第四章 串

一、选择题

1.B	2.E	3.C	4.A	5.C	6.A	7.1D	7.2F	8.B 注	9.D	10.B	

注：子串的定义是：串中任意个连续的字符组成的子序列，并规定空串是任意串的子串，任意串是其自身的子串。若字符串长度为 n（n>0），长为 n 的子串有 1 个，长为 n-1 的子串有 2 个，长为 n-2 的子串有 3 个，……，长为 1 的子串有 n 个。由于空串是任何串的子串，所以本题的答案为：8*（8+1）/2+1=37。故选 B。但某些教科书上认为“空串是任意串的子串”无意义，所以认为选 C。为避免考试中的二意性，编者认为第 9 题出得好。

二、判断题

1. √	2. √	3. √									

三．填空题

1．(1) 由空格字符（ASCII 值 32）所组成的字符串　(2)空格个数　2．字符

3．任意个连续的字符组成的子序列　4．5　5.0(m+n)

6．01122312　7．01010421　8．(1)模式匹配　(2)模式串

9．(1)其数据元素都是字符(2)顺序存储(3)和链式存储(4)串的长度相等且两串中对应位置的字符也相等

10．两串的长度相等且两串中对应位置的字符也相等。

11.’xyxyxywwy’　12．*s++=*t++ 或（*s++=*t++）!= ‘\0’

13.（1）**char** s[ ]　(2) j++　(3) i >= j

14．[题目分析]本题算法采用顺序存储结构求串 s 和串 t 的最大公共子串。串 s 用 i 指针（1<=i<=s.len)。t 串用 j 指针（1<=j<=t.len)。算法思想是对每个 i（1<=i<=s.len，即程序中第一个 **WHILE** 循环），来求从 i 开始的连续字符串与从 j（1<=j<=t.len，即程序中第二个WHILE 循环）开始的连续字符串的最大匹配。程序中第三个（即最内层）的 **WHILE** 循环，是当 s 中某字符（s [i]）与 t 中某字符（t [j]）相等时，求出局部公共子串。若该子串长度大于已求出的最长公共子串（初始为 0 ），则最长公共子串的长度要修改。

程序（a)：(1)（i+k<=s.len）AND(j+k<=t.len) AND(s[i+k]=t[j+k])

//如果在 s 和 t 的长度内，对应字符相等，则指针 k 后移（加 1)。

(2) con:=false　//s 和 t 对应字符不等时置标记退出

(3) j:=j+k　//在 t 串中，从第 j+k 字符再与 s[i]比较

(4) j:=j+1　//t 串取下一字符

(5) i：=i+1　//s 串指针 i 后移（加 1)。

程序（b)：(1) i+k<=s.len && j+k<=t.len && s[i+k]==t[j+k]　//所有注释同上（a)

(2) con=0　(3) j+=k　(4) j++　(5) i++

15.（1）0　（2）next[k]

16.(1)i：=i+1　(2)j:=j+1　(3)i:=i-j+2　(4)j:=1;　(5)i-mt（或 i:=i-j+1）　(6)0

17.程序中递归调用

（1）ch1<>midch　//当读入不是分隔符&和输入结束符$时，继续读入字符

（2）ch1=ch2　//读入分隔符&后，判 ch1 是否等于 ch2，得出真假结论。

（3）answer：=true

（4）answer：=false

（5）read（ch）

（6）ch=endch

18.（1）initstack（s）　//栈 s 初始化为空栈。

(2) setnull (exp)　　//串 exp 初始化为空串。
(3) ch in opset　　//判取出字符是否是操作符。
(4) push (s,ch)　　//如 ch 是运算符，则入运算符栈 s。
(5) sempty (s)　　//判栈 s 是否为空。
(6) succ := false　　//若读出 ch 是操作数且栈为空，则按出错处理。
(7) exp　　(8)ch　//若 ch 是操作数且栈非空，则形成部分中缀表达式。
(9) exp　　(10) gettop(s) //取栈顶操作符。
(11) pop(s)　　//操作符取出后，退栈。
(12) sempty(s)　　//将 pre 的最后一个字符（操作数）加入到中缀式 exp 的最后。

四．应用题

1．串是零个至多个字符组成的有限序列。从数据结构角度讲，串属于线性结构。与线性表的特殊性在于串的元素是字符。

2．空格是一个字符，其 ASCII 码值是 32。空格串是由空格组成的串，其长度等于空格的个数。空串是不含任何字符的串，即空串的长度是零。

3．最优的 T(m,n)是 O（n)。串 S2 是串 S1 的子串，且在 S1 中的位置是 1。开始求出最大公共子串的长度恰是串 S2 的长度，一般情况下，T(m,n) =O(m*n)。

4．朴素的模式匹配（Brute－Force）时间复杂度是 O（m＊n），KMP 算法有一定改进，时间复杂度达到 O（m＋n)。本题也可采用从后面匹配的方法，即从右向左扫描，比较 6 次成功。另一种匹配方式是从左往右扫描，但是先比较模式串的最后一个字符，若不等，则模式串后移；若相等，再比较模式串的第一个字符，若第一个字符也相等，则从模式串的第二个字符开始，向右比较，直至相等或失败。若失败，模式串后移，再重复以上过程。按这种方法，本题比较 18 次成功。

5．KMP 算法主要优点是主串指针不回溯。当主串很大不能一次读入内存且经常发生部分匹配时，KMP 算法的优点更为突出.

6．模式串的 next 函数定义如下：

next [j] =

$$\begin{cases} 0 & \text{当} j=1\text{时} \\ \max\{k \mid 1<k<j \text{且} ' p_1 \cdots p_{k-1} ' = ' p_{j-k+1} \cdots p_{j-1} ' \} & \text{当此集合不空时} \\ 1 & \text{其它情况} \end{cases}$$

根据此定义，可求解模式串 t 的 next 和 nextval 值如下：

j	1	2	3	4	5	6	7	8	9	10	11	12
t 串	a	b	c	a	a	b	b	a	b	c	a	b
next[j]	0	1	1	1	2	2	3	1	2	3	4	5
nextval[j]	0	1	1	0	2	1	3	0	1	1	0	5

7．解法同上题 6，其 next 和 nextval 值分别为 0112123422 和 0102010422。

8．解法同题 6，t 串的 next 和 nextval 函数值分别为 0111232 和 0110132。

9．解法同题 6，其 next 和 nextval 值分别为 011123121231 和 011013020131。

10．p1 的 next 和 nextval 值分别为：0112234 和 0102102；p2 的 next 和 nextval 值分别为：

0121123 和 0021002。

11．next 数组值为 011234567　改进后的 next 数组信息值为 010101017。

12．011122312。

13．next 定义见题上面 6 和下面题 20。串 p 的 next 函数值为：01212345634。

14.(1)S 的 next 与 nextval 值分别为 012123456789 和 002002002009,p 的 next 与 nextval 值分别为 012123 和 002003。

（2）利用 BF 算法的匹配过程：

第一趟匹配：aabaabaabaac
　　aabaac(i=6, j=6)

第二趟匹配：aabaabaabaac
　　aa(i=3, j=2)

第三趟匹配：aabaabaabaac
　　a(i=3, j=1)

第四趟匹配：aabaabaabaac
　　aabaac(i=9, j=6)

第五趟匹配：aabaabaabaac
　　aa(i=6, j=2)

第六趟匹配：aabaabaabaac
　　a(i=6, j=1)

第七趟匹配：aabaabaabaac
（成功）　　aabaac(i=13, j=7)

利用 KMP 算法的匹配过程：

第一趟匹配：aabaabaabaac
　　aabaac(i=6, j=6)

第二趟匹配：aabaabaabaac
　　(aa)baac

第三趟匹配：aabaabaabaac
　　（成功）(aa)baac

15．（1）p 的 nextval 函数值为 0110132。（p 的 next 函数值为 0111232）。

（2）利用 KMP(改进的 nextval)算法，每趟匹配过程如下：

第一趟匹配：abcaabbabcabaacbacba
　　abcab(i=5, j=5)

第二趟匹配：abcaabbabcabaacbacba
　　abc(i=7, j=3)

第三趟匹配：abcaabbabcabaacbacba
　　a(i=7, j=1)

第四趟匹配：abcaabbabcabaac bacba
（成功）　　abcabaa(i=15, j=8)

16．KMP 算法的时间复杂性是 O（m+n）。

p 的 next 和 nextval 值分别为 01112212321 和 01102201320。

17．（1）p 的 nextval 函数值为 01010。（next 函数值为 01123）

（2）利用所得 nextval 数值，手工模拟对 s 的匹配过程，与上面 16 题类似，为节省篇幅，故略去。

18．模式串 T 的 next 和 nextval 值分别为 0121123 和 0021002。

19．第 4 行的 p[J]=p[K]语句是测试模式串的第 J 个字符是否等于第 K 个字符，如是，则指针 J 和 K 均增加 1，继续比较。第 6 行的 p[J]=p[K]语句的意义是，当第 J 个字符在模式匹配中失配时，若第 K 个字符和第 J 个字符不等，则下个与主串匹配的字符是第 K 个字符；否则，若第 K 个字符和第 J 个字符相等，则下个与主串匹配的字符是第 K 个字符失配时的下一个（即 NEXTVAL[K]）。

该算法在最坏情况下的时间复杂度 O（m^2）。

20．（1）当模式串中第一个字符与主串中某字符比较不等（失配）时，next[1]=0 表示模式

串中已没有字符可与主串中当前字符 s[i]比较，主串当前指针应后移至下一字符，再和模式串中第一字符进行比较。

（2）当主串第 i 个字符与模式串中第 j 个字符失配时，若主串 i 不回溯，则假定模式串第 k 个字符与主串第 i 个字符比较，k 值应满足条件 1<k<j 并且 '$p_1 \cdots p_{k-1}$' = '$p_{j-k+1} \cdots p_{j-1}$'，即 k 为模式串向后移动的距离，k 值有多个，为了不使向右移动丢失可能的匹配，k 要取大，由于 max{k}表示移动的最大距离，所以取 max{k}，k 的最大值为 j-1。

（3）在上面两种情况外，发生失配时，主串指针 i 不回溯，在最坏情况下，模式串从第 1 个字符开始与主串第 i 个字符比较，以便不致丢失可能的匹配。

21. 这里失败函数 f，即是通常讲的模式串的 next 函数，其定义见本章应用题的第 6 题。

进行模式匹配时，若主串第 i 个字符与模式串第 j 个字符发生失配，主串指针 i 不回溯，和主串第 i 个字符进行比较的是模式串的第 next[j]个字符。模式串的 next 函数值，只依赖于模式串，和主串无关，可以预先求出。

该算法的技术特点是主串指针 i 不回溯。在经常发生“部分匹配”和主串很大不能一次调入内存时，优点特别突出。

22. 失败函数（即 next）的值只取决于模式串自身，若第 j 个字符与主串第 i 个字符失配时，假定主串不回溯，模式串用第 k（即 next[j]）个字符与第 i 个相比，有 ' $p_1 \cdots p_{k-1}$' = '$p_{j-k+1} \cdots p_{j-1}$'，为了不因模式串右移与主串第 i 个字符比较而丢失可能的匹配，对于上式中存在的多个 k 值，应取其中最大的一个。这样，因 j-k 最小，即模式串向右滑动的位数最小，避免因右移造成的可能匹配的丢失。

23. 仅从两串含有相等的字符，不能判定两串是否相等，两串相等的充分必要条件是两串长度相等且对应位置上的字符相同（即两串串值相等）。

24.（1）s1 和 s2 均为空串；（2）两串之一为空串；（3）两串串值相等（即两串长度相等且对应位置上的字符相同）。（4）两串中一个串长是另一个串长（包括串长为 1 仅有一个字符的情况）的数倍，而且长串就好象是由数个短串经过连接操作得到的。

25、题中所给操作的含义如下：

//：连接函数，将两个串连接成一个串

substr（s,i,j）：取子串函数，从串 s 的第 i 个字符开始，取连续 j 个字符形成子串

replace（s1,i,j,s2）：置换函数，用 s2 串替换 s1 串中从第 i 个字符开始的连续 j 个字符

本题有多种解法，下面是其中的一种：

```
（1） s1=substr（s,3,1）      //取出字符：'y'
（2） s2=substr（s,6,1）      //取出字符：'+'
（3） s3=substr（s,1,5）      //取出子串：'(xyz)'
（4） s4=substr（s,7,1）      //取出字符：'*'
（5） s5=replace（s3,3,1,s2）//形成部分串：'(x+z)'
（6） s=s5//s4//s1             //形成串 t 即 '(x+z）*y'
```

五、算法设计

1、[题目分析]判断字符串 t 是否是字符串 s 的子串，称为串的模式匹配，其基本思想是对串 s 和 t 各设一个指针 i 和 j，i 的值域是 0..m-n，j 的值域是 0..n-1。初始值 i 和 j 均为 0。模式匹配从 s_0 和 t_0 开始，若 $s_0=t_0$，则 i 和 j 指针增加 1，若在某个位置 $s_i!=t_j$，则主串指针 i 回溯到 i=i-j+1，j 仍从 0 开始，进行下一轮的比较，直到匹配成功（j>n-1），返回子串在主串的位置（i-j）。否则，当 i>m-n 则为匹配失败。

```
int index(char s[],t[],int m,n)
```

```
//字符串 s 和 t 用一维数组存储，其长度分别为 m 和 n。本算法求字符串 t 在字符串 s 中的第一次出现，如是，输出子串在 s 中的位置，否则输出 0。
{int i=0,j=0;
 while (i<=m-n && j<=n-1)
   if (s[i]==t[j]){i++;j++;}  //对应字符相等，指针后移。
      else {i=i-j+1;j=0;}       //对应字符不相等，I 回溯，j 仍为 0。
   if(i<=m-n && j==n) {printf("t 在 s 串中位置是%d",i-n+1);return(i-n+1);}//匹配成功
   else  return(0); //匹配失败
}//算法 index 结束
main ()//主函数
{char s[],t[]; int m,n,i;
 scanf("%d%d",&m,&n);  //输入两字符串的长度
 scanf("%s",s);        //输入主串
 scanf("%s",t);        //输入子串
 i=index(s,t,m,n);
}//程序结束
```

[程序讨论]因用 C 语言实现，一维数组的下标从 0 开始，m-1 是主串最后一个字符的下标，n-1 是 t 串的最后一个字符的下标。若匹配成功，最佳情况是 s 串的第 0 到第 n-1 个字符与 t 匹配，时间复杂度为 o（n）；匹配成功的最差情况是，每次均在 t 的最后一个字符才失败，直到 s 串的第 m-n 个字符成功，其时间复杂度为 o（(m-n) *n），即 o（m*n）。失败的情况是 s 串的第 m-n 个字符比 t 串某字符比较失败，时间复杂度为 o（m*n）。之所以串 s 的指针 i 最大到 m-n，是因为在 m-n 之后，所剩子串长度已经小于子串长度 n，故不必再去比较。算法中未讨论输入错误（如 s 串长小于 t 串长）。

另外，根据子串的定义，返回值 i-n+1 是子串在主串中的位置，子串在主串中的下标是 i-n。

2.［问题分析］在一个字符串内，统计含多少整数的问题，核心是如何将数从字符串中分离出来。从左到右扫描字符串，初次碰到数字字符时，作为一个整数的开始。然后进行拼数，即将连续出现的数字字符拼成一个整数，直到碰到非数字字符为止，一个整数拼完，存入数组，再准备下一整数，如此下去，直至整个字符串扫描到结束。

```
int CountInt ()
// 从键盘输入字符串，连续的数字字符算作一个整数，统计其中整数的个数。
 {int i=0, a[];      // 整数存储到数组 a，i 记整数个数
  scanf ("%c", &ch); // 从左到右读入字符串
  while (ch!='#')  // '#' 是字符串结束标记
    if (isdigit (ch)) // 是数字字符
     {num=0;         // 数初始化
      while (isdigit (ch) && ch!='#') // 拼数
       {num=num*10+'ch' - '0';
        scanf ("%c", &ch);
       }
      a[i]=num; i++;
      if (ch!='#') scanf ("%c", &ch);  // 若拼数中输入了 '#'，则不再输入
     } // 结束 while (ch! = '#')
```

```
    printf（“共有%d 个整数，它们是:”i);
    for（j=0；j<i；j++)
     {printf（“%6d”，a[j]);
       if（(j+1）%10==0）printf（“\n”);｝// 每 10 个数输出在一行上
}// 算法结束
```

[算法讨论]假定字符串中的数均不超过 32767，否则，需用长整型数组及变量。

3、[题目分析]设以字符数组 s 表示串，重复子串的含义是由一个或多个连续相等的字符组成的子串，其长度用 max 表示，初始长度为 0，将每个局部重复子串的长度与 max 相比，若比 max 大，则需要更新 max，并用 index 记住其开始位置。

```
int LongestString(char s[],int n)
//串用一维数组 s 存储，长度为 n，本算法求最长重复子串，返回其长度。
{int index=0,max=0;  //index 记最长的串在 s 串中的开始位置，max 记其长度
 int length=1,i=0,start=0;   //length 记局部重复子串长度，i 为字符数组下标
 while(i<n-1)
   if(s[i]==s[i+1]) {i++; length++;}
   else //上一个重复子串结束
      {if(max<length) {max=length; index=start;} //当前重复子串长度大，则更新 max
       i++;start=i;length=1;                    //初始化下一重复子串的起始位置和长度
      }
 printf(“最长重复子串的长度为%d，在串中的位置%d\n”，max,index);
 return(max);
}//算法结束
```

[算法讨论]算法中用 i<n-1 来控制循环次数，因 C 数组下标从 0 开始，故长度为 n 的串，其最后一个字符下标是 n-1，当 i 最大为 n-2 时，条件语句中 s[i+1]正好是 s[n-1]，即最后一个字符。子串长度的初值数为 1，表示一个字符自然等于其身。

算法的时间复杂度为 O(n)，每个字符与其后继比较一次。

4、[题目分析]教材中介绍的串置换有两种形式：第一种形式是 replace（s,i,j,t)，含义是将 s 串中从第 i 个字符开始的 j 个字符用 t 串替换,第二种形式是 replace(s,t,v)，含义是将 s 串中所有非重叠的 t 串用 v 代替。我们先讨论第一种形式的替换。因为已经给定顺序存储结构，我们可将 s 串从第（i+j-1）到串尾（即 s.curlen）移动 t.curlen-j 绝对值个位置（以便将 t 串插入)：若 j>t.curlen，则向左移；若 j<t.curlen，则向右移动；若 j=t.curlen，则不必移动。最后将 t 串复制到 s 串的合适位置上。当然，应考虑置换后的溢出问题。

```
int replace(strtp s,t,int i,j)
//s 和 t 是用一维数组存储的串，本算法将 s 串从第 i 个字符开始的连续 j 个字符用 t 串置换，操作成功返回 1，否则返回 0 表示失败。
{if(i<1 || j<0 || t.curlen+s.curlen-j>maxlen)
   {printf(“参数错误\n”);exit(0);}       //检查参数及置换后的长度的合法性。
 if(j<t.curlen) //若 s 串被替换的子串长度小于 t 串长度，则 s 串部分右移,
    for(k=s.curlen-1;k>=i+j-1;k--) s.ch[k+t.curlen-j]=s.ch[k];
 else if (j>t.curlen)      //s 串中被替换子串的长度小于 t 串的长度。
        for(k=i-1+j;k<=s.curlen-1;k++) s.ch[k-(j-t.curlen)]=s.ch[k];
```

```
    for(k=0;k<t.curlen;k++) s.ch[i-1+k]=t.ch[k];  //将 t 串复制到 s 串的适当位置
    if(j>t.curlen)                                s.curlen=s.curlen-(j-t.curlen);else
    s.curlen=s.curlen+(t.curlen-j);
  }//算法结束
```

[算法讨论]若允许使用另一数组，在检查合法性后，可将 s 的第 i 个（不包括 i）之前的子串复制到另一子串如 s1 中，再将 t 串接到 s1 串后面,然后将 s 的第 i+j 直到尾的部分加到 s1 之后。最后将 s1 串复制到 s。主要语句有：

```
for(k=0;k<i;k++) s1.ch[k]=s.ch[k]; //将 s1 第 i 个字符前的子串复制到 s1，这时 k=i-1
for(k=0;k<t.curlen;k++) s1.ch[i+k]=t.ch[k]//将 t 串接到 s1 的尾部
l=s.curlen+t.curlen-j-1;
for(k=s.curlen-1;k>i-1+j;k--);//将子串第 i+j-1 个字符以后的子串复制到 s1
     s1.ch[l--]=s.ch[k]
for(k=0;k<s.curlen+t.curlen-j;k++) s.ch[k]=s1.ch[k];//将结果串放入 s。
```

下面讨论 replace（s,t,v）的算法。该操作的意义是用串 v 替换所有在串 s 中出现的和非空串 t 相等的不重叠的子串。本算法不指定存储结构，只使用串的基本运算。

```
void replace(string s,t,v)
//本算法是串的置换操作，将串 s 中所有非空串 t 相等且不重复的子串用 v 代替。
{i=index(s,t);//判断 s 是否有和 t 相等的子串
 if(i!=0)//串 s 中包含和 t 相等的子串
  {creat(temp,”);            //creat 操作是将串常量（此处为空串）赋值给 temp。
   m=length(t);n=length(s);  //求串 t 和 s 的长度
   while(i!=0)
      {assign(temp,concat(temp,substr(s,1,i-1),v));//用串 v 替换 t 形成部分结果
       assign(s,substr(s,i+m,n-i-m+1));          //将串 s 中串后的部分形成新的 s 串
       n=n-(i-1)-m;                              //求串 s 的长度
       i=index(s,t);                             //在新 s 串中再找串 t 的位置
      }
      assign(s,contact(temp,s));  //将串 temp 和剩余的串 s 连接后再赋值给 s
  }//if 结束
}//算法结束
```

5、[题目分析]本题是字符串的插入问题，要求在字符串 s 的 pos 位置，插入字符串 t。首先应查找字符串 s 的 pos 位置，将第 pos 个字符到字符串 s 尾的子串向后移动字符串 t 的长度，然后将字符串 t 复制到字符串 s 的第 pos 位置后。

对插入位置 pos 要验证其合法性，小于 1 或大于串 s 的长度均为非法，因题目假设给字符串 s 的空间足够大，故对插入不必判溢出。

```
void  insert(char *s,char *t,int pos)
  //将字符串 t 插入字符串 s 的第 pos 个位置。
{int i=1,x=0;  char *p=s,*q=t;  //p，q 分别为字符串 s 和 t 的工作指针
 if(pos<1) {printf(“pos 参数位置非法\n”);exit(0);}
 while(*p!=’\0’ &&i<pos) {p++;i++;} //查 pos 位置
```

```
    //若 pos 小于串 s 长度，则查到 pos 位置时，i=pos。
  if(*p == '/0') {printf("%d 位置大于字符串 s 的长度",pos);exit(0);}
  else       //查找字符串的尾
    while(*p!= '/0') {p++; i++;}  //查到尾时，i 为字符 '\0' 的下标，p 也指向'\0'。
  while(*q!='\0') {q++; x++;}    //查找字符串 t 的长度 x，循环结束时 q 指向'\0'。
  for(j=i;j>=pos ;j--){*(p+x)=*p; p--;}//串 s 的 pos 后的子串右移，空出串 t 的位置。
  q--;  //指针 q 回退到串 t 的最后一个字符
  for(j=1;j<=x;j++) *p--=*q--;  //将 t 串插入到 s 的 pos 位置上
```

[算法讨论] 串 s 的结束标记('\0')也后移了，而串 t 的结尾标记不应插入到 s 中。

6.[题目分析]本题属于查找，待查找元素是字符串（长 4），将查找元素存放在一维数组中。二分检索（即折半查找或对分查找），是首先用一维数组的“中间”元素与被检索元素比较，若相等,则检索成功，否则，根据被检索元素大于或小于中间元素，而在中间元素的右方或左方继续查找，直到检索成功或失败（被检索区间的低端指针大于高端指针）。下面给出类 C 语言的解法

```
typedef struct node
{char data[4];//字符串长 4
}node;
```

非递归过程如下：

```
int binsearch(node string [];int n;char name[4])
//在有 n 个字符串的数组 string 中，二分检索字符串 name。若检索成功，返回 name 在 string 中的下标，否则返回-1。
{int low = 0,high = n - 1;//low 和 high 分别是检索区间的下界和上界
 while(low <= high)
  {mid = (low + high) /2; //取中间位置
   if(strcmp(string[mid],name) ==0) return (mid); //检索成功
   else if(strcmp(string[mid],name)<0) low=mid+1; //到右半部分检索
        else high=mid-1;                          //到左半部分检索
  }
 return 0; //检索失败
}//算法结束
```

最大检索长度为 $\log_2 n$。

7. [题目分析]设字符串存于字符数组 X 中，若转换后的数是负数，字符串的第一个字符必为 '-'，取出的数字字符，通过减去字符零（'0'）的 ASCII 值，变成数，先前取出的数乘上 10 加上本次转换的数形成部分数，直到字符串结束，得到结果。

```
long atoi(char X[])
 //一数字字符串存于字符数组 X 中，本算法将其转换成数
  {long num=0;
   int i=1; //i 为数组下标
   while (X[i]!= '\0') num=10*num+(X[i++]-'0');//当字符串未到尾，进行数的转换
   if(X[0]=='-') return (-num);      //返回负数
   else return ((X[0]-'0')*10+num); //返回正数，第一位若不是负号，则是数字
```

}//算法 atoi 结束

[算法讨论]如是负数，其符号位必在前面，即字符数组的 x[0]，所以在作转换成数时下标 i 从 1 开始，数字字符转换成数使用 X[i]-'0',即字符与'0'的 ASCII 值相减。请注意对返回正整数的处理。

8.[题目分析]本题要求字符串 s1 拆分成字符串 s2 和字符串 s3，要求字符串 s2“按给定长度 n 格式化成两端对齐的字符串”，即长度为 n 且首尾字符不得为空格字符。算法从左到右扫描字符串 s1，找到第一个非空格字符，计数到 n，第 n 个拷入字符串 s2 的字符不得为空格，然后将余下字符复制到字符串 s3 中。

```
void format (char *s1,*s2,*s3)
 //将字符串 s1 拆分成字符串 s2 和字符串 s3，要求字符串 s2 是长 n 且两端对齐
{char *p=s1, *q=s2;
 int i=0;
 while(*p!= '\0' && *p== ' ') p++;//滤掉 s1 左端空格
 if(*p== '\0') {printf("字符串 s1 为空串或空格串\n");exit(0); }
 while( *p!='\0' && i<n){*q=*p; q++; p++; i++;}//字符串 s1 向字符串 s2 中复制
 if(*p =='\0'){ printf("字符串 s1 没有%d 个有效字符\n",n); exit(0);}
 if(*(--q)==' ' ) //若最后一个字符为空格，则需向后找到第一个非空格字符
  {p-- ;           //p 指针也后退
   while(*p==' '&&*p!='\0') p++;//往后查找一个非空格字符作串 s2 的尾字符
     if(*p=='\0') {printf("s1 串没有%d 个两端对齐的字符串\n",n); exit(0);}
   *q=*p;          //字符串 s2 最后一个非空字符
   *(++q)='\0';    //置 s2 字符串结束标记
   }
*q=s3;p++;       //将 s1 串其余部分送字符串 s3。
while (*p!= '\0') {*q=*p; q++; p++;}
*q='\0';         //置串 s3 结束标记
}
```

9.[题目分析]两个串的相等，其定义为两个串的值相等，即串长相等，且对应字符相等是两个串相等的充分必要条件。因此，首先比较串长，在串长相等的前提下，再比较对应字符是否相等。

```
int equal(strtp s,strtp t)
//本算法判断字符串 s 和字符串 t 是否相等，如相等返回 1，否则返回 0
{if (s.curlen!=t.curlen) return (0);
for (i=0; i<s.curlen;i++)      //在类 C 中，一维数组下标从零开始
   if (s.ch[i]!= t.ch[i])return (0);
return (1);   //两串相等
}//算法结束
```

10.［问题分析］由于字母共 26 个，加上数字符号 10 个共 36 个，所以设一长 36 的整型数组，前 10 个分量存放数字字符出现的次数，余下存放字母出现的次数。从字符串中读出数字字符时,字符的 ASCII 代码值减去数字字符‘0’的 ASCII 代码值,得出其数值(0..9)，字母的 ASCII 代码值减去字符‘A’的 ASCII 代码值加上 10，存入其数组的对应下标分量中。遇其它符号不作处理，直至输入字符串结束。

```
void Count（）
//统计输入字符串中数字字符和字母字符的个数。
```

```
{int i，num[36];
 char ch;
 for（i＝0; i<36; i++）num[i]＝0；// 初始化
 while（(ch＝getchar（)) != ‘#’）  // ‘#’表示输入字符串结束。
     if（‘0’ <=ch<= ‘9’）{i=ch－48;num[i]++;}          // 数字字符
     else  if（‘A’ <=ch<= ‘Z’）{i=ch-65+10;num[i]++;} // 字母字符

 for（i=0; i<10; i++）   // 输出数字字符的个数
       printf（“数字%d 的个数＝%d\n”, i，num[i]);
 for（i＝10; i<36; i++）// 求出字母字符的个数
       printf（“字母字符%c 的个数＝%d\n”, i＋55，num[i]);
} // 算法结束。
```

11.[题目分析]实现字符串的逆置并不难，但本题“要求不另设串存储空间”来实现字符串逆序存储，即第一个输入的字符最后存储，最后输入的字符先存储，使用递归可容易做到。

```
 void InvertStore(char A[])
//字符串逆序存储的递归算法。
{ char ch;
  static int i = 0;//需要使用静态变量
  scanf ("%c",&ch);
  if (ch!= '.')    //规定'.'是字符串输入结束标志
    {InvertStore(A);
     A[i++] = ch;//字符串逆序存储
    }
  A[i] = '\0';  //字符串结尾标记
}//结束算法 InvertStore。
```

12. 串 s'''可以看作由以下两部分组成：'caabcbca...a'和 'ca...a'，设这两部分分别叫串 s1 和串 s2，要设法从 s,s' 和 s''中得到这两部分，然后使用联接操作联接 s1 和 s2 得到 s''' 。

```
i=index(s,s');  //利用串 s'求串 s1 在串 s 中的起始位置
s1=substr(s,i,length(s) - i + 1); //取出串 s1
j=index(s,s''); //求串 s''在串 s 中的起始位置，s 串中'bcb'后是'ca...a')
s2=substr(s,j+3,length(s) - j - 2); //形成串 s2
s3=concat(s1,s2);
```

13.[题目分析]对读入的字符串的第奇数个字符，直接放在数组前面，对第偶数个字符，先入栈，到读字符串结束，再将栈中字符出栈，送入数组中。限于篇幅，这里编写算法，未编程序。

```
void RearrangeString()
 //对字符串改造，将第偶数个字符放在串的后半部分，第奇数个字符前半部分。
{char ch,s[],stk[];            //s 和 stk 是字符数组（表示字符串）和字符栈
 int i=1,j;                    //i 和 j 字符串和字符栈指针
 while((ch=getchar())!=' #' )// ' #' 是字符串结束标志
   s[i++]=ch;                  //读入字符串
 s[i]=' \0' ;                  //字符数组中字符串结束标志
```

```
 i=1;j=1;
 while(s[i])                    //改造字符串
   {if(i%2==0) stk[i/2]=s[i]; else s[j++]=s[i];
    i++;  }//while
 i--; i=i/2;                          //i 先从’\0’后退，是第偶数字符的个数
 while(i>0) s[j++]=stk[i--]    //将第偶数个字符逆序填入原字符数组
}
```

14.[题目分析]本题是对字符串表达式的处理问题，首先定义 4 种数据结构：符号的类码，符号的 TOKEN 表示，变量名表 NAMEL 和常量表 CONSL。这四种数据结构均定义成结构体形式，数据部分用一维数组存储，同时用指针指出数据的个数。算法思想是从左到右扫描表达式，对读出的字符，先查出其符号类码：若是变量或常量，就到变量名表和常量表中去查是否已有，若无，则在相应表中增加之，并返回该字符在变量名表或常量表中的下标；若是操作符，则去查其符号类码。对读出的每个符号，均填写其 TOKEN 表。如此下去，直到表达式处理完毕。先定义各数据结构如下。

```
struct // 定义符号类别数据结构
   {char  data[7];   //符号
    char  code[7];   //符号类码
   }TYPL;
typedef  struct      //定义 TOKEN 的元素
   {int typ;         //符号码
    int addr;        //变量、常量在名字表中的地址
    }cmp;
struct {cmp data[50]; //定义 TOKEN 表长度<50
        int last;     //表达式元素个数
        }TOKEN;
struct {char data[15];   //设变量个数小于 15 个
        int last;        //名字表变量个数
       }NAMEL;
struct {char data[15];   //设常量个数小于 15 个
        int  last;       //常量个数
       }CONSL;
int operator (char cr)
 //查符号在类码表中的序号
 {for (i=3; i<=6; i++)
  if (TYPL.data[i]==cr)  return (i);
 }
void PROCeString ()
 //从键盘读入字符串表达式（以‘#’结束），输出其 TOKEN 表示。
 {NAMEL.last=CONSL.last=TOKEN.last=0;    //各表元素个数初始化为 0
  TYPL.data[3]= ‘*’; TYPL.data[4]= ‘+’; TYPL.data[5]= ‘(’;
  TYPL.data[6]= ‘)’;      //将操作符存入数组
  TYPL.code[3]= ‘3’; TYPL.code[4]= ‘4’; TYPL.code[5]= ‘5’;
  TYPL.code[6]= ‘6’;  //将符号的类码存入数组
  scanf (“%c”, &ch);    //从左到右扫描（读入）表达式。
```

```
  while（ch!＝‘#’）    //‘#’是表达式结束符
    {switch（ch）of
      {case‘A’：case  ‘B’：case  ‘C’：//ch 是变量
                TY=0;        //变量类码为 0
                 for（i=1; i<=NAMEL.last; i++）
                   if（NAMEL.data[i]==ch）break; //已有该变量，i 记住其位置
                 if（i>NAMEL.last）{NAMEL.data[i]=ch; NAMEL.last++; }//变量
加入
       case‘0’：case‘1’：case‘2’：case‘3’：case‘4’：case‘5’://处
理常量
       case‘6’：case  ‘7’:case‘8’：case‘9’：TY=1; //常量类码为 1
                for（i=1; i<=CONSL.last; i++）
                  if（CONSL.data[i]==ch）break; ////已有该常量，i 记住其位
置
                if（i>CONSL.last）{CONSL.data[i]=ch; CONSL.last++;} //将新
常量加入
       default:    //处理运算符
                   TY＝operator（ch）; //类码序号
                   i=’\0’;          //填入 TOKEN 的 addr 域（期望输出空白）
      }//结束 switch，下面将 ch 填入 TOKEN 表
     TOKEN.data［++TOKEN.last］.typ=TY; TOKEN.data［TOKEN.last］.addr=i;
     scanf（“%c”，&ch);   //读入表达式的下一符号。
   }//while
 }//算法结束
```

[程序讨论]为便于讨论，各一维数组下标均以 1 开始，在字符为变量或常量的情况下，将其类码用 TY 记下，用 i 记下其 NAMEL 表或 CONSL 表中的位置，以便在填 TOKEN 表时用。在运算符（‘+’，‘*’，‘(’，‘)’）填入 TOKEN 表时，TOKEN 表的 addr 域没意义，为了程序统一，这里填入了’\0’。本题是表达式处理的简化情况（只有 3 个单字母变量，常量只有 0..9，操作符只 4 个），若是真实情况，所用数据结构要相应变化。