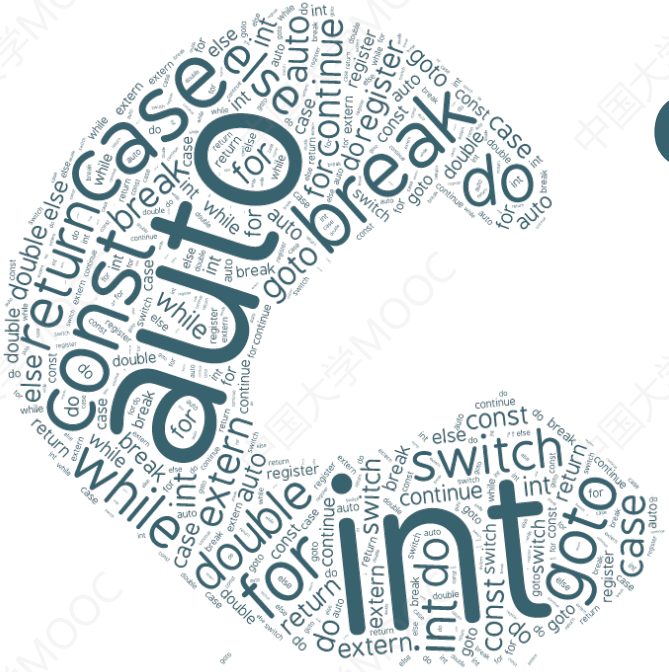


chapter 7

字符串

目录
content

7.1 字符串的基本概念

字符：任何计算机系统都使用一个可被本系统识别的字符集，该字符集包括了人们常用的字母、数字以及诸如句号、逗号、括号之类的特殊字符。

国际上较通用的字符集是"美国标准信息交换代码”（The American Standard Code for Information Interchange, ASCII）字符集，已被计算机等行业广泛接受为标准。

ASCII字符集（127个）

- **字母**：大写字母A~Z，小写字母a~z
- **数字**：0~9
- **特殊字符** (29个)：

  ！ # & * （） - + = _ ， . / ? < > : ; ’ ” | \ [] {} ~ ` ^
- **空格符**：空格、水平制表符（tab）、垂直制表符、换行、换页
- **不能显示的字符**：空字符（' \0 '）、退格(' \b ')、回车（' \r '）等

7.1.1 字符

字符类型	字符表示	字符含义	ASCII码值	“\ddd”表示	“\xhh”表示
字母	‘a’	字母（a）	97	\141	\x61
数字	‘1’	数字（1）	49	\061	\x31
特殊字符	‘!’	感叹号符（!）	33	\041	\x21
	‘\’’	单引号符（’）	96	\140	\x60
	‘\”’	双引号符（”）	34	\042	\x22
	‘\\’	反斜线符（\）	92	\134	\x5C
空格符	‘\n’	回车换行	10	\012	\x0A
	‘\f’	走纸换页	12	\014	\x0C
不能显示的字符	‘\0’	空字符	0	\000	\x00
	‘\b’	退格	8	\010	\x08
	‘\r’	回车	13	\015	\x0D

字符

数字字符 和 **数值型数字** 之间的关系：

1) char a = ' 3 ' ;　　int b = 3;

内存：0x33　　3

以字符型数据输出 ‘3’

以整型数据输出 51

字符型数据和整型数据可以通用，相当于对字符的ASCII码进行操作。

2) char a[20] = " 10000 ";　　int b = 10000;

内存：0x31 30 30 30 30　　0x27 10

数字字符按ASCII值转换为二进制存储在存储空间中，而数值型数字则直接按二进制形式存储在存储空间中。

一般用数值型数字表示数字的变量占用较小的内存。

整数运算　　1+1=2

字符运算　　‘1’ + ‘1’ =0x31 + 0x31=0x62

7.1.2 字符串

字符串：由一对双引号括起的字符序列，字符串中可以包括字母、数字以及各种各样的字符等等。

e.g.

"zhang san"　　表示一个人名

"hust.wuhan.china"　　表示一个地址

"87243092"　　表示一个电话号码

在有效字符串的末尾存放 '\0' 作为字符串结束的标志。

可以使用字符数组或字符指针来对字符串进行处理

字符数组：char color[20] = "blue";

字符指针：char *colorPtr = "blue";

使用字符指针与字符型数组本质上是具有相通性的，都是对字符串的地址进行操作，而使用字符指针更加方便易懂

7.1.3 字符数组与字符指针

字符数组：元素类型为字符型的数组，字符数组中的一个元素存放一个字符。

字符数组的定义方法： char string[20];

一个长度为n的字符数组可以存储长度不超过n-1个字符的字符串，因为在有效字符串的末尾会存入一个空字符 ‘\0’ ，但它并不是字符串的一部分，所以字符串的最大长度始终比数组长度小1。

7.1.3 字符数组与字符指针

字符数组在处理字符串时的初始化方法

方法1：逐个元素赋初值

```
char string[20]={ 's' , 't' , 'r' , 'o' , 'n' , 'g' , '\0' };
```

等效

```
string[0]= 's';
string[1]= 't';
string[2]= 'r';
      .
      .
string[6]= '\0';
```

在指定初值时，必须在最后一个值之后必须明确地写上 ‘\0’

方法2：整个字符串赋初值

```
char string[20]="hello";
```

自动在末尾加有 '\0 '字符，作为一个结束标志

在内存中的存储状态

7.1.3 ——一维字符数组与单个字符串

例7.1 计算字符串长度的程序

```
#include <stdio.h>
int length (char *string);
void main()
{
    int m;
    char a[20];
    gets(a);
    m = length(a);
    printf("The length of string a is %d\n",m);
}
int length (char *string)
{
    char * p = string;
    while(* p ++ !='\0')
        ;
    return (p-string);
}
```

gets函数用于输入可以包含空格的字符串

统计字符的个数，如果字符不等于 ‘\0’ 则循环

运行结果：

Hello world!

The length of string a is 12

7.1.3 ——二维字符数组与多个字符串

例7.2 二维字符数组在处理多个字符串时的的定义和初始化方法

[方法一]

```
char string[3][10]={  "pascal",
                      "cobol" ,
                      "fortran" };
```

多个字符串的显式初始化

[方法二]

```
char string[3][20];
int i;
for(i=0;i<3;i++)
{
    gets(string[i]);
}
for(i=0;i<3;i++)
{
    printf("%s\n", string[i]);
}
```

利用scanf函数进行循环输入
实现多个字符串的初始化

存储状态：

p	a	s	c	a	l	\0			
c	o	b	o	l	\0				
f	o	r	t	r	a	n	\0		

7.1.3 ——字符指针

字符指针：即字符型指针，它是一个指针变量，可以方便地对字符串中的字符进行处理。

字符指针的定义方法：char *string;

字符指针初始化：

方法1：直接使用字符串常量作为初始值

```
char *string = "we are family!";
```

方法2：将一个字符串常量赋予一个指针

```
char  *p;
p="c  program";
```

① 系统开辟一块区域存储这个字符串

② 将首地址赋予指针

由于这块区域已经固定，因此如果要处理该常量字符串时，要保证处理后的字符串长度不能超过初始的字符串长度，否则会修改初始字符串所在区域后面的数据，有可能会影响系统的运行。

在使用字符数组和字符指针时应注意的问题：

字符指针：

```
char *p;
scanf("%s",p);
```

scanf(“%s”,p)是将输入的字符串存放在p所指向的地址中，在p未赋值之前，执行scanf(“%s”,p)是错误的

字符数组：

```
char  name[20];
name="c  program";
```

因为name是个地址常量，系统不允许向它赋值，正确的初始化形式为

```
char  name[20] ="c  program";
```

——字符指针

例7.3 向字符指针赋字符串

```
#include <stdio.h>
void main( )
{
    char *s="good";
    char *p;
    while(*s!='\0')
        printf("%c",*s++);
    printf("\n");
    p="morning";
    while(*p!='\0')
        printf("%c",*p++);
    printf("\n");
}
```

直接使用字符串常量作为初始值进行初始化

将一个字符串常量赋予一个指针完成字符串的初始化

运行结果：

good

morning

例7.4 输入一行字符串，将其逆序输出

```
#include <stdio.h>
void reverse(char *sPtr);
void main()
{
    char s[80];
    printf("Enter a line of text:\n");
    gets(s);
    printf("\nThe line printed backward is\n");
    reverse(s);
}
void reverse(char *sPtr)
{
    if(sPtr[0]=='\0')
    {
        return;
    }
    else
    {
        reverse(&sPtr[1]);
        putchar(sPtr[0]);
    }
}
```

运行结果：

```
Enter a line of text:
Hello world!
The line printed backward is:
!dlrow olleH
```

利用字符指针将单个字符串在函数间传递

reverse函数递归调用，从最后的字符开始，实现逆序输出

7.2 字符串相关库函数及其使用

相关库函数	函数声明	函数功能
字符串输入输出函数	char *gets (char *s);	字符串输入
	int puts (const char *s);	字符串输出
字符串转换函数	double atof (const char *nPtr);	字符串转换为浮点数
	int atoi (const char *nPtr);	字符串转换为整数
	long atol (const char *nPtr);	字符串转换为长整型
	void itoa(int n,char s[],int radix);	整数n转换为字符串
字符串处理函数	char *strcpy(char *dest,char *src);	复制字符串
	char *strcat(char *dest,char *src);	连接字符串
字符串比较函数	int strcmp(char *s1,char *s2);	比较字符串
其他函数	int strlen(char *s);	求字符串长度
	char *strupr(char *str); char *strlwr(char *str);	字符串大小写转换
	char *strstr(char *s1,char *s2);	指定字符串在给定字符串中第一次出现的位置

7.2.1 字符串输入输出函数

- **字符串输入函数　　char *gets (char *s);**
  - 功能：从键盘输入一个字符串到字符数组

```
执行：    char string[80];
          gets(string);
键盘输入：  good↙
```

- **字符串输出函数　　int puts (const char *s);**
  - 功能：将一个字符串输出到终端

```
执行：    char string[]={"hello"};
          puts(string);
```

7.2.2 字符串转换函数

- **double atof (const char *nPtr);**
  - 功能：将字符串转换为浮点数的函数
- **int atoi (const char *nPtr);**
  - 功能：将字符串转换为整数的函数
- **long atol (const char *nPtr);**
  - 功能：将字符串转换为长整形的函数
- **void itoa(int n,char s[],int radix);**
  - 功能：将整数n转换为字符串的函数

7.2.2 字符串转换函数

例7.5 将某一个整数转换为相对应的数字串

```
#include <stdio.h>
#define LENGTH 6
void reverse(char *s);
void itoa(int n,char *s);
void main(void)
{
   int n;
   char s[LENGTH];
   printf("input a integer:");
   scanf("%d",&n);
   itoa(n,s);
   printf("string: %s\n",s);
}
void itoa(int n,char *s)
{
   int i,sign;
   if((sign=n)<0)
   {
     n=-n;
   }
   i=0;
   do
   {
     s[i++] = n%10 + '0';
   }while((n/=10)>0);
   if(sign<0)
   {
     s[i++]='-';
   }
   s[i]='\0';
   reverse(s);
}
void reverse(char *s)
{
   int i, j, k;
   for(i=0,j = strlen(s)-1;i<j;i++,j--)
   {
     k=s[i];
     s[i]=s[j];
     s[j]=k;
   }
}
```

将输入的整数n转换为字符串

n为负值，转换为正数

字符串翻转函数

运行结果：
input a integer: -102
string: -102

7.2.3 字符串处理函数

- **字符串复制函数　　char *strcpy(char *dest,char *src);**

  - 功能：将指针src所指向的字符串复制到指针dest所指向的内存区域中，函数返回dest所指向的字符串的首地址。

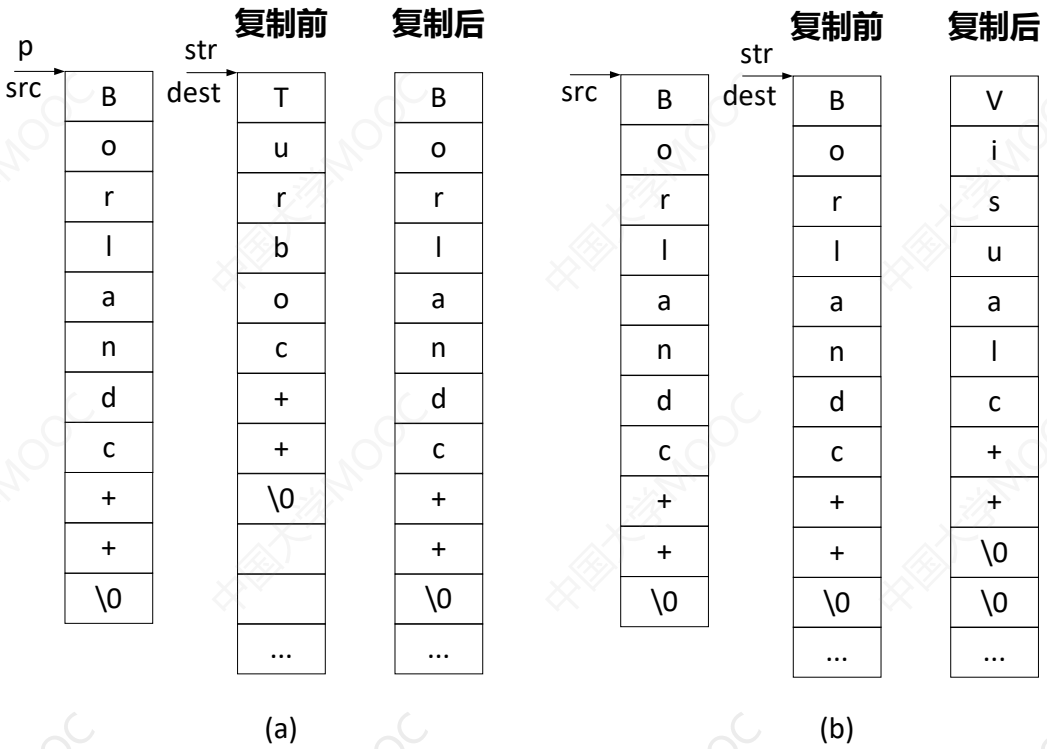
```
char *strcpy(char *dest,char *src)
{
        char *temp = dest;
        while((*dest++=*src++)!='\0')
                ;
        return temp;
}
```

例7.6 字符串复制函数strcpy的使用

```
#include <string.h>
#include <stdio.h>
void main()
{
    char str[80]="TurboC++";
    char *p="BorlandC++";
    strcpy(str,p);
    strcpy(str,"VisualC++");
    printf("%s\n",str);
}
```

第一次调用strcpy的复制过程如图（a）所示

第二次调用strcpy的复制过程如图（b）所示



字符串处理函数

字符串连接函数　　char *strcat(char *dest,char *src);

- 功能：strcat函数的功能是将两个字符串连接。具体的就是把字符串2或者字符数组2中的字符串连接到字符数组1中的字符串的后面，结果存放在字符数组1中。

```
char *strcat(char *dest,char *src)
{
        char *temp = dest;
        while(*dest!='\0') dest++;
        while(*src!='\0') *dest++ = *src++;
        *dest = '\0';
        return temp;
}
```

7.2.3 字符串处理函数

字符串连接函数strcat的使用

```
char string1[80]={"good morning,"};
char string2[ ]={"everyone!"};
strcat(string1,string2);
printf("%s",string1);
```

运行结果：
good morning,everyone!

- 字符数组1必须足够大，以便容纳连接后的新字符串
- 连接前两个字符串的后面都有一个'\0'，连接时将字符串1后面的'\0'去掉了，只在新串最后保留一个'\0'。

7.2.4 字符串比较函数

- **字符串比较函数　　int strcmp(char *s1,char *s2);**

字符串比较，即对两个字符串从第一个字符开始逐个字符相比（按照ASCII码大小比较），直到出现第一个不同的字符或遇到'\0'为止。

（1）如果字符串1中的字符ASCII码较大，则认为字符串1大于字符串2，函数返回值为一正整数。
（2）如果字符串1中的字符ASCII码较小，则认为字符串1小于字符串2，函数返回值为一负整数。
（3）如果全部字符相等，则认为两字符串相等，函数返回值为0。

```
int strcmp(char *s1,char *s2)
{
      for(; *s1==*s2 && *s1;s1++,s2++)
                  ;
       return  *s1 - *s2;
}
```

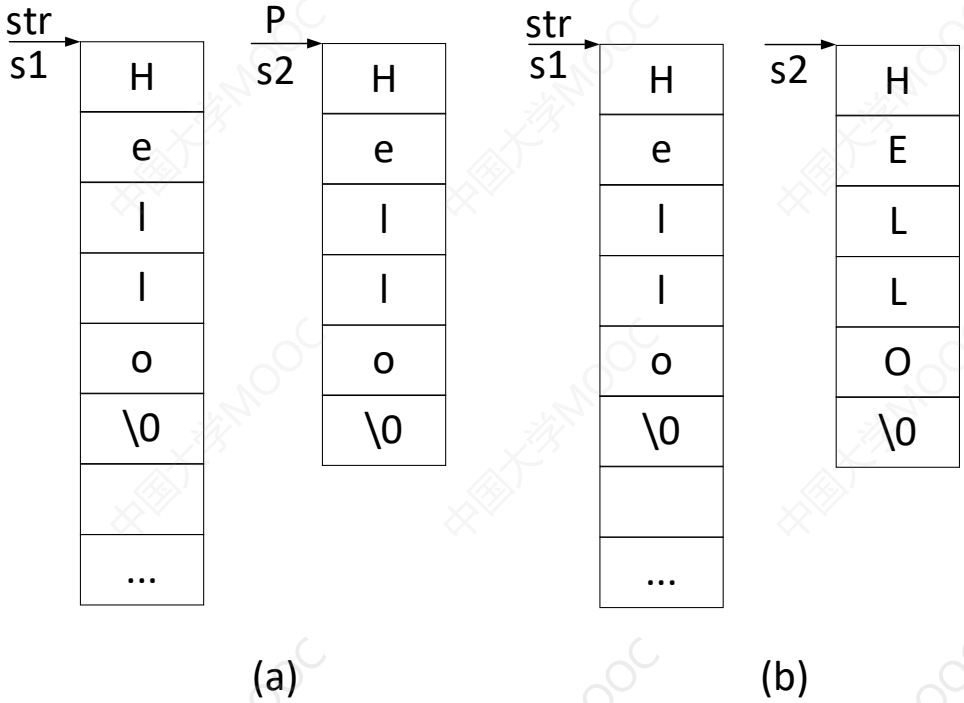
7.2.4 字符串比较函数

例7.7 字符串比较函数strcmp的使用

```
#include <string.h>
#include <stdio.h>
void main()
{
    char str[80]="Hello";
    char *p="Hello";
    if(strcmp(str,p)==0)
        printf("two strings are equal\n");
    if(strcmp(str,"HELLO")>0)
        printf("string(str)is larger\n ");
}
```

第一次调用strcmp的比较过程如图（a）所示

第二次调用strcmp的比较过程如图（b）所示



7.2.5 其它函数

- **求字符串的长度函数　int strlen(char *s);**
  - 功能：求有效字符串的长度

```
int strlen(char *s)
{
    char *p=s;
    while(*p!='\0')
        p++;
    return p-s;
}
```

- **字符串大小写转换函数**

char *strupr(char *str);
char *strlwr(char *str);

  - 功能：strlwr函数将字符串中的大写字母换成小写字母。
    strupr函数将字符串中的小写字母换成大写字母。

7.2.5 其它函数

字符串大写转换函数

```
char *strupr(char *str)
 {
     char *str1=str;
     while (*str1!='\0')
     {
         if(*str1>='a'&& *str1<='z')
             *str1 -= 0x20;
         str1++;
     }
    return str;
}
```

字符串小写转换函数

```
char *strlwr(char *str)
{
    char *str2=str;
    while (*str2!='\0')
    {
        if(*str2>='A'&& *str2<='Z')
            *str2 += 0x20;
        str2++;
    }
    return str;
}
```

‘a’~‘z’的ASCII码值为0x61~0x7A， ‘A’~‘Z’的ASCII码值为0x41~0x5A，相差0x20

7.2.5 其它函数

- **查找指定字符串在给定字符串中第一次出现的函数**

char *strstr(char *s1,char *s2);

- 功能：这个函数在s1中查找整个s2第一次出现的起始位置，并返回一个指向该位置的指针。如果s2并没有完整地出现在s1的任何地方，函数将返回一个NULL指针。如果第二个参数是一个空字符串，函数就返回s1。

其它函数

例7.8 在给定字符串中查找子串最后（最右）一次出现的位置

```
#include <stdio.h>
#include <string.h>
char *my_strstr(char *s1,char *s2)
{
        char *last;
        char *current;
        last = NULL;
        if( *s2 != '\0')
        {
          current = strstr(s1,s2);
          while (current!=NULL)
          {
            last = current;
            current = strstr(last+1,s2);
          }
        }
        return last;
}
```

调用函数strstr查找s2在s1中第一次出现的位置，找到，返回找到的位置，否则，返回NULL

然后查找该字符串下一个匹配位置

7.3 单个字符串的处理

对字符串进行处理时与一般数组的不同之处：

	数组	字符串
定义	int a[n];	char s[n];
函数参数传递	数组名 数组长度	字符串首地址

由于‘\0’作为字符串的结束标志，所以其长度已经隐藏在其中

一般情况下，字符串的赋值需要通过输入输出函数或者标准库函数来实现；而常量字符串可以采用更为简洁的方法，即直接将常量字符串赋予一个指针。

一般单个字符串可以利用字符指针（动态内存分配）、字符数组来处理，而常量字符串则可以直接使用字符指针进行处理。

7.3 ——利用指针处理单个常量字符串

例7.9 在给定字符串中查找特定的字符

```
#include <stdio.h>
#include <string.h>
char *strchr1(char *str,char ch);
void main()
{
    char *string = "this is a string";
    char *pstr, ch= 'c';
    pstr=strchr1(string,ch);
    if(pstr)
        printf("the character %c is at the position:
            %d\n",ch, pstr-string+1);
    else
        printf("the character %c is not found\n",ch);
}
```

对常量字符串的处理

```
char *strchr1(char *str,char ch)
{
    char *temp=NULL;
    for(;*str!= '\0';str++)
    if(*str == ch)
    {
        temp=str;
        break;
    }
    return temp;
}
```

字符串的结束标志'\0'

运行结果：
the character c is not found

例7.10 删除给定字符串中的数字字符

```
#include <stdio.h>
#include <string.h>
char *delnum (char *s);
void main()
{
    char string[80];
    printf("input string:\n");
    gets(string);
    puts(delnum(string));
}
```

使用字符数组名作为实参进行参数传递

```
char *delnum(char *s)
{
    int i;
    char *temp = s;
    for(i=0; s[i]!='\0';)
    {
        if(s[i]>='0'&&s[i]<='9')
            strcpy(s,s+1);
        else
            s++;
    }
    return temp;
}
```

保存字符串的首地址

运行结果：

```
input string:
I have 12 dreams
I have  dreams
```

7.3 ——利用动态内存分配处理单个字符串

内存分配

静态内存分配：编译器在处理程序源代码时（编译时）分配

动态内存分配：程序执行时调用运行时刻库函数进行分配

```
void * malloc (unsigned size);
void free (void * ptr);
```

利用malloc()函数在堆区中开辟内存空间，并将该内存空间的首地址存放在指针中返回，当该字符串使用完后，应立即用配对的标准库函数free()释放这一内存空间

内存是计算机系统的重要资源之一，必须有效地加以管理以求得最佳使用效率，而动态内存技术是有效管理内存资源的最好方法。

7.3 ——利用动态内存分配处理单个字符串

例7.11 删除字符串中除了前导*外的其他*号

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void del(char *s,char *p);
void main()
{
    char str[20],*p;
    printf("Enter a string:\n");
    gets(str);
    if((p=(char *)malloc(strlen(str)+1))==NULL)
    {
        printf("\n not enough memory to allocate
            buffer! ");
        exit(1);
    }
    del(str,p);
    strcpy(str,p); free(p);
    printf("%s\n",str);
    getch();
}
```

保留前导的*号

动态内存分配

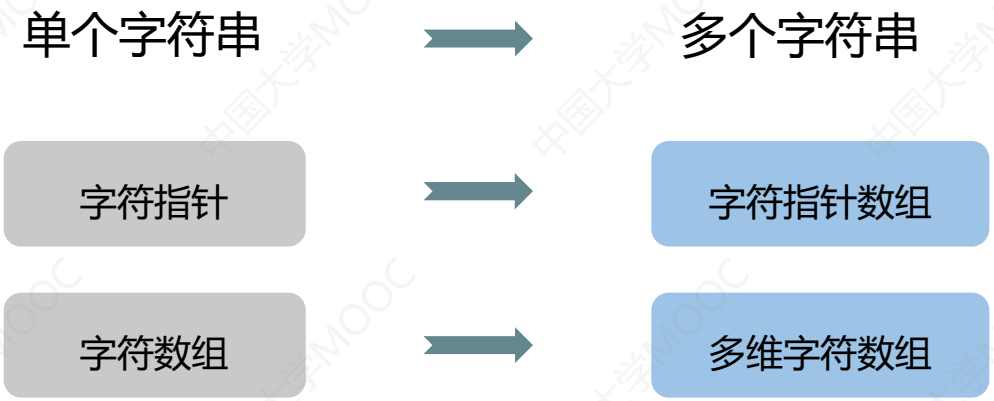
将处理后的字符串占用的内存释放掉

```
void del(char *s,char *p)
{
    while(*s&&*s=='*')
    {
        *p=*s; p++; s++;
    }
    while(*s)
    {
        if(*s!='*')
        {
            *p=*s; p++;
        }
        s++;
    }
    *p='\0';
}
```

运行结果：

```
Enter a string:
****A*BC*DEF****
****ABCDEF
```

7.4 多个字符串的处理



与单个字符串类似的是，一般多个字符串可以利用字符指针数组（动态内存分配）、多维字符数组来处理，而多个常量字符串则可以直接使用字符指针数组处理。

7.4 ——利用字符指针数组处理多个常量字符串

例7.12 用字符指针数组处理多个字符串的排序问题

```
#include <stdio.h>
#include <string.h>
void sortstr(char **v, int n);
void main( )
{
    char *proname[ ]={"pascal","basic","cobol","prolog","lisp"};
    int i;
    sortstr(proname,5);
    for(i=0;i<5;i++)
            printf("%s\n",proname[i]);
}
```

字符指针数组用来存储多个常量字符串

运行结果：

```
basic
cobol
lisp
pascal
prolog
```

形参v是一个二级指针

```
void sortstr(char **v, int n)
{
    int i,j;
    char * temp;
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
        {
            if(strcmp(v[i],v[j])>=0)
            {
                temp=v[i];
                v[i]=v[j];
                v[j]=temp;
            }
        }
}
```

7.4 ——利用多维字符数组处理多个字符串

例7.13 使用选择法对输入的字符串进行升序排序

```
#include <stdio.h>
#include <string.h>
void inpstr(char (*p)[80],int n);
void sortstr(char (*p)[80],int n);
void outpstr(char (*p)[80],int n);
void main()
{
        char str[10][80];
        inpstr(str,10);
        sortstr(str,10);
        outpstr(str,10);
}
void inpstr(char (*p)[80],int n)
{
      int i;
      for(i=0;i<n;i++)
        gets(p[i]);
}
```

数组指针作形参，二维数组名作实参

```
void sortstr(char (*p)[80],int n)
{
      int i,j;
      char temp[80];
       for(i=0;i<n-1;i++)
         for(j=i+1;j<n;j++)
           if(strcmp(p[i],p[j])>0)
           {
              strcpy(temp,p[i]);
              strcpy(p[i],p[j]);
              strcpy(p[j],temp);
           }
}
void outpstr(char (*p)[80],int n)
{
      int i;
      for(i=0;i<n;i++) puts(p[i]);
}
```

采用选择法进行排序

例7.13 使用选择法对输入的字符串进行升序排序

输入：

publisher
word
excel
access
frontpage
outlook
onenote
infopath
powerpoint
visio

输出：

access
excel
frontpage
infopath
onenote
outlook
powerpoint
publisher
visio
word

7.4 ——利用动态内存分配处理多个字符串

例7.14 使用冒泡法对输入的字符串进行升序排序

```
#include <stdio.h>
#include <stdlib.h>
void main()
 {
      int i;
      char*str[10]={NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL};
      char temp[100];
      for(i=0;i<10;i++)
      {
            gets(temp);
            str[i]=(char *)malloc(sizeof(temp));
            if(str[i]==NULL)
            {    printf("not enough memory to allocte buffer.\n");  exit(1);    }
            strcpy(str[i],temp);
      }
      sort(str,10);
      for(i=0;i<10;i++)    puts(str[i]);
      for(i=0;i<10;i++)
      {
           if(str[i]!=NULL)
           free(str[i]);
      }
}
```

根据字符串长度动态分配内存，
并根据字符串长度动态分配内存

例7.14 使用冒泡法对输入的字符串进行升序排序

```
void sort(char **str,int n)
{
    int i,j;
    char *p;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-1-i;j++)
        {
            if(strcmp(str[j],str[j+1])>=0)
            {
                p=str[j];
                str[j]=str[j+1];
                str[j+1]=p;
            }
        }
    }
}
```

采用冒泡法进行排序

排序具体实现的方法是相应指针的互换

运行结果：
运行结果与例7.13相类似

7.5 带参数的main函数

命令行：在操作系统状态下，为了执行某个程序而键入的一行字符

copy file.txt file2.txt <CR>

命令名　　命令行参数　　结束符

```
int main(int argc, char *argv[])
{
   …
}
```

- argc的值是命令行中包括命令在内的所有参数的个数之和。
- 指针数组argv[ ]的各个指针分别指向命令中命令名和各个参数的字符串。其中指针argv[0]总是指向命令名字符串，从argv[1]开始依次指向按先后顺序出现的命令行参数字符串。

7.5 带参数的main函数

例如

C语言程序test带有三个命令行参数，
其命令行是：

test progl.c prog2.c /p <CR>

则 argc初始化为4
argv初始化为
argv[0]="test";
argv[1]="progl.c";
argv[2]="prog2.c";
argv[3]="/p";
argv[4]=0;

最后一个参数是编译系统为了程序处理的方便

例7.15 打印命令行参数的程序

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv)
{
    while(*++argv!=NULL)
    printf("%s\n",*argv);
    return 1;
}
```

打印命令行参数，直至遇到NULL指针

7.6 综合举例

例7.16 输入若干个字符串，统计其中各种字符的个数，然后按统计的个数从大到小输出统计结果

问题分析与设计：

模块划分：该问题可以分成四个功能块：输入、统计、排序和输出。

数据结构：多个字符串采用二维字符数组存储char str[N][80]；字符种类分大写字母、小写字母、数字字符、空格、其它字符5种，其个数的统计结果存放在一维整型数组中int result[5]；由于排序输出时需显示相对应的字符种类的名称，将其存放在一个二维字符数组char name[5][20] 中。

7.6 综合举例

例7.16 输入若干个字符串，统计其中各种字符的个数，然后按统计的个数从大到小输出统计结果

函数原型：

(1) void input(char (*p)[80],int n);

功能：多个字符串的输入

形参：字符数组指针用来传递多个字符串，n用来传递输入字符串的个数

(2) void statistic(char (*p)[80],int n,int *presult);

功能：统计字符串中各种字符的个数

形参：字符数组指针用来接收实参传递过来的多个字符串，n用来传递输入字符串的个数，一级整型指针用来接收传递各种字符的个数

(3) void sort(int *presult, int n, char (*pname)[20]);

功能：多个字符串的排序

形参：一级整型指针用来传递各种字符的个数，n用来传递字符种类数，字符数组指针用来传递字符种类名称

(4) void output(int *presult, int n, char (*pname)[20]);

功能：多个字符串的输出

综合举例

例7.16　输入若干个字符串，统计其中各种字符的个数，然后按统计的个数从大到小输出统计结果

```
#include <stdio.h>
#include <string.h>
#define N  10
void input(char (*p)[80],int n);
void statistic(char (*p)[80],int n,int *presult);
void sort(int *presult, int n, char (*pname)[20]);
void output(int *presult, int n, char (*pname)[20]);
void main()
{
  char str[N][80];
  char name[5][20]=    {"capital","lowercase","digital","space","other "};
  int result[5];

  input(str,N);
  statistic(str,N,result);
  sort(result,5,name);
  output(result,5,name);
  return ;
}
```

宏定义字符串的个数N为10

函数声明

函数调用

多个字符串的输入、统计、排序、输出

综合举例

多个字符串输入函数

```
void input(char (*p)[80],int n)
{
    int i;

    for(i=0;i<n;i++)
    {
       printf("input %d th string:\n",i+1);
       gets(p[i]);
    }
    return;
}
```

多个字符串统计函数

```
void statistic(char (*p)[80],int n,int *presult)
{
    int i,j;
    for(i=0;i<5;i++)
       presult[i]=0;
    for(i=0;i<n;i++)
    {
       for(j=0; p[i][j]!='\0'; j++)
       {
          if(p[i][j]>= 'A' && p[i][j]<= 'Z')
              presult[0]++;
          else if(p[i][j]>= 'a' && p[i][j]<= 'z' )
              presult[1]++;
          else if(p[i][j]>= '0' && p[i][j]<= '9' )
              presult[2]++;
          else if(p[i][j] == ' ' )
              presult[3]++;
          else
              presult[4]++;
       }
    }
}
```

统计结果初始化为0

综合举例

多个字符串排序函数

```
void sort(int *presult, int n, char (*pname)[20])
{
    int i,j;
    int temp;
    char str[20];

    for(i=0;i<n-1;i++)
      for(j=i+1;j<n;j++)
        if(presult[i]<presult[j])
        {
          temp=presult[i];
          presult[i]=presult[j];
          presult[j]=temp;

          strcpy(str,pname[i]);
          strcpy(pname[i],pname[j]);
          strcpy(pname[j],str);
        }
    return;
}
```

选择法排序

多个字符串输出函数

```
void output(int *presult, int n, char (*pname)[20])
{
    int i;

    for(i=0;i<n;i++)
      printf("%s:%d\n",pname[i],presult[i]);
    return ;
}
```

7.6 综合举例

例7.17　信息录入系统，首先输入密码验证正确后，根据显示的编号与姓名，输入相应的个人信息：生日、年龄以及家庭住址，并能对个人信息进行修改和显示。

问题分析与设计：

模块划分：该问题可以分成四个功能块：密码验证、信息初始化输入、信息修改和信息显示。

数据结构：初始密码设计为字符串常量char *word=“hust111”；姓名为字符串常量设计为成字符指针数组；生日和家庭住址也为字符指针数组；年龄设计为一维整形数组。

7.6 综合举例

例7.17　信息录入系统，首先输入密码验证正确后，根据显示的编号与姓名，输入相应的个人信息：生日、年龄以及家庭住址，并能对个人信息进行修改和显示。

函数原型：

(1) int password(char *pword);

功能：验证输入的密码是否与初始密码相同

形参：字符指针用来传递初始密码字符串

(2) void init (char *pname[],char *pdate[], int page[], char *paddress[]);

功能：信息初始化输入

形参：用来传递个人信息：姓名、生日、年龄以及家庭住址

(3) void menu(char *pname[], char *pdate[], int page[], char *paddress[]);

功能：首先选择修改什么个人信息（生日、年龄以及家庭住址），随后根据编号选择修改哪个人的信息，最后进行信息的输入与更新

(4) void show(char *iname[], char *idate[], int iage[], char *iaddress[],int n);

void showall(char *iname[], char *idate[], int iage[], char *iaddress[]);

功能：显示某个人或全部的个人信息

7.6 综合举例

例7.17　信息录入系统，首先输入密码验证正确后，根据显示的编号与姓名，输入相应的个人信息：生日、年龄以及家庭住址，并能对个人信息进行修改和显示。

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
        int p=1;
        char *word="hust111";
        char *name[3]={"zhang san","li si","wang wu"};
        char *date[3];
        int age[3];
        char *address[3];
        p=password(word);
        if (!p)
        {
            init(name, date, age, address);
            menu(name, date, age, addre
        }
        else
        {
            printf("The password is wrong!");
            return;
        }
}
```

如果密码正确则显示操作界面，否则直接退出

信息初始化录入后进入信息修改操作菜单

综合举例

密码验证函数

```
int password(char *pword)
{
  char *iword;
  printf("Please input the password:\n");
  gets(iword);
  return (strcmp(pword,iword));
}
```

信息显示函数

```
void show(char **iname, char **idate,
          int *iage, char **iaddress,int n)
{
  printf("number:\t%d\n",n);
  printf("name:\t%s\n",iname[n-1]);
  printf("date:\t%s\n",idate[n-1]);
  printf("age:\t%d\n",iage[n-1]);
  printf("address:%s\n",iaddress[n-1]);
}
```

信息初始化输入函数

```
void init (char **pname,char **pdate, int *page,
           char **paddress)
{
    int i;
    char str[20];
    for(i=0;i<3;i++)
    {
      printf("number:%d\nname:%s\n",i+1,pname[i]);
      printf("Please input the date:\n");
      gets(pdate[i]);
      printf("Please input the age:\n");
      gets(str);
      page[i]=atoi(str);
      printf("Please input the address:\n");
      gets(paddress[i]);
    }
    printf("Initialization complete!\n\n");
    showall (pname, pdate, page, paddress);
}
```

将用gets得到的字符串转换成整型进行存储

综合举例

信息修改函数

```
void menu(char **pname, char **pdate, int *page,
          char **paddress)
{
    char a, str[20];
    int i;
    do
    {
        printf("****      menu     ****\n");
        printf("**** 1.modify date    ****\n");
        printf("**** 2.modify age     ****\n");
        printf("**** 3.modify address ****\n");
        printf("**** 4.show          ****\n");
        printf("**** 5.exit          ****\n");
        gets(str);
            a=str[0];
    }while(a<'1'||a>'5');
```

1~3分别用来修改个人信息中的生日、年龄以及家庭住址；4用来显示个人完整信息；5表示退出系统；其他输入无效。

```
    switch(a-'0')
    {
        case 1:
        printf("Which one do you want to modify?\n\n");
        printf("number:");
        gets(str);
        i=atoi(str);
        show (pname, pdate, page, paddress,i);
        printf("Please input the date:\n");
        gets(pdate[i-1]);
        break;
         …
        case 4:
        showall (pname, pdate, page, paddress);
        getch();
        break;
        case 5:
        return;
    }
        menu(pname, pdate, page, paddress);
}
```

修改操作完成后回到菜单选择界面，直至选择exit选项

7.6 综合举例

例7.18　关于通讯录的字符串应用程序。通讯录中的信息包括有个人的编号、姓名和电话号码，所需要实现的功能有通讯录信息的添加、显示和修改。

问题分析与设计：

模块划分：该通讯录管理程序包括3个功能模块，分别是添加个人信息、显示所有成员信息、修改个人信息。

数据结构：该程序要处理的数据是通讯录中个人的信息，包括编号、姓名、电话号码。由于个人信息为字符串的形式，其地址存放在char *data[100][3]的二维字符指针数组中，最多可包含100人的信息，采用动态内存分配的方式进行处理。

7.6 综合举例

例7.18　关于通讯录的字符串应用程序。通讯录中的信息包括有个人的编号、姓名和电话号码，所需要实现的功能有通讯录信息的添加、显示和修改。

函数原型：

(1) int input(unsigned int (*linepstr)[3],int *n);

功能：添加个人信息。

形参：unsigned int (*linepstr)[3]数组指针通过访问存储个人信息地址的字符数组来传递个人信息；int *n用来传递已存个人信息数量。

(2) void print(unsigned int (*pri)[3],int n);

功能：显示所有个人信息。

形参： unsigned int (*pri)[3]数组指针功能同上，int n指已存个人信息数量。

(3) void modify(unsigned int (*mod)[3],int n);

功能：修改个人信息。

形参：unsigned int (*mod)[3]功能同上,int n指已存个人信息数量。

综合举例

例7.18　关于通讯录的字符串应用程序。通讯录中的信息包括有个人的编号、姓名和电话号码，所需要实现的功能有通讯录信息的添加、显示和修改。

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    char *data[100][3];
    int i,j,n=0,s=0,flag=0;
    for(i=0;i<100;i++)
    {
        for(j=0;j<3;j++)
        data[i][j]=NULL;
    }
    for(;;)
    {
        switch(menu_select())
        {
            case 1:
            printf("\n\t Add records to phone book\n");
            s+=input((unsigned int (*)[3])data,&n);
            break;
            case 2:
            print((unsigned int (*)[3])data,s);
            break;
            case 3:
            modify((unsigned int (*)[3])data,s);
            break;
            case 4:
            flag=1;
            break;
        }
        if(flag==1)
            break;
    }
    for (i=0;i<s;i++)
        for(j=0;j<3;j++)
            if(data[i][j]!=NULL)
                free(data[i][j]);
}
```

调用菜单函数，1表示通讯录信息的添加，2表示通讯录信息的显示，3表示修改通讯录中的信息，4表示退出该系统

指针数据初始化为NULL

综合举例

通讯录信息添加函数

```
int input(unsigned int (*linepstr)[3],int *n)
{
        char sign='0',*p,str[20];
        int a=*n ,len;
        while(sign!='n'&&sign!='N')
        {
            printf("\t number:\n");
            gets(str);
            len=strlen(str);
            if((p=(char *)malloc(len))==NULL)
            {
                printf("not enough memory to allocte buffer.\n");break;
            }
            strcpy(p,str);
            *(*(linepstr+*n)+0)=(unsigned int) p;
            ...
              printf("\n\t continue to add(Y/N)\n");
              gets(str);
               sign=str[0];
              (*n)++;
          }
          return (*n-a);
}
```

动态内存分配适当的内存空间

将地址变为无符号整数存到指针*linepstr指向的数组中，并作为指针数组char *data[100][3]中的一个元素参与后来的应用

综合举例

通讯录信息修改函数

```
void modify(unsigned int (*mod)[3],int n)
{
        char str[20];
        int m=-1,i;
        printf("\t Please input the number that
                you want to modify:\n");
        gets(str);
        for(i=0;i<n;i++)
         {
                if(!strcmp(str,(char *)mod[i][0]))
                    m=i;
                    break;
                }
        }
```

遍寻通讯录，找到相应的编号进行修改

```
if(m!=-1)
    {
            printf("\t number\n");
            gets((char*)mod[m][0]);
            printf("\t name\n");
            gets((char*)mod[m][1]);
            printf("\t phone\n");
            gets((char*)mod[m][2]);
    }
    else
    {
            printf("\t Can't find the number,please
                    input again!\n");
            modify((unsigned int (*)[3])mod,n);
    }
}
```

本章小结

字符串是C语言的难点和重要内容，需要在深刻理解概念的基础上，熟练掌握，本章需重点掌握的内容有：

- 掌握字符与字符串的基本概念，学会利用字符数组和字符指针去定义和处理字符串
- 掌握与字符串相关的常用库函数，会使程序实现更简练
- 掌握C语言中单个字符串处理方法和应用
- 掌握C语言中多个字符串处理方法和应用
- 了解带参数的main函数及其应用

END