

              if(\*ptoken == '')//contain the token

              {

                 const char\* p;

                 for(p=head-1;p>=rear;p--)

                        stack1.push(\*p);

                 ptoken = token;

                 rear = head;

              }

              else

              {

                 stack1.push(\*rear);

                 head=++rear;

                 ptoken = token;

              }

       }

       char \* return\_v = new char[strlen(s1)+1];

       int i=0;

       while(!stack1.empty())

       {

              return\_v[i++] = stack1.top();

              stack1.pop();

       }

       return\_v[i]='';

       return return\_v;

}

int main(int argc, char\* argv[])

{cout<<"This is mianwww 's Chinese site: http://www.mianwww.com.cn/cn

";

       cout<<reverse("This is mianwww's Chinese site: http://www. mianwww.com.cn/cn"," mianwww ");

       return 0;

}

8, 删除数组中重复的数字问题：一个动态长度可变的数字序列，以数字0为结束标志，要求将重复的数字用一个数字代替，例如：

将数组 1,1,1,2,2,2,2,2,7,7,1,5,5,5,0 转变成1,2,7,1,5,0 问题比较简单，要注意的是这个数组是动态的。所以避免麻烦我还是用了STL的vector。

#include <iostream>

#include <vector>

using namespace std;

//remove the duplicated numbers in an intger array, the array was end with 0;

//e.g. 1,1,1,2,2,5,4,4,4,4,1,0 --->1,2,5,4,1,0

void static remove\_duplicated(int a[], vector<int>& \_st)

{

       \_st.push\_back(a[0]);

       for(int i=1;\_st[\_st.size()-1]!=0;i++)

       {

              if(a[i-1]!=a[i])

                 \_st.push\_back(a[i]);

       }

}

当然如果可以改变原来的数组的话，可以不用STL，仅需要指针操作就可以了。下面这个程序将修改原来数组的内容。

void static remove\_duplicated2(int a[])

{

       if(a[0]==0 || a==NULL)

              return;

       int insert=1,current=1;

       while(a[current]!=0)

       {

              if(a[current]!=a[current-1])

              {

                 a[insert]=a[current];

                 insert++;

                 current++;

              }

              else

                 current++;

       }

       a[insert]=0;

}

9,如何判断一棵二叉树是否是平衡二叉树问题：判断一个二叉排序树是否是平衡二叉树解决方案：

根据平衡二叉树的定义，如果任意节点的左右子树的深度相差不超过1，那这棵树就是平衡二叉树。

首先编写一个计算二叉树深度的函数，利用递归实现。

template<typename T>

static int Depth(BSTreeNode<T>\* pbs)

{

       if (pbs==NULL)

              return 0;

       else

       {

              int ld = Depth(pbs->left);

              int rd = Depth(pbs->right);

              return 1 + (ld >rd ? ld : rd);

       }

}

下面是利用递归判断左右子树的深度是否相差1来判断是否是平衡二叉树的函数：

template<typename T>

static bool isBalance(BSTreeNode<T>\* pbs)

{

       if (pbs==NULL)

              return true;

       int dis = Depth(pbs->left) - Depth(pbs->right);

       if (dis>1 || dis<-1 )

              return false;

       else

              return isBalance(pbs->left) && isBalance(pbs->right);

4.abstract class Something {  
private abstract String doSomething ();  
}

该段代码有错吗？

答案: 错。abstract的methods不能以private修饰。abstract的methods就是让子类implement(实现)具体细节的，怎么可以用private把abstract method封锁起来呢? (同理，abstract method前不能加final)。

5.看看下面的代码段错在哪里？

public class Something {  
void doSomething () {  
private String s = “”;  
int l = s.length();  
}  
}

答案: 错。局部变量前不能放置任何访问修饰符 (private，public，和protected)。final可以用来修饰局部变量  
(final如同abstract和strictfp，都是非访问修饰符，strictfp只能修饰class和method而非variable)。

6. 下面该段代码是否有错，若是有错错在哪里？  
abstract class Name {  
private String name;  
public abstract boolean isStupidName(String name) {}  
}  
答案: 错。abstract method必须以分号结尾，且不带花括号。