C++笔试面试（概念题集）

1>比较一下C++中static_cast和dynamic_cast的区别。

解答：dynamic_cast考虑到继承关系的限制，static_cast则进行强制类型转换。

2>struct 和class的区别。

解答：struct中的成员变量和成员函数默认访问权限是public，而class则为private。

3>当一个类A中没有生命任何成员变量和成员函数，那么这时的sizeof(A)的值，是否为零。为什么。

解答：肯定不为零，如果占用的内存为零，那么建立一个Class A[n]的数组，数组之间的对象就无法区别。

4>在8086汇编下，逻辑地址和物理地址是如何装换的？

解答：通用寄存器给出的地址，是段内偏移地址，相应段寄存器地址*10H+通用寄存器内存地址就得到想要的物理地址。

5>C++的多态实现方式。

解答：一是函数重载，运算符重载。二就是虚函数的应用。

6>const 与#define相比优点。

解答：const常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对#define只进行简单的替换。有些集成工具可以对const常量进行调试，但会对宏常量进行调试。

7>类成员函数的重载、覆盖和隐藏的区别。

解答：

成员函数被重载的特征：相同的函数名、相同的范围（在同一个类内）、参数不同（参数的个数或者类型）、virtual关键字可有可无

覆盖指派生类函数覆盖基类函数，特征：不同范围（基类和派生类）、函数名字相同、参数相同、基类必须有virtual关键字

隐藏是指派生类的函数屏蔽了基类同名函数，特征：如果派生类的函数与基类同名，但是参数不同，基类不论是否有virtual关键字，基类的同名函数都被隐藏。如果函数同名，参数也相同，基类没有virtual关键字，也隐藏基类的函数。

8>mian主函数执行完毕后，是否还有可能再执行一段代码？

解答：可以再执行一段代码。用_onexit注册一个函数即可执行。

9>不能做switch的参数类型？

解答：实型。

（大数据问题集）

1>海量日志数据，提取出某日访问百度次数最多的那个IP。

首先对于某日，提取出访问百度的日志中的IP取出来，逐个写入到一个大文件中。注意IP是32位的，最多的IP个数2^32，相当于4G的大小（不可以一次放入到主内存当中）。同样可以采用映射的方法，比如模1000，将整个大文件映射成1000个小文件，再找出每个小文件中频率较大的几个IP（采用hash_map进行频率统计）然后再在这1000多个甚至更多的IP中找出最大的那个，即是题目所要求的那个IP。

2>搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串长度1-255字节。

假定目前有1千万条记录（虽然总数是1千万条，但是如果去掉重复就只有不超过3百万条），请统计出最热门的10个查询串。

先对这批海量数据预处理，在O(N)的时间内用Hash表完成统计。之后借助堆这个数据结构，找出TOPK，时间复杂度为n*logK。

3>有一个1G大小的一个文件，里面每行是一个词，词的大小不超过16字节，内存限制为1M，返回频率最高的100个词。

方案：顺序读取文件，对于每个词x，取hash(x)%5000,然后按照该值存到5000个小文件当中。每个文件大概在200k左右。如果其中出现超过1M的文件则继续按照这个方法分割下去，直到每个文件都不超过1M为止。接下来对于每个文件统计其出现的词以及相应的频

率（采用hash_map实现），并得出频率最高的100个词（采用最小堆实现），并写入文件，这样差不多就有5000个文件，接下来要做的工作就是对这5000个文件进行归并排序，并且返回前100个词。

4>有10个文件，每个文件1G，每个文件的每行存放的都是用户query，每个文件的query都可能重复，要求按照query的频度进行排序。

顺序读取10个文件，按照hash(query)%10的结果写入到另外10个文件中，这样新生成的每个文件大小约为1G。找一台内存2G的机器，依次用hash_map来统计每个query出现的频率query_count，然后根据频率使用快速排序排序，将结果输出到文件。

5>给定a,b两个文件，各存放50亿个url，每个url各占64字节，内存限制为4G，让你找出两个文件中共同的URL。

方案1：可以估计每个文件大小5G*64=320G，远远大于内存。所以不能将其完全加载到内存当中，可以考虑分治的方法。

遍历文件a，对于每个URL求取hash(URL)%1000，然后根据所取得的值将URL分别存储到1000个小文件当中，这样每个文件约为300M。遍历b文件采用与a文件相同的处理方法。这样可能相同的URL都在同一个小文件当中，不同的小文件不可能有相同的URL。然后我们就把问题简化成找出这1000个小文件中相同的URL即可。

对于求每个小文件中相同的URL，我们可以把其中的一个小文件url存储到hash_set中，然后遍历另个小文件的每一个URL，如果存在在hash_set中，就把它加入到结果文件。

方案2：如果不考虑100%的正确率。可以采用位运算的方法。4G内存大概可以表示340G个bit。将其中一个文件映射到这340G个位。然后读取另外一个文件，检查对应的bit是否已经存在，如果存在则加入到结果文件。

6>在2.5亿个整数中找出不重复的整数，注意，内存不足

方案1:采用2-Bitmap(每个数分配2bit，00表示不存在，01表示出现过一次，10表示出现过多次，11不表示任何意义)进行，共需要2^32*2bit=1GB的内存。然后扫描这2.5亿个数，查看Bitmap中相对应位。输出对应位为01的整数即可。

方案2:分治的思想，将大文件划分成小文件，然后在小文件中找出不重复的整数，并排序，合并，去重复。

7>给40亿个不重复的unsigned int的整数，没排序过，然后再给出一个数，如何快速的判断这个数是否在这40亿个数当中。

申请512M的内存，一个bit位代表一个unsigned int值。读入40亿个数，设置相应的bit位，读入要查询的数值，看看它的bit位是否为1.

8>怎么在海量数据中找出重复次数最多的一个?

先做 hash，然后求模映射成小文件，（相同的数据必定映射到相同的文件当中），然后 hash_map 找出频率最高的那个数。对于分成多少个小文件，就会有多少个频率较高的数，然后在这些数当中找出频率最高的那个。

（算法题集一）

1>你让一些人为你工作了七天，你要用一根金条作为报酬。金条被分成七小块，每天给出一小块。如果你只能将金条分割两次，你怎样分给这些工人？

解答：将金条分割成1+2+4.这样这三个数字将可以组成1-7的任何数字。

2->用一种算法来颠倒一个链表的顺序。

解答：递归的解法：

```
Node *Reverse（Node *head）
{
    if(head==NULL) return head;
    if(head->next==NULL) return head;
    Node *p=Reverse(head->next);
    head->next->next=head;
    head->next=NULL;
```

```
    return p;
}
```

非递归算法：

```
Node *ReverseNonRecurisve（Node *head）
{
    if(head==NULL)  return head;
    Node *p=head;
    Node* pre=NULL;
    while(p->next!=NULL)
    {
        Node *pNext=p->next;
        p->next=pre;
        pre=p;
        p=pNext;
    }
    p->next=pre;
    return p;
}
```

3>算法通用字符串匹配

解法：

```
int match(char *str,char *ptn)
```

```
{
    if(*ptn=='\0') return 1;
    if(*ptn=='*')
    {
        do{
            if(match(str++,ptn+1))   return 1;
        }while(*str!='\0');
    }
    if(*str=='\0')   return 0;
    if(*str==*ptn || *ptn=='?')
          return match(str+1,ptn+1);
    return 0;
}
```

4>比较两个字符串，用O(n)时间和恒量空间

解答：

```
int strcmp(char *p1,char *p2)
{
    while(*p1!='\0' && *p2!='\0' &&*p1==*p2)
    {
        p1++;
        p2++;
```

```
    }
    if(*p1=='\0' &&*p2=='\0')    return 0;
   if(*p1=='\0')   return -1;
    if(*p2=='\0')   return 1;
    return (*p1-*p2);
}
```

5>不用乘法或者加法增加8倍。用类似的方法增加7倍。

解答：n<<3　　(n<<3)-n

6>判断整数序列是不是二叉树的后序遍历结果。

输入一个整数数组，判断该数组是不是某二元查找树的后序遍历结果。

解答：

```
void isPostorderResult(int a[],int left,int right,bool &flag)
{
    if(flag &&left<right)
    {
        int i=left;
        while(a[right]>a[i])   i++;
        for(int j=i;j<right;j++)
        {
```

```
            if(a[j]<a[right]) {flag=false; return;}
        }
        IsPosterResult(a,left,i-1,flag);
        IsPosterResult(a,i+1,right,flag);
    }
}
```

7>求二叉树中节点的最大距离（距离定义为两个节点之间边的个数）

解答：

```
int maxDistance(Node *root){
    int depth;
    return helper(root,depth);
}
int helper(Node* root,int &depth)
{
    if(root==NULL){
        depth=0; return 0;
    }
    int l,r;
    int maxleft=helper(root->left,l);
    int maxright=helper(root->right,r);
   depth=max(l,r)+1;
```

```
    return max(maxleft,max(right,l+r));
}
```

8>在一个字符串中找到第一个只出现一次的字符。例如输入abaccdeff，则输出b。

解答：

```
char FirstSingle(char *str)
{
    int c[255];   memset(c,0,sizeof(c));
    char *p=str;
    while(*p!='\0')
    {
        c[*p]++;
         p++;
    }
    p=str;
    while(*p!='\0'){
        if(c[*p]==1)     return *p;
    }
    return '\0';
}
```

1>二元查找树变成排序的双向链表

题目:输入一棵二元查找树，将该二元查找树转换成一个排序的双向链表。（要求不能创建任何新的结点，只是调整指针的指向）

树结点定义：

```
Struct BSTreeNode
{
Int m_value;
BSTreeNode *left;
BSTreeNode *right;
};
```

解题思路：

该题涉及树的中序遍历思想（二叉搜索树的中序遍历时一个有序的序列），一般采用递归的方式实现。递归算法（注意边界（即如何退出递归问题），以及何时进入递归）。

ANWER：

```
BSTreeNode * TreeToLinkedList(BSTreeNode *root)
{
BSTreeNode *head,*trail;
Helper(head,trail,root);
Return head;
```

```
}

Void Helper(BSTreeNode &*head,BSTreeNode &*trail,BSTreeNode
*root)
{
BSTreeNode *l,*r;
If(root==NULL)
{
    Head=NULL; trail=NULL;
    Return ;
}
Helper(head,l,root->left);
Helper(r,trail,root->right);
If(l!=NULL)
{
    l->right=root;
    root->left=l;
}
Else
{
    Head=root;
}
```

```
If(r!=NULL)
{
    r->left=root;
    root->right=r;
}
Else
{
    Trail=root;
}
}
```

2>设计包含min函数的栈

定义栈的数据结构，要求添加一个min函数，能够得到栈的最小元素。

要求函数min、push以及pop的时间复杂度都为O(1).

```
Struct MinStackElement{
Int data;
Int min;
};

Struct MinStack{
MinStackElement *data;
```

```
Int size;
Int top;
};
```

ANSWER:

```
MinStack MinStackInit(int MaxSize)
{
Minstack stack;
Stack.size=MaxSize;
Stack.top=0;
Stack.data=(MinStackElement*)malloc(sizeof(MinStackElement)*Maxsiz
e);
Return stack;
}

Void MinStackFree(MInStrack stack)
{
    Free(stack.data);
}

Void MinStackPush(Minstack stack,int d)
{
```

```
    If(stack.top==stack.size) ERROR(“out of Memory”);
    MinstackElement *p=stack.data[stack.top];
    P->data=d;
    p->min=(stack.top==0?d:stack.data[top-1]);
    if(p->min>d)
       p->min=d;
    top++;
}

Int MinStackPop(MinStack stack)
{
If(stack.top==0)
   ERROR(“error of memory”);
Return stack.data[--stack.top],data;
}

Int MinstackMin(Minstack stack)
{
If(stack.top==0)
   ERROR(“out Of memory”);
Return stack.data[stack.top-1].min;
}
```

3>求子数组的最大和

题目:输入一个数组，里面有正数也有负数

数组中连续一个或者多个整数组成一个子数组，每个子数组都有一个和。求所有子数组的最大值，要求时间复杂度O(n)。

解题思路:经典的动态规划题目，采用动态规划，就可能在o(n)时间内完成。

ANSWER:

```
Int MaxSubArray(int a[],int n)
{
If(n<=0) ERROR(“ERROR array Size”);
Int sum=0;
Int max=-(1<<31);
Int cur=0;
While(cur<n)
{
    Sum+=a[cur++];
    If(sum>max)
    {
        Max=sum;
```

```
}
If(sum<0)
    Sum=0;
}
Return max;
}
```

4>在二元树中找出和为某一值的所有路径

题目:输入一个整数和一棵二元树。从树根结点开始往下访问一直到叶子结点所经过的结点形成一条路径。打印出与输入整数相等的所有路径。

树结点结构定义:

```
Struct BinaryTreeNode{
Int m_value;
BinaryTreeNode *left;
BinaryTreeNode *right;
};
```

ANWSER:

```
Void PrintPaths(TreeNode *root,int sum)
```

```
{
Int path[MAX_HEIGHT];
Helper(root,sum,path);
}

Void Helper(TreeNode *root,int &sum,int &path[],int &cur)
{
Path[cur++]=root.data;
Sum-=root.data;
If(root->left==NULL &&root->right==NULL)
{
    If(sum==0)
        Print("The road");
}
Else
{
    If(root->left!=NULL)
        Helper(root->left,sum,path,cur);
    If(root->right!=NULL)
        Helper(root->right,sum,path,cur);
}
Cur--;
```

Sum+=root.data;

}

5>查找最小的K个元素

题目:输入n个整数，输出其中最小的K个。

解题思路:建立一个只能包含K个元素的大顶堆，遍历输入的n个数，若堆中包含的元素的个数小于K个，则加入到堆中。若包含的元素个数已经等于K个。那么喝堆顶的元素比较，若小于堆顶的元素则去掉堆顶元素，该元素入堆，否则直接丢弃。

6>给出两个单向链表的头指针，比如h1、h2,判断这两个个链表是否相交。

为了简化问题我们假设两个链表均不带环。

问题扩展：

如果链表可能存在环？如果需要求出两个链表相交的第一个结点？

ANWSER:

Struct Node{

Int data;

Struct Node *next;

};

```
/*there is no cycle*/
Int isJoinedSimple(Node *h1,Node *h2)
{
While(h1->next!=NULL)
    H1=h1->next;
While(h2->next!=NULL)
    H2=h2->next;
Return h1==h2;
}

/*it maybe exits cycle*/
Int isJoined(Node *h1,Node *h2)
{
Node *cycle1=HasCycle(h1);
Node *cycle2=HasCycle(h2);
If(cycle1==NULL && cycle2==NULL)
    isJoinedSimple(h1,h2);                  //无环的情况，采用简单的判断
方法

if((cycle1==NULL && cycle2!=NULL)||(cycle2==NULL &&cycle1!+NULL))
    return 0;
```

```
Node *p=cycle1;
While(true)
{
    If(p==cycle2 ||p->next==cycle2)
       Return 1;
    P=p->next->next;
    Cycle1=cycle1->next;
    If(p==cycle1)     return 0;
}
}

Node * HasCycle(Node *h)
{
Node *p=h,q=h;
While(p!=NULL && q->next!=NULL)
{
    P=p->next;
    Q=q->next;
    If(p==q)
    {
       Return p;
}
```

```
}
Return NULL;
}
```

7>用一种算法逆序一个链表。

ANWSER:

```
Node *Reverse(Node *head)
{
If(head==NULL) return head;
If(head->next==NULL) return head;
Node *p=Reverse(head->next);
Head->next->next=head;
Head->next=NULL;
Return p;
}

Node *ReverseNon(Node *head)
{
If(head==NULL)   return head;
Node *p=head;
Node *pre=NULL;
While(p->next!=NULL)
```

```
{
    Node * tmp=p->pNext;
    P->next=pre;
    Pre=p;
    P=tmp;
}
p->next=pre;
return p;
}
```

8>通用字符串匹配算法

ANWSER:

```
Int isMatch(char *str,char *ptn)
{
If(*ptn==’\0’)   return 1;
If(*ptn==’*’)
{
    Do{
      If(isMatch(str++,ptn+1))
           Return 1;
}while(*str!=’\0’);
}
```

```
If(*str==’\0’)   return 0;
If(*str==*ptn ||*ptn==’?’)
   Return isMatch(str+1,ptn+1);
Return 0;
}
```

9>反转一个字符串

ANWSER:

```
Void   ReverseString(char *str,int n)
{
Char *p=str+n-1;
While(str<p)
{
    Char ch=*str;
    *str=*p
    *p=ch;
    Str++; p--;
}
}
```

10>比较两个字符串，用o(n)时间和恒量空间。

ANWSER：

```
Int strcmp(char *p1,char *p2)
{
While(*p1!='\0' && *p2!='\0' &&*p1==*p2)
{
    P1++;p2++;
}
If(*p1=='\0' && *p2=='\0')   return 0;
If(*p1=='\0')   return -1;
If(*p2=='\0')   return 1;
Return (*p1-*p2);
}
```

11>判断整数序列是不是二元查找树的后序遍历结果

题目:输入一个整数数组，判断该数组是不是某二元查找树的后续遍历结果。

ANWSER:

```
Int isPostorderResult(int a[],int n)
{
    Return helper(a,0,n-1);
}

Int helper(int a[],int s,int e)
```

```
{
If(e==s)   return 1;
Int i=e-1;
While(a[e]<a[i] && i>=s) i--;
If(!helper(a,i+1,e-1))
    Return 0;
Return helper(a,s,i);
}
```

12>求二叉树中结点的最大距离

ANWSER:

```
Int maxDistance(Node *root)
{
Int depth;
Return helper(root,depth);
}

Int helper(Node* root,int &depth)
{
If(root==NULL)
{
    Depth=0;   return 0;
```

```
}
Int ld,rd;
Int maxleft=helper(root->left,ld);
Int maxright=helper(root->right,rd);
Depth=max(ld,rd)+1;
Return max(maxleft,max(maxright,ld+rd));
}
```

13>输出一个单向链表中倒数第K个结点。

ANWSER:

```
Struct ListNode{
Int m_value;
ListNode *m_pNext;
};

ListNode *LastK(ListNode *head,int k)
{
If(k<0)   exit(0);
Node *p=head,*pk=head;
For(;k>0;k--)
{
    If(pk->next!=NULL)
```

```
        Pk=pk->next;
    Else
        Return NULL;
}
While(pK->next!=NULL)
{
    P=p->next;
    pK=pK->next;
}
Return p;
}
```

14->输入一个已经按照升序排序过的数组和一个数字，在数组中查找两个数，使他们的和正好是输入的那个数字。输出1对即可以。

ANSWER:

```
Void Find2Number(int a[],int n,int dest)
{
Int *start=a,*end=a+n-1;
Int sum=*start+*end;
While(sum!=dest && start<end)
{
If(sum<dest)
```

```
{
    Sum-=*start;
    Sum+=*++start;
}
Else
{
    Sum-=*end;
    Sum+=*--end;
}    }
If(sum==dest) cout<<*start<<" "<<*end<<endl;
}
```

15->输入一棵二元查找树，将该树转换位它的镜像

ANSWER：

```
Struct BSTreeNode{
Int m_value;
BSTreeNode *left;
BSTreeNode *right;
};
Void Mirror(BSTreeNode *root)
{
If(root==NULL)   return;
```

```
Swap(root->left,root->right);
Mirror(root->left);
Mirror(root->left);
}
```

16->输入一个二元树，从上往下打印每个结点，同一层按照从左往右的顺序打印。

ANSWER:

引入队列可以很简单的就解决这一道题。

17->在字符串中找到第一次值出现一次的字符。

ANSWER:

```
Char FirstSingle(char *str)
{
Int c[255];
Memset(c,0,sizeof(a));
Char *p=str;
While(*p!='\0')
{
    c[*p]++;
    p++;
}
```

```
P=str;
While(*p!='\0')
{
    If(c[*p]==1)
        Return *p;
}
Return '\0';
}
```

18->n 个数字(0,1,….,n-1)形成一个圆圈，从数字0开始，每次从这个圈中删除第m个数字，当一个数字删除后，从被删除数字的下一个继续删除第m个数字。求出这个圆圈中剩下的最后一个数字。

ANSWER:

答案比较简单，但是整个过程比较难理解。

```
Int joseph(int n,int m)
{
Int fn=0;
For(int i=2;i<=n;i++)
    Fn=(fn+m)%i;
Return fn;
}
```

19->输入一个表示整数的字符串，把字符串转换成整数并输出

ANSWER:

```
Int atoi(char *str)
{
Int   neg=0;
Char *p=str;
If(*p==’-’)
{
    P++;
    Neg=1;
}else if(*p==’+’)
{
    P++;
}
Int result=0;
While(*p!=’\0’)
{
    If(*p>=’0’ && *p<=’9’)
{
    Num=num*10+(*p-‘0’);
}
Else
```

```
{
    ERROR(“illegal number”);
}
P++;
}
Return num;
}
```

20->输入两个整数n,m，从数列1,2,3……,n中随意取出几个数，使其和等于m。要求求出其中所有的可能组合。

ANSWER:

```
Void FindCombination(int n,int m)
{
If(n>m) findCombination(m,m);
Int aux[n];
Memset(aux,0,sizeof(aux));
Helper(m,0,aux,n);
}

Void Helper(int dest,int index,int aux[],int n)
{
```

```
If(dest==0)
{cout<<aux<<endl;}
If(dest<0 || index==n) return;
Helper(dest,index+1,aux,n);
Aux[index]=1;
Helper(dest-index-1,index+1,aux,n);
Aux[index]=0;
}
```

21->合并一个链表

ANSWER:

```
Node *merge(Node *h1,Node *h2)
{
If(h1==NULL) return h2;
If(h2==NULL) return h1;

Node* head;
If(h1->data>h2->data)
{
    Head=h2;h2=h2->next;
}
Else
```

```
{
    Head=h1;h1=h1->next;
}
Node *p=head;
While(h1!=NULL &&h2!=NULL)
{
    if(h1->data>h2->data)
    {
       p->next=h2;h2=h2->next;p=p->next;
}
Else
{
    p->next=h1;h1=h1->next;p=p->next;
}
}
If(h1!=NULL) p->next=h1;
If(h2!=NULL) p->next=h2;
Return head;
}
```

22->写一个函数，实现在字符串中找出连续最长的数字串，并把这个串的长度返回。

ANSWER:

```
Int continumax(char* outputstr,char* inputstr)
{
Int len=0;
Char* pstart=NULL;
Int max=0;
While(true)
{
    If(*inputstr>=’0’ &&*inputstr<=’9’)
       Len++;
    Else
    {
        If(len<max)
           Pstart=inputstr-len;
        Len=0;
}
If(*inputstr++=’\0’)   break;
}
For(int i=0;i<len;i++)
{
    *outputstr++=pstart++;
    *outputstr=’\0’;
```

```
    Return max;
}
}
```

23->整数二进制表示中1的个数

ANWSER:

```
Int countof1(int n)
{
Int c=0;
While(n!=0)
{
    N=n&(n-1);
    C++;
}
Return c;
}
```

29->栈的push、pop序列

ANWSER:

```
Int isPopSeries(int push[],int pop[],int n)
{
Stack<int> st;
```

```
Int i1=0,i2=0;
St.push(push[i1]);
While(i2<n)
{
    While(st.empty()||st.top()!=pop[i2])
    {
       If(i1<n)
          St.push(push[++i1]);
       Else
          Return 0;
       While(!=st.empty()&&st.top()==pop[i2])
{
    St.pop();   i2++;
}
}
}
Return 1;
}
```

30->求一个数组的最长递减子序列。比如{9、4、3、2、5、4、3、2}的最长递减子序列为{9、5、4、3、2}

ANSWER:

```
Int [] FindDecreasing(int[] a)
{
Int *Ds=new int[a.size()]();
Int dsl=0;
Int lastIndex=0;
For(int i=0;i<a.length;i++)
{
    Int s=0,t=dsl-1;
    While(s<=t)
    {
       Int m=s+(s-t)/2;
       If(Ds[m]<a[i])
          T=m-1;
       Else
          S=m+1;
    Ds[s]=a[i];
    If(s>dsl){dsl=s+1,lastIndex=i}
}
}
For(int i=lastIndex,j=dsl-1;i>=0&&j>=0;i--)
{
   If(a[i]==Ds[j])   j--;
```

```
    Else if(a[i]<Ds[j] &&a[i]>Ds[j+1])
        Ds[j--]=a[i];
}
Return copy(Ds,0,dsl);
}
```

31->一个整数数组，长度为n，将其分为m份，使各分的和相等，求m的最大值。

例如{3,、2、4、3、6}可以分成{3、2、、4、3、6}m=1

{3、6}{2、4、3}m=2 {3、3}{2、4}{6}m=3

ANWSER:

```
Int maxShares(int a[],int n)
{
    Int sum=0;
    For(int i=0;i<n;i++) sum+=a[i];
    For(int m=n;m>=2;m--)
    {
        If(sum%m!=0) continue;
        Int *aux=new int[n]();
        If(TestShares(a,n,m,sum/m,aux,1))    return m;
}
Return 1;
```

```
}

Int TestShares(int a[],int n,int m,int Dest,int aux[],int id)
{
If(Dest==0)
{
   Id++;   if(id==m+1) return 1;
}
For(int i=0;i<n;i++)
{
   If(aux[i]!=0) continue;
    Aux[i]=id;
    If(testShares(a,,n,m,Dest-a[i],aux,id))
       Return 1;
    Aux[i]=0;
}
}
```

32->旋转数组的value值查找问题

ANWSER:

```
Int ShiftedBinarySearch(int a[],int k)
{
```

```
    Return Helper(a,k,0,n-1);
}
Int Helper(int a[],int k,int s,int t)
{
If(s>t)   return -1;
Int m=s+(s-t)/2;
If(a[m]==k) return m;
Else if(a[s]>=k &&k>a[m])   return Helper(a,k,s,m-1);
Else return helper(a,k,m+1,t);
}
```

33->调整数组顺序是奇数位于偶数的前面

ANWSER：

```
Void partition(int a[],int n)
{
Int i=0;
Int j=n-1;
While(i<j)
{
    While(i<j &&a[i]&1==0) i++;
    While(i<j &&a[j]&1==1) j--;
    Swap(a[i],a[j]);
```

```
}
}
```

1> 编写strcpy函数，已知函数原型char*strcpy(char* strDest,char* strSrc)

ANSWER:

```
Chat* strcpy(char* strDest,char* strSrc)
{
If(strSrc==NULL)  return NULL;
Char*ch1=strSrc,*ch2=strDest;
While(*ch1!='\0')
{
    *ch2++=*ch1++;
}
*ch2='\0';
Return strDest;
}
```

2> 用递归的方法判断整数组a[N]是不是升序排序

ANSWER：

```
Boolean isAscending(int a[])
{
```

```
    ReturnisAscending(a,0);
}
Bool isAscending(int a[],int start)
{
    Returnstart==length-1 ||isAscending(a,start+1);
}
```

3>　删除字符串中的数字并压缩字符串

```
Char * partition(const char *str)
{
Char* i=str;
Char* j=str;
While(*i!='\0')
{
    If(*i>'9' || *i<'0')
        *j++=*i++;
    Else
         *i++;
}
*j='\0';
Return str;
}
```

4>　函数将字符串的字符’*’移到字符串的前部，前面的非’*’字符移到字符串的后面部分。

ANSWER：

```
Int partitionStr(char a[])
{
Int count=0;
Inti=a.length-1,j=a..length-1;
While(i>=0)
{
    If(a[i]!=’*’)
    {
        Swap(a,i--,j--)
}
Else
{
    i--;
    count++;
}
}
Return count;
```

```
}
```

5> 已知一个字符串，寻找字符串sub在原字符串中出现的次数。

ANSWER:

```
Int count_of_substr(chat *str,char *sub)
{
Int n=strlen(sub);
Int count=0;
Char *p=stt;
While(p!='\0')
{
    If(strcmp(p,sub,n))
        Count++;
    P++;
}
Return count;
}
```

6> 一个最小堆，也是完全二叉树，用按层遍历数组表示。

1， 求节点a[n]的子节点的访问方式

2， 插入一个节点的程序void add_element(int *a,int size,int val);

3， 删除最小节点的程序

ANWSER:

```
Void add_element(int *a,int size,int val)
{
A[size]=val;
Int p=size/2-1;
Int c=size;
While(p>=0)
{
    If(a[p]<a[c])
        Break;
    Else
    {
        A[c]=a[p];
        C=p;
        P=(p-1)/2;
}
}
A[c]=val;
}

Void Del(int *a,int size)
{
```

```
A[0]=a[size-1];
Int val=a[0];
Size--;
Int p=0;
Int c=2*p+1;
While(c<=size-1)
{
    If(c<size-1 &&a[c]>a[c+1])
        C++;
    If(a[p]<=a[c])
        Break;
    Else
    {
        A[p]=a[c];
        P=c; c=2*p+1;
}
}
A[p]=val;
}
```

7>　字符串的组合

ANWSER:

```
Void Combination_m(chat *pStr,intm,vector<char> &result)
{
If(pStr==NULL||(*pStr=='\0' &&m!=0))
    Return ;
If(m==0)
    Cout<<result<<endl;

Result.push_back(*pStr);
Combination_m(pStr+1,m-1,result);
Result.pop_back();
Combination_m(pStr+1,m,result);
}

Void Combination(char *pStr)
{
If(pStr==NULL||*pStr=='\0')
    Return ;
Intlen=strlen(str);
For(inti=1;i<=len;i++)
{
    Vector<char> result;
    Combination_m(pStr,I,resulr);
```

```
}

}
```