

2. 不同条件下排序方法的选择

(1) 若 n 较小 (如 $n \leq 50$)，可采用直接插入或直接选择排序。

当记录规模较小时，直接插入排序较好。否则因为直接选择移动的记录数少于直接插入，应选直接选择排序为宜。

(2) 若文件初始状态基本有序 (指正序)，则应选用直接插入排序、冒泡排序或随机的快速排序为宜。

(3) 若 n 较大，则应采用时间复杂度为 $O(nlgn)$ 的排序方法 (快速排序、堆排序或归并排序)。

快速排序被认为是目前基于比较的内部排序中最好的方法。当待排序的关键字随机分布时，快速排序的平均时间最短。

堆排序所需的辅助空间少于快速排序，并且不会出现快速排序可能出现的最坏情况。这两种排序都是不稳定的。

若要求排序稳定，则可选用归并排序。但本章介绍的从单个记录起进行两两归并的排序算法并不值得提倡，通常可以将它和直接插入排序结合在一起使用。先利用直接插入排序求得较长的有序子文件，然后再两两归并之。因为直接插入排序是稳定的，所以改进后的归并排序仍是稳定的。

13.9 时间复杂度

面试例题 1：定义了如下类和有序表关键字序列为 b c d e f g q r s t，则在二分法查找关键字 b 的过程中，先后进行比较的关键字依次是多少？[中国某互联网公司 2009 年 11 月面试题]

- A. f c b B. f d b C. g c b D. g d b

解析：二分法查找是指已知有序队列中找出与给定关键字相同的数的具体位置。原理是分别定义三个指针 low、high、mid，分别指向待查元素所在范围的下界和上界及区间的中间位置，即 $mid = (low + high) / 2$ ，让关键字与 mid 所指的数比较，若相等则查找成功并返回 mid，若关键字小于 mid 所指的数则 $high = mid - 1$ ，否则 $low = mid + 1$ ，然后继续循环直到找到或找不到为止。下面代码是二分法的 C++ 实现：

```
#include <stdio.h>
#include <iostream>
using namespace std;

#define MAXSIZE 10
```

```

typedef struct{
    int list[MAXSIZE];
    int length;
}List;

int dichotomy_search(List s,int k)
{
    int low,mid,high;
    low=0;
    high=s.length-1;
    mid=(low+high)/2;
    while(high>=low)
    {
        cout << mid << endl;
        if(s.list[mid]>k){//turn to the left part

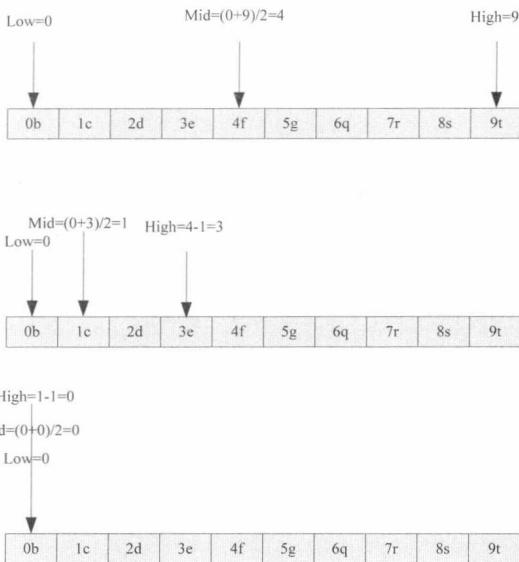
            high=mid-1;
            mid=(low+high)/2;
        }
        else if(s.list[mid]<k){ //turn to the right part
            low=mid+1;
            mid=(low+high)/2;
        }
        else
            return(mid+1);//The key has been searched
    }
    return 0;//no such key
}

int main(int argc,char **argv)
{
    List s;
    int i,k,rst;
    int a[MAXSIZE]={1,3,6,12,15,19,25,32,38,87};
    for(i=0;i<MAXSIZE;i++){
        s.list[i]=a[i];
    }
    s.length=MAXSIZE;
    printf("Input key number:");
    scanf("%d",&k);

    rst=dichotomy_search(s,k);
    if(rst==0){
        printf("Key:%d is not in the list!\n",k);
    }
    else{
        printf("The key is in the list,position is:%d \n",rst);
    }
    return 0;
}

```

对于本题而言，要比较三个关键字，分别是 f、e、b。具体情况如下图所示。



答案：A

面试例题 2： Which of the choices below correctly describes the amount of time used by the following code? (下面哪个选项正确地描述了代码运行的调度次数?) [美国著名软件公司 M2009 年 10 月面试题]

```
n=10;
for(i=1; i<n; i++)
    for(j=1; j<n; j+=n/2)
        for(k=1; k<n; k=2*k)
            x = x +1;
```

- A. $\Theta(n^3)$ B. $\Theta(n \log n)$ C. $\Theta(n(\log n)^2)$ D. $\Theta(n \log n)$

解析：本题考量面试者对时间复杂度的理解。本题涉及如下概念：

1) 时间频度

一个算法执行所耗费的时间，从理论上是不能算出来的，必须上机运行测试才能知道。但不可能也没有必要对每个算法都上机测试，只需知道哪个算法花费的时间多，哪个算法花费的时间少就可以了。并且一个算法花费的时间与算法中语句的执行次数成正比例，哪个算法中语句执行次数多，它花费时间就多。一个算法中的语句执行次数称为语句频度或时间频度。记为 $T(n)$ 。

2) 时间复杂度

在刚才提到的时间频度中， n 称为问题的规模，当 n 不断变化时，时间频度 $T(n)$ 也会不断变化。但有时我们想知道它变化时呈现什么规律。为此，我们引入时间复杂度概念。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数，用 $T(n)$ 表示，若有某个辅助函数 $f(n)$ ，使得当 n 趋近于无穷大时， $T(n)/f(n)$ 的极限值为不等于零的常数，则称 $f(n)$ 是 $T(n)$ 的同数量级函数。记作 $T(n)=O(f(n))$ ，称 $O(f(n))$ 为算法的渐进时间复杂度，简称时间复杂度。

在各种不同算法中，若算法中语句执行次数为一个常数，则时间复杂度为 $O(1)$ ，另外，在时间频度不相同时，时间复杂度有可能相同，如 $T(n)=n^2+3n+4$ 与 $T(n)=4n^2+2n+1$ ，它们的频度不同，但时间复杂度相同，都为 $O(n^2)$ 。

按数量级递增排列，常见的时间复杂度有：

```
常数阶 O(1)
对数阶 O(log 2 n)
线性阶 O(n)
线性对数阶 O(n log 2 n)
平方阶 O(n^2)
立方阶 O(n^3)
...
k 次方阶 O(n^k)
指数阶 O(2^n)
```

随着问题规模 n 的不断增大，上述时间复杂度不断增大，而算法的执行效率不断降低。

3) 算法的时间复杂度

若要比较不同的算法的时间效率，受限要确定一个度量标准，最直接的办法就是将计算法转化为程序，在计算机上运行，通过计算机内部的计时功能获得精确的时间，然后进行比较。但该方法受计算机的硬件、软件等因素的影响，会掩盖算法本身的优劣，所以一般采用事先分析估算的算法，即撇开计算机软硬件等因素，只考虑问题的规模（一般用自然数 n 表示），认为一个特定的算法的时间复杂度，只取决于问题的规模，或者说它是问题的规模的函数。

为了方便比较，通常的做法是，从算法选取一种对于所研究的问题（或算法模型）来说是基本运算的操作，以其重复执行的次数作为评价算法时间复杂度的标准。该基本操作多数情况下是由算法最深层环内的语句表示的，基本操作的执行次数实际上就是相应语句的执行次数。

一般说来：

```
T(n) = O(f(n))
O(1) < O(log 2 n) < O(n) < O(n log 2 n) < O(n^2) < O(n^3) < O(2^n)
```

所以要选择时间复杂度量级低的算法。

至于本题，在这里观看代码可知， $x=x+1$ ，是循环最内侧代码，其时间复杂度最高，所以只求这句代码的复杂度即可。从内到外看，k 循环从 $1=2^0$ 开始每次变成原来的 2 倍，一直到大于 $n-1$ ，所以应该是循环体运行次数是 $\lfloor \log(n) \rfloor$ ，时间复杂度为 $O(\log(n))$ （计算机中 \log 默认底数是 2）；j 循环从 1 开始每次递增 $n/2$ ，一直到 $n-1$ ，第一次递增之后 j 变成 $(n+2)/2$ ，第二次递增 j 则是 $n+1$ 所以应该是循环了 2 次，但是时间复杂度还是 $O(1)$ ，因为常数次数的时间复杂度都是 $O(1)$ 的，i 循环从 1 开始，每次增 1 一直到 $n-1$ ，所以循环体运行 $n-1$ 次，时间复杂度为 $O(n)$ 。最后相乘得到总的时间复杂度就是 $O(n * 1 * \log(n)) = O(n * \log(n))$ ；这里要强调一下：时间复杂度都不带常数项或者常数系数的，所以不存在所谓 $O(2n)$ 这样的时间复杂度。

答案：D

面试例题 3：以下哪种节构，平均来讲获取任意一个指定值最快，为什么？[中国某互联网公司 2009 年 11 月面试题]

- A. 二叉排序树 B. 哈希表 C. 栈 D. 队列

解析：一般来说，哪个需要的额外空间越多，哪个越快。

哈希表和哈希函数是大学数据结构中的课程，实际开发中我们经常用到 Hashtable 这种节构，当遇到键-值对存储，采用 Hashtable 比 ArrayList 查找的性能高。为什么呢？我们在享受高性能的同时，需要付出高额外空间的代价。那么使用 Hashtable 是否就是很好的选择呢？就此疑问，做分析如下：

1. 于键-值查找性能高

数据结构描述线性表和树时，记录在节构中的相对位置是随机的，记录和关键字之间不存在明确的关系，因此在查找记录的时候，需要进行一系列的关键字比较，这种查找方式建立在比较的基础之上，在 Java 中（Array,ArrayList,List）这些集合节构采用了上面的存储方式。比如，现在我们有一个班同学的数据，包括姓名、性别、年龄、学号等。假如数据如下：

姓名	性别	年龄	学号
张三	男	15	1
李四	女	14	2
王五	男	14	3

假如，我们按照姓名来查找，查找函数 `FindBy Name(string name)`：

(1) 查找“张三”：

只需在第一行匹配一次。

(2) 查找“王五”

在第一行匹配，失败；

在第二行匹配，失败；

在第三行匹配，成功。

上面两种情况，分析了最好的情况和最坏的情况，那么平均查找次数应该为 $(1+3)/2=2$ 次，即平均查找次数为（记录总数+1）的 $1/2$ 。尽管有一些优化的算法，可以使查找排序效率增高，但是复杂度会保持在 \log_2^n 的范围之内。

如何更更快的进行查找呢？我们所期望的效果是一下子就定位到要找记录的位置之上，这时候时间复杂度为 1，查找最快。如果我们事先为每条记录编一个序号，然后让它们按号入位，我们又知道按照什么规则对这些记录进行编号的话，如果我们再次查找某个记录的时候，只需要先通过规则计算出该记录的编号，然后根据编号，在记录的线性队列中，就可以轻易地找到记录了。

注意，上述的描述包含了两个概念，一个是用于对学生进行编号的规则，在数据结构中，称之为哈希函数，另外一个是按照规则为学生排列的顺序节构，称之为哈希表。

仍以上面的学生为例，假设学号就是规则，老师手上有一个规则表，在排座位的时候也按照这个规则来排序，查找李四，首先该教师会根据规则判断出，李四的编号为 2，就是在座位中的 2 号位置，直接走过去，就可以找到李四了。

流程如下：

从上面的图中，可以看出哈希表可以描述为两个表，一个表用来装记录的位置编号，另一个表用来装记录；此外存在一套规则，用来表述记录与编号之间的联系。这个规则通常是如何制定的呢？

1) 直接定址法

对于整型的数据 `GetHashCode()` 函数返回的就是整型本身，其实就是基于直接定址的方法，比如有一组 0~100 的数据，用来表示人的年龄。那么，采用直接定址的方法构成的哈希表为：

0	1	2	3	4	5
0岁	1岁	2岁	3岁	4岁	5岁
.....					

这样的定址方式简单方便，适用于原数据能够用数字表述或者原数据具有鲜明顺序关系的情形。

2) 数字分析法：

有这样一组数据，用于表述一些人的出生日期：

年	月	日
75	10	1
75	12	10
75	02	14

分析一下，年和月的第一位数字基本相同，造成冲突的几率非常大，而后面三位差别比较大，所以采用后三位：

3) 平方取中法

取关键字平方后的中间几位作为哈希地址。

4) 折叠法

将关键字分割成位数相同的几部分，最后一部分位数可以不相同，然后取这几部分的叠加和（取出进位）作为哈希地址，比如有这样的数据：

20144545473

可以：

$$\begin{array}{r}
 & 5473 \\
 + & 4454 \\
 + & 201 \\
 = & 10128
 \end{array}$$

取出进位 1，取 0128 为哈希地址。

5) 取余法

取关键字被某个不大于哈希表表长 m 的数 p 除后所得余数为哈希地址。 $H(key)=key \bmod p$ ($p \leq m$)。

6) 随机数法

选择一个随机函数，取关键字的随机函数值为它的哈希地址，即 $H(key)=\text{random}(key)$ ，其中 random 为随机函数。通常关键字长度不等时采用此法。

总之，哈希函数的规则是通过某种转换关系，使关键字适度的分散到指定大小的的顺序节构中。越分散，则以后查找的时间复杂度越小，空间复杂度越高。

2. 使用 hash 付出的代价

hash 是一种典型以空间换时间的算法，比如原来一个长度为 100 的数组，对其查找，只需要遍历且匹配相应记录即可，从空间复杂度上来看，假如数组存储的是 Byte 类型数据，那么该数组占用 100Byte 空间。现在我们采用 hash 算法，我们前面说的 hash 必须有一个规则，约束键与存储位置的关系，那么就需要一个固定长度的 hash 表，此时，仍然是 100Byte 的数组，假设我们需要的 100Byte 用来记录键与位置的关系，那么总的空间为 200Byte，而且用于记录规则的表大小会根据规则，大小可能是不定的。

hash 表最突出的问题在于冲突，就是两个键值经过哈希函数计算出来的索引位置很可能相同。

答案：B

面试例题4: Which of the following operation performs NOT faster on an ordered data over a disordered data? (有序队列数据相对于无序队列数据, 下列哪种操作并不快?) [中国某杀毒软件公司2009年11月面试题]

- A. Find the minimum (找出最小值)
- B. Calculate the average value (估算平均值)
- C. Find the median (找出中间值)
- D. Find the one with maximal occurrence (找出最大出现可能性)

解析: 对于这4种情况分别分析如下:

- 对于寻找最小值: 有序队列数据的时间复杂度是 $O(1)$, 无序队列数据的时间复杂度是 $O(n)$ 。
- 对于估算平均值: 有序队列数据的时间复杂度是 $O(n)$, 无序队列数据的时间复杂度是 $O(n)$ 。
- 对于找出中间值: 有序队列数据的时间复杂度是 $O(1)$, 无序队列数据的时间复杂度是 $O(n)(O(n)的算法类似于快排)$ 。
- 对于找出最大出现可能性: 有序队列数据的时间复杂度是 $O(n)$, 无序队列数据的时间复杂度是 $O(nlgn)$ (使用平衡查找节构而不是哈希表)。

答案: B

面试例题5: 有20个数组, 每个数组里面有500个数, 升序排列, 求出这10000个数字中最大的500个。求复杂度[中国某著名搜索引擎公司B2012年11月面试题]

解析: 20个数组的最小元素全部进堆。每次取最小的一个的时候, 从最小元素对应的数组里取下来一个放进堆里。堆里一直最多有20个数, 充分利用20个数组的有序性。

答案: 复杂度 $500 * \log(20)$

面试例题6: 辗转相除法的时间复杂度是多少? [美国某搜索引擎公司G2013年4月面试题]

答案: 欧几里得算法, 又称辗转相除法, 用于求两个自然数的最大公约数。算法的思想很简单, 基于下面的数论等式 $\gcd(a, b) = \gcd(b, a \bmod b)$, 其时间复杂度为 $O(\log n)$ 。

面试例题7: How does cloud computing provides on-demand functionality? (云计算是如何提供按需模式的功能的?)

答案: 云计算网络、互联网的一种比喻说法, 它提供了以互联网按需模式访问共享的虚拟化IT资源的方式, 所有的资源以资源池的方式存在, 提供配置化的访问方式, 资源类型包括网络、服务器、存储、应用和服务。

面试例题 8：可扩展性和伸缩性的区别是什么？

答案：可扩展性是云计算的特性之一，它通过增加资源容量的方式来满足增长的系统压力，如果系统压力超出一定范围，允许系统架构以按需模式扩展系统容量和系统性能。可扩展性可以通过软件框架来实现：动态加载的插件、顶端有抽象接口的认真设计的类层次结构、有用的回调函数构造以及功能很有逻辑并且可塑性很强的代码结构。

高可伸缩性代表一种弹性，在系统扩展成长过程中，软件能够保证旺盛的生命力，通过很少的改动甚至只是硬件设备的添置，就能实现整个系统处理能力的线性增长，实现高吞吐量和低延迟高性能。

面试例题 9：云计算的三层架构分别是什么？

答案：按照云计算平台提供的服务种类，划分出了云计算平台的三层架构，即：

Infrastructure as a Service(IaaS)：提供CPU，网络，存储等基础硬件的云服务。在IaaS这一层，著名的云计算产品有Amazon的S3(Simple Storage Service)，提供给用户云存储服务。

Platform as a Service(PaaS)：提供类似于操作系统层次的服务与管理，比如Google GAE，你可以把自己写的Java应用（或者是Python）丢在Google的GAE里运行，GAE就像一个“云”操作系统，对你而言，不用关心你的程序在哪台机器上运行。

Software as a Service(SaaS)。代表如亚马逊的Amazon Web services(AWS)，PaaS的代表如Google App Engine(GAE)，以及SaaS的代表如IBM Lotus Live。IaaS就是我们所熟悉的软件即服务。事实上SaaS的概念出现早于云计算，只不过云计算的出现让原来的SaaS找到了自己更加合理的位置。本质上，SaaS的理念是：有别的传统的许可证付费方式（比如购买Windows Office），SaaS强调按需使用付费。SaaS著名的产品很多，比如IBM的LotusLive，Salesforce.com等。

第 14 章

字 符 串

基本上求职者进行笔试时没有不考字符串的。字符串也是一种相对简单的数据结构，容易多次引起面试官反复发问。我曾不止一次在面试时被考官要求当场写出 strcpy 函数的表达方式。事实上，字符串也是一个考验程序员编程规范和编程习惯的重要考点。不要忽视这些细节，因为这些细节会体现你在操作系统、软件工程、边界内存处理等方面的知识掌控能力，也会成为企业是否录用你的参考因素。

14.1 整数字符串转化

面试例题 1：怎样将整数转化成字符串数，并且不用函数 itoa？

解析：整数转化成字符串，可以采用加'0'，再逆序的办法，整数加'0'就会隐性转化成 char 类型的数。

答案：程序代码如下：

```
#include <iostream>
#include <stdio.h>

int main(void)
{
    intnum=12345, j=0, i=0;
    char temp[7], str[7];
    //itoa(number, string, 10);
    while(num)
    {
        temp[i]=num%10+'0';
        i++;
        num=num/10;
    }
}
```

```
temp[i]=0;
printf(" temp=%s\n", temp);
i=i-1;
printf(" temp=%d\n", i);
//刚刚转化的字符串是逆序的，必须把它反转过来
while(i>=0)
{
    str[j]=temp[i];
    j++;
    i--;
}
str[j]=0;
printf(" string=%s\n", str);
return 0;
```

扩展知识

如果可以使用 itoa 函数的话，则十分简单，答案如下：

```
#include <iostream>
#include <stdio.h>

int main(void)
{
    int number = 12345;
    char string[7];

    itoa(number, string, 10);
    printf("integer = %d string = %c\n", number, string[1]);
    return 0;
}
```

面试例题 2：编程实现字符串数转化成整数的办法。[中国某著名 IT 培训企业公司 2005 年面试题]

解析：字符串转化成整数，可以采用减'0'再乘 10 累加的办法，字符串减'0'就会隐性转化成 int 类型的数。

答案：程序代码如下：

```
#include <iostream>
#include <stdio.h>

int main(void)
{
    int num = 12345, j=0, i=0, sum=0;
    char temp[7]={'1','2','3','4','5',
                  '\0'}, str[7];
    while(temp[i])
```

```
{
    sum=sum*10+(temp[i]-'0' );
    i++;
}
printf(" sum=%d\n", sum);
return 0;
```

14.2 字符数组和 strcpy

面试例题 1：Write a function about string copy, the strcpy prototype is "char* strcpy(char* strDest, const char* strSrc);". Here strDest is destination string, strSrc is source string. (已知 strcpy 函数的原型是 char *strcpy(char *strDest, const char *strSrc);，其中 strDest 是目的字符串，strSrc 是源字符串。)

(1) Write the function strcpy, don't call C/C++ string library. (不调用 C++/C 的字符串库函数，请编写函数 strcpy。)

(2) Here strcpy can copy strSrc to strDest, but why we use char* as the return value of strcpy? (strcpy 能把 strSrc 的内容复制到 strDest，为什么还要 char * 类型的返回值？) [中国台湾某著]

名 CPU 生产公司 2005 年面试题]

解析：字符串复制函数问题。

答案：(1) 代码如下：

```
char *strcpy(char *strDest, const char *strSrc);
{
    assert((strDest!=NULL)&&(strSrc!=NULL));
    char *address=strDest;
    while( (*strDest++=*strSrc++)!='\0')
        NULL ;
    return address ;
}
```

(2) 为了实现链式表达式，返回具体值。

例如：

```
int length = strlen( strcpy( strDest, "hello world" ) );
```

面试例题 2：下面的程序会出现何种问题？[美国某著名计算机软件公司面试题]

```
#include <iostream>
#include <stdio.h>
int main(void)
{
    char s []="123456789";
```

```
char d []="123";
strcpy(d,s);
printf("%s,\n%s",d,s);
return 0;
```

解析：以上程序输出结果是 123456789,56789。

没经验的程序员一定会在此大跌眼镜的，源字串竟然被截掉了一部分（截掉的长度恰是目标字串原来的长度。至于原因，应该是当初分配的内存地址是连续内存的问题，原来是 1234\0123456789\0, strcpy 后变成了 123456789\06789\0），所以在分配空间的时候要给源字符串和目标字符串留足够的空间。

把目标字串定义在前，源字串定义在后，虽然可以看到正确的输出结果 123456789, 123456789。但会产生一个运行期错误，原因估计是越过了目标字串的实际空间，访问到了不可预知的地址。

微软在这里是写得非常简单的，代码如下：

```
char * cdecl strcpy(char * dst, const char * src)
{
    char * cp = dst;
    while( *cp++ = *src++ )
        ; /* Copy src over dst */
    return( dst );
}
```

微软为什么这么写？它这样安全漏洞太多了，所以必须预先为目标字串分配足够的空间，并且使用这个函数的时候得小心翼翼才行。

为了提高性能，减去那些罗嗦的安全检查是必要的。况且程序员在使用时应该知道哪些

条件下会发生访问违例，这种做法就是把责任推给了程序员，让他来决定安全与性能的取舍。

答案：123456789,56789。

复制函数的一个完整的标准写法如下：

```
#include<stdio.h>
#include<malloc.h>
#include<assert.h>
#include<string.h>
void stringcpy(char *to,
    const char *form)
{
    assert(to!=NULL && form!=NULL);
    while(*form!='\0')
    {
        *to++=*form++;
    }
    *to='\0';
}
```

```
}
int main(void)
{
    char *f;
    char *t;
    f=(char *)malloc(15);
    t=(char *)malloc(15);
    stringcpy(f,"asdfghjkl");
    stringcpy(t,f);
    printf("%s\n",f);
    printf("%s\n",t);
    return 0;
}
```

扩展知识（数组大小分配）

在使用数组的时候，总有一个问题困扰着我们：数组应该有多大？

在很多的情况下，你不能确定要使用多大的数组。你可能并不知道该班级的学生的人数，那么你就要把数组定义得足够大。这样，你的程序在运行时就申请了固定大小的、你认为足够大的内存空间。即使你知道该班级的学生数，但是如果因为某种特殊原因人数有增加或者减少，你又必须重新修改程序，扩大数组的存储范围。这种分配固定大小的内存分配方法称为静态内存分配。但是这种内存分配的方法存在比较严重的缺陷，特别是处理某些问题时，在大多数情况下会浪费大量的内存空间；在少数情况下，当你定义的数组不够大时，还可能引起下标越界错误，甚至导致严重后果。

那么有没有其他的方法来解决这样的问题呢？有，那就是动态内存分配。

所谓动态内存分配就是指在程序执行的过程中动态地分配或者回收存储空间的内存分配方法。动态内存分配不像数组等静态内存分配方法那样需要预先分配存储空间，而是由系统根据程序的需要即时分配，且分配的大小就是程序要求的大小。从以上动、静态内存分配比较可以知道动态内存分配相对于静态内存分配的特点：

- 不需要预先分配存储空间。
- 分配的空间可以根据程序的需要扩大或缩小。

1. 如何实现动态内存分配及其管理

要实现根据程序的需要动态分配存储空间，就必须用到以下几个函数。

1) malloc 函数

malloc 函数的原型为：

```
void *malloc (unsigned int size)
```

其作用是在内存的动态存储区中分配一个长度为 size 的连续空间。其参数是一个无符号整型数，返回值是一个指向所分配的连续存储域的起始地址的指针。还有一点必须注意的是，若函数未能成功分配存储空间（如内存不足）就会返回一个 NULL 指针，所以在调用该函数时应该检测返回值是否为 NULL 并执行相应的操作。

下例是一个动态分配的程序：

```
main()
{
    int count,*array;
    /*count 是一个计数器，array 是一个整型指针，也可以理解为指向一个整型数组的首地址*/
    if((array(int *) malloc(10*sizeof(int)))==NULL)
    {
        printf("不能成功分配存储空间。");
        exit(1);
    }
    for (count=0;count < 10;count++) /*给数组赋值*/
        array[count]=count;
    for(count=0;count < 10;count++) /*打印数组元素*/
        printf("%2d",array[count]);
}
```

上例中动态分配了 10 个整型存储区域，然后进行赋值并打印。例中 if((array(int *) malloc(10*sizeof(int)))==NULL)语句可以分为以下几步：

- (1) 分配 10 个整型的连续存储空间，并返回一个指向其起始地址的整型指针。
- (2) 把此整型指针地址赋给 array。
- (3) 检测返回值是否为 NULL。

2) free 函数

由于内存区域总是有限的，不能无限制地分配下去，而且一个程序要尽量节省资源，所以当所分配的内存区域不用时，就要释放它，以便其他的变量或者程序使用。这时我们就要用到 free 函数。其函数原型是：

```
void free(void *p)
```

作用是释放指针 p 所指向的内存区域。

其参数 p 必须是先前调用 malloc 函数或 calloc 函数（另一个动态分配存储区域的函数）时返回的指针。给 free 函数传递其他的值很可能造成死机或其他灾难性的后果。

注意：这里重要的是指针的值，而不是用来申请动态内存的指针本身。例如：

```
int *p1,*p2;
p1=malloc(10*sizeof(int));
p2=p1;
.....
free(p2) /*或者 free(p2) */
```

malloc 返回值赋给 p1，又把 p1 的值赋给 p2，所以此时 p1、p2 都可作为 free 函数的参数。

数的参数。malloc 函数对存储区域进行分配。free 函数释放已经不用的内存区域。所以有这两个函数就可以实现对内存区域进行动态分配并进行简单的管理了。

面试例题 3：编写一个函数，作用是把一个 char 组成的字符串循环右移 n 个。比如原来是“abcdefghi”，如果 n=2，移位后应该是“hiabcdefg”。

函数头是这样的：

```
//pStr 是指向以'\0'结尾的字符串的指针  
//steps 是要求移动的n  
  
void LoopMove ( char * pStr, int steps )
```

```
{  
    //请填充  
}
```

解析：这个试题主要考查面试者对标准库函数的熟练程度，在需要的时候引用库函数可以很大程度上简少程序编写的工作量。

最频繁被使用的库函数包括 strcpy、memcpy、memset。

答案：

解答 1：

```
void LoopMove ( char *pStr, int steps )  
{  
    int n = strlen( pStr ) - steps;  
    char tmp[MAX_LEN];  
    strcpy ( tmp, pStr + n );
```

```
strcpy ( tmp + steps, pStr );  
*( tmp + strlen ( pStr ) ) = '\0';  
strcpy( pStr, tmp );  
}
```

解答 2：

```
void LoopMove ( char *pStr, int steps )  
{  
    int n = strlen( pStr ) - steps;  
    char tmp[MAX_LEN];  
    memcpy( tmp, pStr + n, steps );  
    memcpy(pStr + steps, pStr, n );  
    memcpy(pStr, tmp, steps );  
}
```

14.3 数组初始化和数组越界

面试例题 1：下面关于数组的初始化正确的是哪项？[中国著名网络企业 XL 公司面试题]

- A. char str[2]={"a","b"};
- B. char str[2][3]={"a","b"};
- C. char str[2][3]={{'a','b'},{'e','d'},{'e','f'}};
- D. char str[]={ "a","b" };

解析：数组初始化问题。

答案：B

面试例题 2: Find the defects in each of the following programs, and explain why it is incorrect. (找出下面程序的错误，并解释它为什么是错的。) [中国台湾某著名杀毒软件公司 2005 年面试题]

```

void test1() {
    char string[10];
    char* str1="0123456789";
    strcpy(string,str1);
    std::cout<<string<<'\n';
}

void test2() {
    char string[10],str1[10];
    for(int i=0;i<10;i++) {
        str1[i]='a';
    }
}

```

```

strcpy(string,str1);
std::cout<<string<<'\n';

}

void test3(char* str1) {
    char string[10];
    if(strlen(str1)<=10) {
        strcpy(string,str1);
    }
    std::cout<<string<<'\n';
}

```

解析：字符数组和 strcpy 问题。

对于函数 test1，这里 string 数组越界。因为字符串长度为 10，还有一个结束符'\0'，所以总共有 11 个字符长度。string 数组大小为 10，这里越界了。但是虽然越界但并不报错，整个程序无论编译还是运行都可以正常通过。字符数组并不要求最后一个字符为'\0'。是否需要加入'\0'，完全由系统需要决定。

使用 strcpy 函数的时候一定要注意前面目的数组的大小必须大于后面字符串的大小，否则便是访问越界。

对于函数 test2，这里最大的问题还是 str1 没有结束符，因为 strcpy 的第二个参数应该是一个字符串常量。该函数就是利用第二个参数的结束符来判断是否复制完毕，所以在 for 循环后面应加上 str1p[9] = "\0"。

字符数组和字符串的最明显的区别就是字符串会被默认地加上结束符'\0'。

对于函数 test3，这里的问题仍是越界问题。strlen 函数得到字符串除结束符外的长度。如果这里是大于等于 10 的话，就很明显是越界了。

小结：上面的 3 个找错的函数，主要是考查对字符串和字符数组概念的掌握，以及对 strcpy 函数和 strlen 函数的理解。

字符数组并不要求最后一个字符为'\0'。是否需要加入'\0'，完全由系统需要决定。但是字符数组的初始化要求最后一个字符必须为'\0'，所以 test2 虽然能够编译通过，但是会出现运行时错误。类似于 `char c[5]={'C','h','i','n','a'}` 这样的定义是错误的。

答案：

可以编译通过的程序如下所示：

```
#include <iostream>

void test1() {
    char string[10];
    char* str1="0123456789";
}

```

```

strcpy(string,str1);
std::cout<<string<<'\n';

}

void test2() {

```

```

char string[10],str1[10];
for(int i=0;i<9;i++) {
    // 错误1
    str1[i]='a';
}
str1[9]='\0';
strcpy(string,str1);
std::cout<<string<<'\n';
}

void test3(char* str1) {
    char string[10];
    if(strlen(str1)<=10) {

```

```

        strcpy(string,str1);
    }
    std::cout<<string<<'\n';
}

int main()
{
    test1();
    test2();
    char* str="0123456789";
    test3(str);
    return 0;
}

```

面试例题3：本段代码有什么问题，如何修改？[中国台湾著名杀毒软件公司 Q 2007 年 9 月面试题]

```

#include <iostream>
using namespace std;

#define MAX 255

main()
{
    char p[MAX+1];

```

```

    char ch;
/* 将 <= 变成 < */
for (ch=0; ch <=MAX; ch++)
    { p[ch]=ch; cout << ch << " " ; }

cout << ch << " " ;
}

```

解析：这是 char 数值问题。char 值范围为 -128~127，由于 char ch; 的执行，程序会在栈中开辟一个大小为 256 的 char 空间，而第 256 个 char 就是我们声明的 ch。

当执行 ch++，而使 ch = 128 时，这会改变第 256 个空间的值，也就是 ch 的值，改变的结果是 ch 重新变成 -128，那么就持续小于 255，继续进行循环，从而陷入死循环。

如果把 char 修改成 unsigned char 还是会有问题，因为 ch <=MAX; 这个等号的存在，所以在等于 255 的时候一样执行循环，然后 255 加 1，一样溢出，ch 的值变为 0，然后不停地循环。唯一的解决方法是将 “<=” 变成 “<”，循环结束后单独给数组最后一个元素 p[255] 赋值。

答案：程序陷入死循环。

正确代码如下：

```

#include <iostream>
using namespace std;

#define MAX 255

main()
{
    char p[MAX+1];
// 修改ch为unsigned char
    unsigned char ch;

```

```

/* 将 <= 变成 < */
for (ch=0; ch <MAX; ch++)
{
    p[ch]=ch; cout << ch << " " ;
}
/* 在此添加一句：给数组最后一个元素 p[255] 赋值 */
p[ch] = ch;
cout << ch << " " ;
}

```

14.4 数字流和数组声明

面试例题 1: Which is not the standard I/O channel? (下面哪一个不是标准输入 / 输出通道?)

[中国某著名综合软件公司 2005 年面试题]

- A. std::cin
- B. std::cout
- C. std::cerr
- D. stream

解析: I/O stream 问题。头文件 iostream 中含有 cin、cout、cerr 几个对象，对应于标准输入流、标准输出流和标准错误流。

答案: D

面试例题 2: Which definition is correct? (下面哪个数组的声明是正确的?) [中国台湾某著名杀毒软件公司 2005 年 9 月面试题]

- A. int a[];
- B. int n=10,a[n];
- C. int a[10+1]={0};
- D. int a[3]={1,2,3,4};

解析: 数组定义问题。

int a[] 是错误的，不允许建立空数组。

int n=10,a[n]; 这是不可以的，在 C++ 中声明一个数组，a[n]，这里的 n 应为一个常量表达式，而原题目中的 n 是一个整型变量，如果是 const int n=10,a[n]; 就是正确的。

int a[10+1]={0}; 是允许的。

int a[3]={1,2,3,4}; 会造成越界问题，因此不允许。

答案: C。

14.5 字符串其他问题

面试例题 1: 求一个字符串中连续出现次数最多的子串，请给出分析和代码。[中国著名 IT 培训企业 2008 年 3 月面试题]

解析: 这里首先要搞清楚子串的概念，1 个字符当然也算字串，注意看题目，是求连续出现次数最多的子串。如果字符串是 abcbcbcabc，这个连续出现次数最多的子串是 bc，连续出现次数为 3 次。如果类似于 abcccabc，则连续出现次数最多的子串为 c，次数也是 3 次。这个题目可以首先逐个子串扫描来记录每个子串出现的次数。比如：abc 这个字串，对应子串为 a/b/c/ab/bc/abc，各出现过一次，然后再逐渐缩小字符子串来得出正确的结果。

答案：完整代码如下：

```

#include<iostream>
#include<vector>
#include<string>
using namespace std;

pair<int, string> fun(const string &str)
{
    vector<string> subs;
    int maxcount=1, count=1;
    string substr;
    int i, len=str.length();
    for(i=0; i<len; ++i)

subs.push_back(str.substr(i, len-i));
    for(i=0; i<len; ++i)
    {
        for(int j=i+1; j<=(len+i)/2; ++j)
        {
            count=1;
            if(subs[i].substr(0, j-i)      ==
subs[j].substr(0, j-i))
            {
                ++count;
                for(int
k=j+(j-i); k<len; k+=j-i){
                    if(subs[i].substr(0, j-i)
== subs[k].substr(0, j-i))
                        ++count;
                    else
                        break;
                }
                if(count>maxcount)
                {
                    maxcount=count;
                    substr=subs[i].substr(0, j-i);
                } //if
            } //if
        } //for
    } //for
    return make_pair(maxcount, substr);
}

pair<int, string> fun1(const string& str)
{
    int maxcount=1, count=1;
    string substr;
    int i=0, j=0;
    int len=str.length();
}

```

```

    int k=i+1;
    while(i<len){
        j=str.find(str[i], k); //从(k~len-1)
范围内寻找 str[i]
        if(j==string::npos           ||
j>(len+i)/2 )//若找不到, 说明(k~len-1)范围内没有
str[i]
        {
            i++;
            k=i+1;
        }else{ //若找到, 则必有(j>=i+1)
            int s=i;
            int sl=j-i; //连续字串的步长
            while(str.substr(s, sl)==str.substr(j, sl)){//
检测连续字串是否相等
                ++count;
                s=j;
                j=j+sl;
            } //while
            if(count>maxcount)//记录次数最多的
连续相同字串
            {
                maxcount=count;
                substr=str.substr(i, sl);
            }
            k=j+1;
            count=1;
        } //else
    } //while
    return make_pair(maxcount, substr);
}

int main()
{
    string str;
    pair<int, string> rs;
    while(cin>>str)
    {
        rs=fun(str);
        cout<<rs.second<<":"<<rs.first<<endl;
        rs=fun1(str);
        cout<<rs.second<<":"<<rs.first<<endl;
    }
    return 0;
}

```

面试例题 2：编程：输入一行字符串，找出其中出现的相同且长度最长的字符串，输出它及其首字符的位置。例如“yyabcdabjcabc”，输出结果应该为 abc 和 3。[中国著名 IT 培训企业 2008 年 3 月面试题]

解析：可以将字符串 yyabcdabjcabc 分解成：

```
yyabcdabjcabc
yabcdabjcabc
abcdabjcabc
bcdabjcabc
cdabjcabc
```

```
.....  
ceg  
eg  
g
```

对这几个字符串排序，然后比较相邻字符串的前驱就可以了，很容易求出最长的公共前驱。

答案：完整代码如下：

```
#include <iostream>
#include<string>
using namespace std;
int main()
{
    string str,tep;
    cout<<"请输入字符串"<<endl;
    cin>>str;
    for(int i=str.length()-1;i>1;i--)
    {
        for(int j=0;j<str.length();j++)
        {
            if(j+i<=str.length())
            {
                size_t t=0;
                size_t num=0;
```

```
//子串
tep=str.substr(j,i); //从大到小取
t=str.find(tep); //正序查找
num=str.rfind(tep); //逆序查找
if(t!=num) //如果两次查找位置不一致
//说明存在重复子串
{
    cout<<tep<<" "<< t+1<<endl; //输出子串及位置
    return 0;
}
}
}
return 0;
}
```

面试例题 3：Please implement the function strstr() (Find a substring, returns a pointer to the first occurrence of strCharSet in string), DO NOT use any C run-time functions. const char* strstr(const char* string, const char* strCharSet); （请写一个函数来模拟 C++ 中的 strstr() 函数：该函数的返回值是主串中字符子串的位置以后的所有字符。请不要使用任何 C 程序已有的函数来完成。）
[中国台湾某著名杀毒软件公司 2005 年面试题]

解析：string 字符串问题。做一个程序模拟 C++ 中的 strstr() 函数。strstr() 函数是把主串中子串及以后的字符全部返回。比如主串是“12345678”，子串是“234”，那么函数的返回值就是“2345678”。

答案：正确程序如下：

```
#include <iostream>
using namespace std;

const char* strstrl(const char* string,
const char* strCharSet)
{
    for(int i=0;string[i]!='\0';i++)
    {
        int j=0;
        int temp=i;
        if(string[i]==strCharSet[j])
        {

while(string[i]==strCharSet[j++])
{
    if((strCharSet[j]=='\0'))
        return &string[i-j];
}
i=temp;
    }
}
```

```

    }
}
return NULL;
}

int main()
{
    char* string="12345554555123";
    cout<<string<<endl;
    char strCharSet[10]={};
    cin>>strCharSet;
    cout<<strstrl(string,strCharSet)
<<endl;
//char*string2=strstr("123455545
//55123","234");
// cout<<string2<<endl;

    return 0;
}
```

面试例题4：将一句话里的单词进行倒置，标点符号不倒换。比如一句话“i come from tianjin.”倒换后变成“tianjin. from come i”。

解析：解决该问题可以分为两步：第一步全盘置换将该句变成“.nijnait morf emoc i”，第二步进行部分翻转，如果不是空格，则开始翻转单词。

答案：

具体代码如下：

```
#include <iostream>
#include <stdio.h>

int main(void)
{
    int num=-12345,j=0,i=0,flag=0,begin,
end;
    char str[]="icomfromtianjin.",temp;
    j=strlen(str)-1;

    printf(" string = %s\n", str);
    //第一步是进行全盘翻转，将单词变成
    // ".nijnait morf emoc i"
    while(j>i)
    {
        //str[j]=temp[i];
        temp=str[i];
        str[i]=str[j];
        str[j]=temp;
        j--;
        i++;
    }
    printf(" string = %s\n", str);
```

```

    i=0;
    //第二步进行部分翻转，如果不是空格 则开始翻
//转单词
    while(str[i])
    {
        //str[j]=temp[i];
        if(str[i]!=' ')
        {
            begin = i;
            while(str[i]&&str[i]!=' ')
                {i++;}
            i=i-1;
            end=i;
        }
        while(end>begin)
        {
            //str[j]=temp[i];
            temp=str[begin];
            str[begin]=str[end];
            str[end]=temp;
            end--;
            begin++;
        }
    }
}
```

```

    }
    i++;
}
printf(" string = %s\n", str);
return 0;
}

```

面试例题 5： Consider a function which, for a given whole number n , returns the number of ones required when writing out all numbers between 0 and n . For example, $f(1)=1$, $f(13)=6$. Notice that $f(1)=1$. What is the next largest n such that $f(n)=n$? $n \leq 4\ 000\ 000\ 000$.

e.g. $f(13)=6$

because the number of “1” in 1,2,3,4,5,6,7,8,9,10,11,12,13 is 6. (1, 11, 12, 13)

(现在要我们写一个函数，计算 4 000 000 000 以内的最大的那个 $f(n)=n$ 的值，函数 f 的功能是统计所有 0 到 n 之间所有含有数字 1 的数字和。

比如: $f(13)=6$

因为“1”在“1,2,3,4,5,6,7,8,9,10,11,12,13”中的总数是 6 (1, 11, 12, 13) [美国著名搜索引擎公司 G 2008 年 4 月面试题]

解析：字符串数字统计问题。

答案：完整代码如下：

```

#include <stdio.h>
unsigned long f(unsigned long n){
    unsigned long fn = 0, ntemp = n;
    unsigned long step;
    for(step = 1; ntemp > 0; step *= 10, ntemp
/= 10){
        fn += (((ntemp - 1) / 10) + 1) * step;
        if(( ntemp % 10 ) ==1){
            fn -= step - (n % step + 1);
        }
    }
    return fn;
}
unsigned long get_max_fn_equal_n(unsigned
long upper_bound){
    unsigned long n = 1, fn = 0;
    unsigned long max = 1;
    while(n <= upper_bound) {
        fn = f(n);
        if(fn == n){
            max = n;
        }
    }
    return max;
}

```

```

printf("%10lu\t", n++);
}
else if( fn < n )
    n += (n-fn)/10 + 1;
else
    n = fn;
}
return max;
}
int main()
{
    unsigned long upper_bound =
4000000000UL;
    printf("[::test] f(%lu) = %lu.\n", 13,
f(13));
    printf("\n[::max] max({f(n)=n, n<=%lu}) =
%lu.\n", upper_bound, get_max_fn_equal_
n(upper_bound));
    return 0;
}

```

对上面代码稍做如下解释：

1) 当 $f(n)>n$ 的时候，令 $c=f(n)-n>0$ ，设 b 属于 $[0,c]$ ，即 $0 \leq b < c$ 。因为 $f(n)$ 是一个非递减函数，当 $n_2 > n_1$ 时，必有 $f(n_2) \geq f(n_1)$ 。那么有 $f(n+b) \geq f(n)$ 。又因为 $b < c$ 且 $c = f(n) - n$ ，所以

$b < f(n) - n$, 得出 $f(n) > n + b$ 。最后得出 $f(n+b) \geq f(n) > n + b$ 。

也就是说只要 b 属于 $[0, c)$, 当 n 递增 b 的时候, 必定有 $f(n+b) > n+b$ 。因此这些值都可以被剪枝忽略掉。我们取 b 的上确界值 c 来说。则有 $f(n+c) \geq f(n) \geq n+c$ 。这时才可能出现 $f(n+c) = n+c$ 的情况。这里是可能, 而不是一定。

所以当 $fn > n$ 的时候, 选取递增步长 $c = f(n) - n$, 令 $n = n + c = n + fn - n = fn$ 。

2) 当 $fn < n$ 的时候, 题目给的上限是 4 000 000 000, 这是一个 10 位数。可以得出结论, 当 n 增加 1 的时候, $f(n)$ 最多增加 10。这是一种极端情况, 即新增加的那个数是 1 111 111 111, 所以多了 10 个 1, 那么 $f(n)$ 最多增加 10。目前, 选取某个步长 b , 当 $n+b$ 时, 依然有 $f(n+b) < n+b$, 但是依然迅速逼近 $f(N)=N$ 。

假设现在 $f(n1) < n1$, 那么想要达到 $f(n1) = n1$ 的情况, $f(n1)$ 至少得增加 $n1 - f(n1)$ 。

而此时, 在之前推出的结论基础上 (当 n 增加 1 的时候, $f(n)$ 最多增加 10), 可以得出 $n1$ 最少增加了 $(n1 - f(n1))/10 + 1$ 。令 $n1$ 增加后结果记为 $n2 = n1 + (n1 - f(n1))/10 + 1$ 。

因此 $n2 > n1$, 所以 $f(n2) \geq f(n1)$, 而 $f(n2) = f\{ n1 + (n1 - f(n1))/10 + 1 \} < f(n1) + n1 - f(n1) = n1$ 。所以 $n2 > n1 > f(n2)$ 。

因此, 当 $fn < n$ 的时候, 取步长为 $(n - fn)/10 + 1$, 这样的话可以迅速逼近 $f(N)=N$ 。

14.6 字符子串问题

面试例题 1: 转换字符串格式为原来字符串里的字符+该字符连续出现的个数, 例如字符串 1233422222, 转化为 1121324125 (1 出现 1 次, 2 出现 1 次, 3 出现 2 次……)。

怎么实现比较简便? [美国著名搜索引擎公司 G 2007 年面试题]

解析: 可以通过 `sprintf` 语句来实现算法。

`sprintf` 跟 `printf` 在用法上几乎一样, 只是打印的目的地不同而已, 前者打印到字符串中, 后者则直接在命令行上输出。

1) 打印字符串

`sprintf` 最常见的应用之一莫过于把整数打印到字符串中, 所以, `sprintf` 在大多数场合可以替代 `itoa`。如:

```
// 把整数 123 打印成一个字符串保存在 s 中
sprintf(s, "%d", 123); // 产生"123"
```

可以指定宽度, 不足的左边补空格:

```
 sprintf(s, "%8d%8d", 123, 4567); //产生: " 123 4567"
```

当然也可以左对齐:

```
 sprintf(s, "%-8d%8d", 123, 4567); //产生: "123 4567"
```

也可以按照十六进制打印:

```
 sprintf(s, "%8x", 4567); //小写十六进制, 宽度占 8 个位置, 右对齐
 sprintf(s, "%-8X", 4568); //大写十六进制, 宽度占 8 个位置, 左对齐
```

2) 连接字符串

printf 的格式控制串中既然可以插入各种东西, 并最终把它们“连成一串”, 自然也就能够连接字符串, 从而在许多场合可以替代 strcat。sprintf 能够一次连接多个字符串, 可以同时在它们中间插入别的内容, 非常灵活。比如:

```
char* who = "I";
char* whom = "Duantao";
sprintf(s, "%s love %s.", who, whom); //产生: "I love Duantao."
```

答案: 程序源代码如下:

```
# include <iostream>
# include <string>
using namespace std;
int main()
{
    cout << "Enter the numbers " << endl;
    string str;
    char reschar[50];
    reschar[0] = '\0';
    getline(cin, str);
    int len=str.length();
    int count=1;
    int k;
    for(k=0; k <= len-1; k++)
    {
        if(str[k+1]==str[k])
        {
            count++;
        }
        else
        {
            sprintf(reschar+strlen(reschar), "%c%d ", str[k],count);
            count=1;
        }
    }
    if(str[k]==str[k-1])
        count++;
    else
        count=1;
    sprintf(reschar+strlen(reschar), "%c%d ", str[k],count);
    cout << reschar << "gg" << endl;
}
```

```
    cout << endl;
    return 0;
}
```

面试例题2：填空题：移动字符串内容。传入参数 char *w 和 m，规则如下：将 w 中字符串的倒数 m 个字符移到字符串前面，其余依次像右移。例如：ABCDEFGHI,M=3,那么移动之后就是 GHIABCDEF。注意不得修改原代码。[日本著名软件企业 H 公司 2013 年 2 月面试题]

```
void fun(char *w, int m)
{
    int i=0; len=strlen(w);
    if(m>len)   m = len;
    while(/* 请输入代码 */){
        /*请输入代码*/ ;
    }
    w[len-m] = '\0';
}
```

答案：代码如下：

```
while(len-m > 0 || (m==0) != 0) {
    for(i=0,w[len]=w[0],++m;i<len;i++) w[i]=w[i+1];
```

第 15 章

设计模式与软件测试

鲁迅先生曾经说过：“地上本没有路，走的人多了也就成了路。”设计模式如同此理，它是经验的传承，并非体系。它是被前人发现，经过总结形成的一套某一类问题的一般性解决方案，而不是被设计出来的定性规则。它不像算法那样可以照搬照用。

设计模式关注的重点在于通过经验提取的“准则或指导方案”在设计中的应用，因此在不同层面考虑问题的时候就形成了不同问题领域上的模式。模式的目标是，把共通问题中的不变部分和变化部分分离出来。不变的部分，就构成了模式。因此，模式是一个经验提取的“准则”，并且在一次一次的实践中得到验证。在不同的层次有不同的模式，小到语言实现，大到架构。在不同的层面上，模式提供不同层面的指导。

和建筑结构一样，软件中亦有诸多的“内力”。和建筑设计一样，软件设计也应该努力疏解系统中的内力，使系统趋于稳定、有生气。一切软件设计都应该由此出发。任何系统都需要有变化，任何系统都会走向死亡。作为设计者，应该拥抱变化，利用变化，而不是逃避变化。

经典设计模式一共有 23 种，面试时重要的不是你熟记了多少个模式的名称，关键还在于付诸实践的运用。为了有效地设计而去熟悉某种模式所花费的代价是值得的，因为很快你会在设计中发现这种模式真的很好，很多时候它会令你的设计更加简单。

软件测试是指使用人工或者自动手段来运行或测试某个系统的过程，其目的在于检验软件是否满足规定的需求或弄清预期结果与实际结果之间的差别。测试工程师是目前 IT 行业极端短缺的人才，未来 5 年 IT 行业最炙手可热的职位。中国软件业每年新增约 20 万测试岗位就业机会，而企业、学校培养出的测试人才却不足需求量的 1/10，这种测试人才需求与供给间的差距仍在拉大。

随着软件市场的成熟，软件对社会运转的巨大贡献已经得到了广泛认可，但是，人们对软件作用期望值也越来越高，更多人将关注点转移到软件的质量和功能可靠性上，而软件产

业在产品性能测试领域存在着严重不足，软件测试水平的高低可以说是决定了软件产业的前途命运。

15.1 设计模式

面试例题 1: Which came first, chicken or the egg? (请用设计模式观点描述先有鸡还是先有蛋?)

[德国某著名软件咨询企业 2005 年 10 月面试题]

解析:

请先看下面的程序：

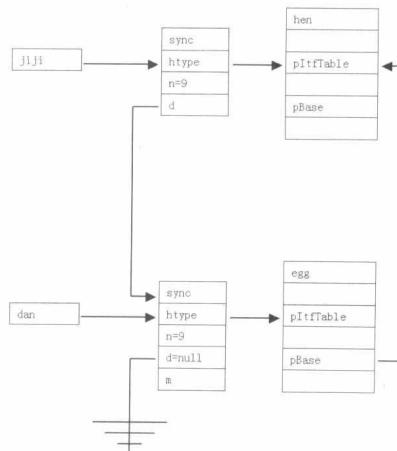
```
using System;
class Client
{
    public static void Main ()
    {
        hen jiji = new hen();
        egg dan = new egg();
        jiji.d = dan;
        Console.WriteLine(jiji.d.m);
    }
}
```

```
class hen
{
    public int n = 9;
    public egg d;
}

class egg : hen
{
    public int m = 10;
}
```

egg 继承自 hen，可以说没有 hen 就没有 egg，可 hen 里面有一个成员是 egg 类型。到底是先有鸡还是先有蛋？这个程序可以正常编译执行并打印结果 10。

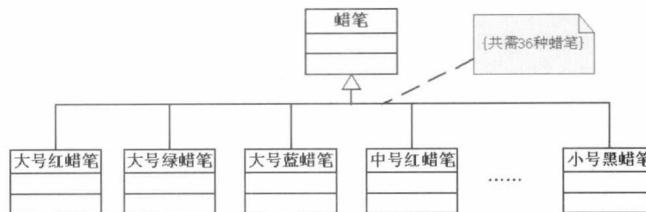
答案：“先有鸡还是先有蛋”问题只是对面向对象本质的一个理解，将人类的自然语言放在此处来理解并不合适。由下图可知，根本不存在鸡与蛋的问题，而是型与值的问题，以及指针引用的问题，因为鸡和蛋两个对象间是“引用”关系而不是“包含”关系。



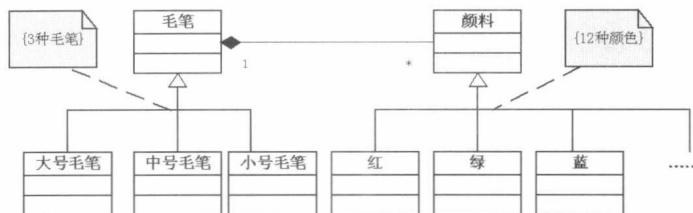
面试例题 2：请用设计模式的思想描述蜡笔和毛笔有什么不同，并用 C# 程序实现。

解析：大家小时候都有用蜡笔画画的经历吧。红红绿绿的蜡笔一大盒，根据想象描绘出各式的图样。而毛笔下的国画更是工笔写意各展风采。而今天我们的故事从蜡笔与毛笔说起。

设想要绘制一幅图画，蓝天、白云、绿树、小鸟。如果画面尺寸很大，那么用蜡笔绘制就会遇到点儿麻烦。毕竟细细的蜡笔要涂出一片蓝天是有些麻烦的。如果有可能，最好有套大号蜡笔，粗粗的蜡笔很快能涂抹完成。至于色彩嘛，最好每种颜色来支粗的，除了蓝天还有绿地呢。这样，如果一套 12 种颜色的蜡笔，我们需要两套 24 支，同种颜色的一粗一细。要是再有一套中号蜡笔就更好了，于是，不多不少总共 36 支蜡笔（见下图）。



再看看毛笔这一边，居然如此简陋：一套水彩 12 色，外加大、中、小 3 支毛笔（见下图）。你可别小瞧这“简陋”的组合，画蓝天用大毛笔，画小鸟用小毛笔，各有专长。



这就是 Bridge 模式。为了一幅画，我们需要准备 36 支型号不同的蜡笔，而改用毛笔的话 3 支就够了，当然还要搭配上 12 种颜料。通过 Bridge 模式，我们把乘法运算 $3 \times 12 = 36$ 改为了加法运算 $3 + 12 = 15$ ，这一改进可不小。那么这里蜡笔和毛笔到底有什么区别呢？

实际上，蜡笔和毛笔的一个关键区别就在于笔和颜色是否能够分离。桥梁模式的用意是“将抽象化（Abstraction）与实现化（Implementation）脱耦，使得二者可以独立地变化”。关键就在于能否脱耦。蜡笔的颜色和蜡笔本身是分不开的，所以必须使用 36 支色彩、大小各异的蜡笔来绘制图画。而毛笔与颜料能够很好地脱耦，各自独立变化，简化了操作。在这里，抽象层面的概念是“毛笔用颜料作画”，而在实现时，毛笔有大、中、小 3 号，颜料有红、绿、蓝等 12 种，于是便可出现 3×12 种组合。每个参与者（毛笔与颜料）都可以在自己的

自由度内随意转换。

蜡笔由于无法将笔与颜色分离，造成笔与颜色两个自由度无法单独变化，使得只有创建36种对象才能完成任务。Bridge模式将继承关系转换为组合关系，从而降低了系统间的耦合，减少了代码编写量。

答案：

代码如下：

```
// 桥接模式类型举例
using System;

// "Abstraction"类
class Abstraction
{
    // 字段
    protected Implementor implementor;

    // 属性
    public Implementor Implementor
    {
        set{ implementor = value; }
    }

    // 函数
    virtual public void Operation()
    {
        implementor.Operation();
    }
}

// "Implementor"类
abstract class Implementor
{
    // 函数
    abstract public void Operation();
}

// "RefinedAbstraction"类
class RefinedAbstraction : Abstraction
{
    // 函数
    override public void Operation()
    {
        implementor.Operation();
    }
}

// "ConcreteImplementorA"类
class ConcreteImplementorA:Implementor
{
```

```
// 函数
override public void Operation()
{
    Console.WriteLine("ConcreteImplementorA
Operation");
}

// "ConcreteImplementorB"
class ConcreteImplementorB:Implementor
{
    // 函数
    override public void Operation()
    {
        Console.WriteLine("ConcreteImplementorB
Operation");
    }
}

/**/**/<summary>
/// Client test
/// </summary>
public class Client
{
    public static void Main(string[]
args )
    {
        Abstraction abstraction=new
RefinedAbstraction();

        // 设置 implementation 委托并且调用
abstraction.Implementor=newConcreteImple-
mentorA();
        abstraction.Operation();

        // 修改 implementation 委托并且调用
    }
}
```

```
abstraction.Implementor=newConcreteImplementorB(); } }
```

面试例题 3：以下代码实现了设计模式中的哪种模式？[美国某著名搜索引擎公司 2005 年面试题]

```
public sealed class SampleSingleton1
{
    private int m_Counter = 0;
    private SampleSingleton1()
    {
        Console.WriteLine("初始化 SampleSingleton1。");
    }
    public static readonly SampleSingleton1 Singleton = new
        SampleSingleton1();
    public void Counter()
    {
        m_Counter++;
    }
}
```

- A. 原型
 - B. 抽象工厂
 - C. 单键
 - D. 生成器

解析：这是一个典型的单键模式。

答案：C

面试例题 4：Consider a context-sensitive help feature for graphical user interface. The user can obtain help information on any part of the interface just by clicking on it. The help that's provided depends on the part of interface that's selected and its context; for example, a button widget in a dialog box might have different help information than a similar button in the main window. If no specific help information exists for that part of the interface, then the help system should display a more general help message about the immediate context—the dialog box as a whole, for example.

(为图形用户界面设计一种特性——能领会语境并提供帮助，使得用户仅仅通过在界面的任何部分上单击鼠标就可以获得相应的帮助信息。它所提供的帮助主要依赖于所选择的界面及相应的语境。

举个例子，在具备这样特性的一个对话框中的一个按钮部件上单击鼠标可能就会有一个与主窗口中类似的按钮但有不同的帮助信息。如果没有该界面的这部分内容的具体帮助信息，那么帮助系统就会根据即时的具体语境提供较为通用的帮助信息，这就是对于上述问题的一个具体的例子。)

为了设计上面的东西该选用（ ）。[中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

- A. 观察者模式
- B. 桥接模式
- C. 供应链模式
- D. 原型模式

解析：本题应该选择观察者模式。为了形象描述观察者模式，举一个给教工发大米的例子。

每年到年底的时候，工会都要给每位教工发大米、油什么的。正好我家的米快吃完了，所以就盼着发大米。可是等了好多天也没见什么动静，扳着手指头数数离春节还有多半个月呢，看我心急的。不过，盼大米的老师恐怕不只是我一个，都等着工会有什么动静就去领大米呢。

教工等着发大米是一个典型的观察者模式。当工会大米来了，教工就会做出响应。不过这观察也有两种说头法：一种是拉（Pull）模式，要求教工时不时到工会绕一圈，看看大米来了没有，恐怕没有人认为这是一种好办法。当然，还有另外一种模式，就是推（Push）模式，大米到了工会，工会会给每家每户把大米送过去。第二种方法好不好呢？好？呵呵，谁说好了，谁说好我让谁到工会上班去。

其实，我们还有一种方法，就是大米到了工会后，工会不把大米给每人送去，而是给每人发个轻量级的“消息”，教工得到消息后，再把大米拉回各家。这要求每位教工有一个工会的引用，在得到消息后到指定的地点领取大米。这样工会不用给每家教工送大米，而教工也不用每天到工会门口巴望着等大米了。

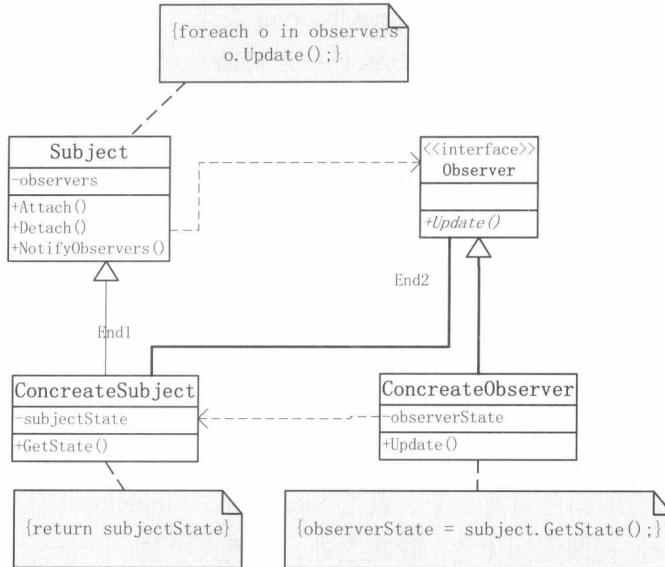
而本题鼠标帮助的例子也与教工拉大米相似。观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题帮助。这个帮助对象在状态上发生变化时，会通知所有观察者对象，使它们能够自动更新自己。

一个软件系统常常要求在某一个对象的状态发生变化的时候，某些其他的对象做出相应的改变。做到这一点的设计方案有很多，但是为了使系统能够易于复用，应该选择低耦合度的设计方案。减少对象之间的耦合有利于系统的复用，但是同时设计师需要使这些低耦合度的对象能够维持行动的协调一致，保证高度的协作（collaboration）。观察者模式是满足这一要求的各种设计方案中最重要的一种。

答案：A

扩展知识（观察者模式的结构）

观察者模式的类如下图所示。



可以看出，在这个观察者模式的实现里有下面这些角色。

- 抽象主题 (Subject) 角色：主题角色把所有对观察者对象的引用保存在一个聚集里，每个主题都可以有任何数量的观察者。抽象主题提供一个接口，可以增加和删除观察者对象。抽象主题角色又叫作抽象被观察者 (Observable) 角色，一般用一个抽象类或者一个接口实现。
- 抽象观察者 (Observer) 角色：为所有的具体观察者定义一个接口，在得到主题的通知时更新自己。这个接口叫作更新接口。抽象观察者角色一般用一个抽象类或者一个接口实现。在这个示意性的实现中，更新接口只包含一个方法（即 Update() 方法），这个方法叫作更新方法。
- 具体主题 (ConcreteSubject) 角色：将有关状态存入具体观察者对象；在具体主题的内部状态改变时，给所有登记过的观察者发出通知。具体主题角色又叫作具体被观察者角色 (ConcreteObservable)。具体主题角色通常用一个具体子类实现。
- 具体观察者 (ConcreteObserver) 角色：存储与主题的状态自恰的状态。具体观察者角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调。如果需要，具体观察者角色可以保存一个指向具体主题对象的引用。具体观察者角色通常用一个具体子类实现。

从具体主题角色指向抽象观察者角色的合成关系，代表具体主题对象可以有任意多个对抽象观察者对象的引用。之所以使用抽象观察者而不使用具体观察者，意味着

主题对象不需要知道引用了哪些 ConcreteObserver 类型，而只需知道抽象 Observer 类型。这就使得具体主题对象可以动态地维护一系列的对观察者对象的引用，并在需要的时候调用每一个观察者共有的 Update()方法。这种做法叫作针对抽象编程。

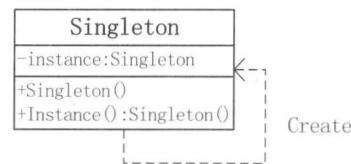
面试例题 5：Implement the simplest singleton pattern(initialize if necessary). (写一个单例模式，如果有必要的话就初始化。) [德国某著名软件咨询企业 2005 年 10 月面试题]

解析：

单例模式的特点有 3 个：

- 单例类只能有一个实例。
- 单例类必须自己创建自己的唯一实例。
- 单例类必须给所有其他对象提供这一实例。

Singleton 模式包含的角色只有一个，就是 Singleton。Singleton 拥有一个私有构造函数，确保用户无法通过 new 直接实例化它。除此之外，该模式中包含一个静态私有成员变量 instance 与静态公有方法 Instance()。Instance 方法负责检验并实例化自己，然后存储在静态成员变量中，以确保只有一个实例被创建。单例模式的结构如右图所示。



答案：

完整代码如下：

```

// 单件模式类型举例
using System;

// "Singleton"
class Singleton
{
    // 字段
    private static Singleton instance;

    // 构造
    protected Singleton() {}

    // 函数
    public static Singleton Instance()
    {
        // 设定初值进行初始化
        if( instance == null )
            instance=new Singleton();
        return instance;
    }
}
  
```

```

    }

    /**
     * <summary>
     * /// Client test
     * /// </summary>
     */
    public class Client
    {
        public static void Main()
        {
            // 构造函数被保护——不能使用 new 字符
            Singletons1=Singleton.Instance();

            Singletons2=Singleton.Instance();

            if( s1 == s2 )
                Console.WriteLine( "The
same instance" );
        }
    }
}
  
```

15.2 软件测试基础

面试例题 1: Why is test automation important? (自动化测试为何重要?) [美国数据库公司 S2009 年 11 月面试题]

解析: 自动化测试从根本上提高 QA 们的职业素质, 让 QA 们从彻底上摆脱重复繁重的测试工作, 而更着重在 QA 的流程上已经完成项目质量的重复保证上。此外某些人对 QA 有些偏见: 对 QA 的普遍认识就是只是测试而已。因为他们能看到 QA 最直接的劳动就是在反反复复勤勤恳恳的测试。

答案: 自动化测试可以让测试人员从枯燥无味的手工重复性测试中解放出来, 并且提高工作效率, 通过自动化测试结果来分析功能和性能上的缺陷。

面试例题 2: Describe criterions that testing is completed. (描述一个测试结束的准则。) [美国数据库公司 S2009 年 11 月面试题]

答案: 一个测试结束的标准可以查看已提交的 bug 是否已经全部解决并已验证关闭, 一般来说, bug 验证率在 95%以上, 并且没有大的影响功能的 bug 处于未解决状态, 就可以测试通过。

面试例题 3: What kinds of content should be included in a Test Plan?(For example, available human resource) (在一个测试计划中能包含哪些内容, 如可用的人力资源?) [美国数据库公司 S2009 年 11 月面试题]

答案: 在一个测试计划中可以包含需要测试的产品的特点和主要功能模块, 列出需要测试的功能点, 并标明侧重点; 测试的策略和记录 (测试工具的确认, 测试用例等文档模板, 测试方法的确定); 测试资源配置 (确定测试每一阶段的任务和所需资源)。

面试例题 4: Please describe differences between functional testing and usability testing.(请描述功能测试和可用性测试之间的区别。) [美国数据库公司 S2009 年 11 月面试题]

解析: 本题涉及几个测试的重要概念:

Functional testing (功能测试), 也称为 behavioral testing (行为测试), 根据产品特征、操作描述和用户方案, 测试一个产品的特性和可操作行为以确定它们满足设计需求。本地化软件的功能测试, 用于验证应用程序或网站对目标用户能正确工作。使用适当的平台、浏览器和测试脚本, 以保证目标用户的体验将足够好, 就像应用程序是专门为该市场开发的一样。

功能测试也叫黑盒子测试或数据驱动测试, 只需考虑各个功能, 不需要考虑整个软件的

内部结构及代码。一般从软件产品的界面、架构出发，按照需求编写出来的测试用例，输入数据在预期结果和实际结果之间进行评测，进而提出使产品更加符合用户使用的要求。

可用性测试是用户在和系统（网站、软件应用程序、移动技术或任何用户操作的设备）交互时对用户体验质量的度量。可用性（Usability）是交互式IT产品/系统的重要质量指标，指的是产品对用户来说有效、易学、高效、好记、少错和令人满意的程度，即用户能否用产品完成他的任务，效率如何，主观感受怎样？实际上是从用户角度所看到的产品质量，是产品竞争力的核心。

答案：功能测试主要是黑盒测试，由测试人员进行，主要验证产品是否符合需求设计的要求；可用性测试主要是由用户（或者测试人员模拟用户行为）来进行的测试，主要是对产品的易用性进行测试，包括有效性（effectiveness）、效率（efficiency）和用户主观满意度（satisfaction）。其中有效性指用户完成特定任务和达到特定目标时所具有的正确和完整程度；效率指用户完成任务的正确和完整程度与所使用资源（如时间）之间的比率；满意度指用户在使用产品过程中所感受到的主观满意和接受程度。

15.3 黑盒测试

面试例题 1: A student needs to score at least 60 points to pass. If they score at least 80 points they will achieve a Merit and if they score 100 points they will achieve the Maximum. Which of the following would be the most likely set of values identified by the boundary Value Testing? [美国著名软件公司 M2009 年 12 月笔试题]

（一个学生需要 60 分才能及格；80 分就可以得优；100 分就是满分了。下面的 4 个选项中哪一个边界测试是最好的，为什么？）

- A. -1,0,59,60,79,80,99,100
- B. 0,59,79,100
- C. 0,1,59,69,70,80,100
- D. 60,80,100

解析：边界值测试，就是找到边界，然后在边界及其边界附近（这里应该包括边界两侧）选点。因此边界 0（隐含需求边界），60，80，100 要测试，边界另一侧的-1，59，79，99 也要测试。对于选项 B、D，只覆盖了边界的一侧，而选项 C 中的 69 和 70 跟边界无关，所以 A 相对最好。

答案：A

扩展知识：

除边界值测试外，面试前最好了解这几个相关概念。

- 健壮性测试：健壮性测试是边界值分析的一种简单扩展。除了变量的 5 个边界值分析之外，还要分析变量值比最高值高出一点和比最低值低一点的情况下会出现什么反应。
- 最坏情况测试：边界值分析时是在单缺陷的假设下进行的。如果不做此假设，那么就会出现同时有多个变量取边界值的情况。最坏情况测试的测试用例的获取，是对每个变量，先进行包含 5 个边界值元素集合的测试，然后对这些集合进行笛卡儿积计算，以生成测试用例。
- 特殊值测试：这种测试不需要使用任何测试方针，只使用最佳工程判断。因此，该方法与测试人员的能力密切相关。
- 随机测试：这种方法不是永远选取有界变量的最小值、略高于最小值、正常值、略低于最大值、最大值，而是使用随机数生成器生成测试用例值。这种测试用例的获取需要用程序来得出，而且还涉及测试覆盖率的问题。

面试例题 2：Function club is used to simulate guest in a club. With 0 guests initially and 50 as max occupancy, when guests beyond limitation, they need to wait outside; when some guests leave the waiting list will decrease. The function will print out number of guests in the club and waiting outside. The function declaration as follows:

```
void club(int x);
```

positive x stands for guests arrived, negative x stands for guests left from within the club. (club 函数用来模拟一个俱乐部的顾客。初始化情况下是 0 个顾客，俱乐部最大规模只能有 50 个顾客，当用户超过了最大规模，他们必须等在外面。当一些顾客离开了等待队列将减少。这个 club 函数将打印在俱乐部里面的顾客人数，和外面的等待人数。函数声明如下：

```
void club(int x);
```

正数 x 代表客人来了，负数 x 代表客人离开了俱乐部。)

For example, club (40) prints 40,0; and then club (20) prints 50,10; and then club (-5) prints 50,5; and then club (-30) prints 25,0; and then club (-30) prints N/A; since it is impossible input. (举例而言，club (40) 打印 40, 0；接着 club (20) 打印 50, 10；接着 club (-5) 打印 50, 5；接着 club (-30) 打印 25, 0；接着 club (-30) 打印 N/A；因为这是不可能实现的。)

To make sure this function works as defined, we have following set of data to pass into the function and check the result are correct. (为了确保函数工作正常，我们使用下列数据来测试函数是否正常，你认为该选哪个选项？) [美国著名软件公司 M2009年12月笔试试题]

a 60
b 20 50 -10
c 40 -30
d 60 -5 -10 -10 10
e 10 -20
f 30 10 10 10 -60
g 10 10 10

h 10 -10 10
A a d e g
B c d f g
C a c d h
D b d g h
E c d e f

解析：本题实际上是考边界条件的测试情况。看有没有覆盖所有的边界条件。设 A 为已在俱乐部的成员，B 为排队的人。

	A	B
Case1	≤ 0	0
Case2	< 50 (Up/Down)	0
Case3	50	> 0 (Up/Down)

对于 a-h 各种情况：

a 60	适用 C3
b 20 50 -10	适用 C2\C3
c 40 -30	适用 C2
d 60 -5 -10 -10 10	适用 C3\C2
e 10 -20	适用 C1
f 30 10 10 10 -60	适用 C1\C2\C3
g 10 10 10	适用 C2
h 10 -10 10	适用 C2

看看条件，肯定要包含 e，因为只有这个 case 能测 N/A 的情况，排除了 B C D 三项；再看 A 和 E，差别在 a 和 c、g 和 f 的选取上，很显然，d 包含 a，f 包含 g，所以排除 A，最终确定 E。

答案：E

扩展知识：本题的测试代码 (C++) 如下：

```
#include <iostream>
using namespace std;
#define MAX_IN_CUSTOM (50)
void club(int x)
{
    // 这里必须使用静态变量
    // 初始情况下，内部客人为 0 个
    static int in_custom = 0;
    // 初始情况下，外部客人为 0 个
    static int out_custom = 0;
    if (x > 0) // 来人的情况
    {
        // 需要将人留在外面
        if ((x+in_custom) >
MAX_IN_CUSTOM )
```

```
{
    // 多于的人留在外面
    out_custom += x+in_custom -
MAX_IN_CUSTOM;
    in_custom = MAX_IN_CUSTOM;
}
else
{
    out_custom = 0; in_custom += x;
}
else if(x < 0) // 走人
{
    x = -x; // 转正
    // 如果走的人比内部与外部之和的人还
```

```

//要多，那么就出现错误了！
// 在下面就表示为 in_custom < 0
// 外面人数更多（超过要走掉的人数）
if ( out_custom > x )
{
    out_custom -= x;
}
else
{
    in_custom-=x-out_custom;
out_custom = 0;
}
} // 打印输出结果
if ( in_custom < 0 )
{
    std::cout << "N/A" << std::endl;
}

else
{
    std::cout << in_custom << "," <<
out_custom << endl;
}

int main()
{
    club(40);
    club(20);
    club(-5);
    club(-30);
    club(-30);
    return 0;
}

```

面试例题 3：int FindMiddle(int a ,int b,int c)和 int CalMiddle(int a ,int b,int c)分别为两个 C 函数，他们都号称能返回三个输入 int 中中间大小的那个 int。你无法看到他们的源代码，那么该如何判断哪个函数的质量好？[中国台湾著名杀毒软件公司 Q2009 年 5 月笔试题]

答案：从编程习惯上看，笔者认为 int FindMiddle(int a ,int b,int c)比较好，因为名字比较明确，就是找中间那一个数，让人一看就明白。

这道题是考软件测试的分析能力，比如一些特殊情况要特殊处理，例如：先测试 0 0 0 看他们的测试结果，再测 0 0 1，再随便输入一些不是数字的数，测一下他们的排错功能，如果他们的结果一样，那就该测他们的算法效率。比如可以计算 10000 个数测试用时：

```

System.out.println(new Date().toString());
for(int i=0; i < 10000; i++){
    for(int j=0; j < 10000; j++) {
        for(int k=0; k < 10000; k++) {
            FindMiddle(i, j, k);
        }
    }
}
System.out.println(newDate().toString())
System.out.println(newDate().toString());
for(int i=0; i < 10000; i++){
    for(int j=0; j < 10000; j++) {
        for(int k=0; k < 10000; k++) {
            CalMiddle(i, j, k);
        }
    }
}
System.out.println(newDate().toString())

```

面试例题 4：Write a function bool symmetry - judge(int m) to judge whether the input parameter m is a symmetry number (eg 3, 66, 121, 3883, 45254)Please describe your algorithm and write code. (写一个函数测试输入数是否为回文数，给出算法和代码) [中国台湾著名杀毒软件公司 Q2009 年 5

月笔试题]

解析：回文数问题 2006 年，2009 年都曾经考过。本题算法是建立数组，按位存储。比较首位和末位是否相同。如不同，则不是回文数。相同则继续比较，首位递增，末位递减直到首位不再小于末位。

答案：完整代码如下：

```
#include <iostream>
using namespace std;

int main()
{
    int j=10,k=12321,p,a[10],ss,i=0,begin,end;
    cout << "please input" << endl;
    cin >> k;
    p=k;
    while(p)
    {
        ss=p%10;
        a[i]=ss;
        p=p/10;
        i++;
    }
    begin = 0;
}
```

```
end = i-1;

while(begin < end)
{
    if(a[begin]!=a[end])
        break;
    else
        { begin++; end--;}
}
if(begin < end)
{cout << "jiade" << endl;
}
else
{cout << "zhende " << endl;
}

cout << "i " << i << endl;
cout << k;

return 0;
}
```

面试例题 5：Please write a test plan and test case the elevator functionalities (请写一个电梯功能的测试用例和测试方案。) [中国台湾著名杀毒软件公司 Q2009 年 5 月笔试题]

解析：电梯调度算法的基本原则就是如果在电梯运行方向上有人要使用电梯则继续往那个方向运动，如果电梯中的人还没有到达目的地则继续向原方向运动。此处将电梯一次从下到上视为一次运行（注意不一定是从底层到顶层），同理，电梯一次从上到下也视为一次运行（注意不一定是从顶层到底层）。

当电梯向上运行时：

- 1 位于当前层以下的向上请求都被忽略留到下次向上运行时处理
- 2 位于当前层以上的向上请求都被记录留到此次运行处理
- 3 无论何层的向下请求都被忽略留到下次向下运行时处理

当电梯向下运行时：

- 1 位于当前层以上的向下请求都被忽略留到下次向下运行时处理
- 2 位于当前层以下的向下请求都被记录留到此次运行处理
- 3 无论何层的向上请求都被忽略留到下次向上运行时处理

答案：如果只考虑单部电梯的实现情况是很简单的。电梯移动引发状态转换，电梯的主要状态有：

- 1 UpRun，电梯上行中

- 2 UpStopClose, 电梯上行楼层暂停未开门
- 3 UpStopOpen, 电梯上行楼层暂停已开门
- 4 DownRun, 电梯下行中
- 5 DownStopClose, 电梯下行楼层暂停未开门
- 6 DownStopOpen, 电梯下行楼层暂停已开门
- 7 FullStopClose, 电梯楼层停止未开门
- 8 FullStopOpen, 电梯楼层停止未开门

电梯类，根据请求转换自身状态，请求分为：

- 1 梯内同向请求
- 2 前方楼层同向请求
- 3 前方楼层逆向请求
- 4 梯内逆向请求
- 5 后方楼层逆向请求
- 6 后方楼层同向请求
- 7 梯内关门请求
- 8 梯内开门请求
- 9 停止时同层同向请求
- 10 停止时同层逆向请求

多部电梯情况相对复杂，每部电梯是一个电梯类的实例，在电梯运行到楼层处时检查请求并进行状态转换，但这样的效果：

- 1 电梯只考虑局部状态没有全局观（只看一步），像苍蝇
- 2 时间一长电梯扎堆（抢任务），像蜜蜂
- 3 电梯经常空跑（任务被其他电梯完成），像蚂蚁

对多部电梯情况进行适当优化：

- 1 增加时间变量
- 2 增加相应配合（只派一部电梯）
- 3 电梯空闲时到重要地点（底层或顶层）

效果：

- 1 电梯行为过分依赖于梯内请求
- 2 电梯不适应突发性集中请求（层层停，没效果）
- 3 电梯上行的行为与下行不同（底层到各层，各层到底层，停层不是聚集点）
- 4 公平服务（先来先向服务）
- 5 反向快速响应（提前转向）
- 6 最长等待者的最小等待（后来者的跳跃型响应）
- 7 最少移动（资源最小调度）

更高级的算法是：

- 1 梯内人优先（判断梯内直达请求）
- 2 梯内人优先（减少无效开门，考虑超载直达）
- 3 不响应下行聚集型请求（减少无效开门，考虑超载直达）

甚至测试可以考虑电梯环境：

- 1 公寓型：只有底层（停车场）到各层或各层到底层，爆发型请求不多
- 2 写字楼型：有若干次爆发性请求（上下班，午休情况）需要测试，有邻层转移要求（同一公司租用）
- 3 酒店型：某些层（大堂，餐厅）有阶段持续请求
- 4 商场型：一般来说均匀调度