

南京大学 计算机科学与技术系

Department of Computer Science & Technology, NJU

Chapter 7

字符串



刘奇志

字符数组及其应用

字符数组与字符串

- 字符数组的定义和初始化
- 字符数组的输入/输出
- 字符数组作为函数的参数
- 用字符指针操纵字符数组
- 字符串的处理
- 字符型指针数组

基于字符数组的信息检索程序

字符数组的定义

- ❁ C语言中只有字符串常量，没有定义字符串变量的基本类型，需要通过构造基类型为char的数组来存储和处理字符串

```
char str[10];
```

- 用char、[]和10构造了一个一维字符数组类型，并用该类型定义了一个一维字符数组str
- 系统会为该字符数组分配10个内存单元 ($10 * \text{sizeof}(\text{char})$)，以存储10个字符型元素

字符数组的初始化:

```
char str[10] = {'H', 'e', 'l', 'l', 'o', ' ', 'N', 'J', 'U', '\0'};
```

- 该初始化方式最好在最后加一个字符串结束标志 '`\0`' (转义符, 对应的ASCII码是0)。
- 部分初始化, 未初始化元素默认为0, 即 '`\0`'

```
char str[10] = "Hello NJU";
```

H	e	l	l	o		N	J	U	\0
---	---	---	---	---	--	---	---	---	----

- 该初始化方式不必加 '`\0`', 因为C语言中的字符串常量最后会自动加 '`\0`', 并赋给定义的数组。
- 如果 `const char str[10] = "Hello NJU";`
//这样数组元素的值不可更改
- 定义的字符数组长度应该保证足以存储该结束标志, 这是一个约定俗成的做法, 可以方便字符串的相关操作。

🌈 'A' 与 "A" 的区别:

➡ `sizeof('A')` 1

01000001

➡ `sizeof("A")` 2

01000001	00000000
----------	----------

字符数组的特殊性

- 访问各个元素时，不需要根据数组长度设计循环流程的终点
- 依靠结束符

字符的输入

输入单个字符

```
char ch;
```

```
ch = getchar( );    //可以转存输入的回车符
```

```
scanf ("%c", &ch);
```

```
cin >> ch;
```

字符串的输入

输入字符串，系统自动添加结束符

```
char str[10];
```

```
#include <string>
string s;
getline(cin, s);
```

`gets(str)` ; //不安全，回车符不会转存到str中，不过可以转存空格和水平制表符

`gets_s(str, 10)` ; //功能同上，最多可输入9个字符

```
cin.getline(str, 10);
cin.get(str, 10);
```

```
scanf("%s", str);
```

//不安全，空白符不会转存到str中

`scanf_s("%s", str, 10)` ; //功能同上，最多可输入9个字符

```
cin >> str;
```


字符与字符串的输出

```
printf("%c \n", ch);  
printf("%s \n", str);
```

<code>cout << ch << endl;</code>
<code>cout << str << endl;</code>

```
puts(str);
```

- 会多输出一个回车换行符

例1 统计输入的字符行中数字、空格及其他字符的个数

```
int nDigit=0, nSpace=0, nOther=0;
char ch;
printf("Enter string line\n");

while ((ch = getchar()) != '\n')
    if (ch >= '0' && ch <= '9')
        ++nDigit;
    else if (ch == ' ')
        ++nSpace;
    else
        ++nOther;

printf("%d, %d, %d \n", nDigit, nSpace, nOther);
```

❁ 例1 统计输入的字符行中数字、空格及其他字符的个数。

```
int nDigit=0, nSpace=0, nOther=0;
char str[80];
printf("Enter string line\n");
gets(str);
for (int i=0; str[i] != '\0'; i++)
    if (str[i] >= '0' && str[i] <= '9')
        ++nDigit;
    else if (str[i] == ' ')
        ++nSpace;
    else
        ++nOther;

printf("%d, %d, %d \n", nDigit, nSpace, nOther);
```

字符数组作为函数的参数

- ❁ 当一维数组作为函数的参数时，通常需把一维数组的名称以及数组元素的个数传给被调函数，以便被调函数确定数组处理的终点。
- ❁ 对于一维字符数组，则不需要传递数组元素的个数，因为可以凭借'\0'来确定其处理终点。

例2 字符串的大小写转换。

...

```
void to_upper(char s[ ]);  
  
int main( )  
{  
    char str[10];  
    printf("Please input a string: \n");  
    scanf("%s", str);  
    to_upper(str);  
    printf("The new string is %s \n", str);  
}  
  
void to_upper(char s[ ])  
{  
    for(int i = 0; s[i] != '\0'; i++)  
    {  
        if(s[i] >= 'a' && s[i] <= 'z')  
            s[i] = s[i] - 'a' + 'A'; //小写转换成大写  
    }  
}
```

🌈 例3 数字字符串到整数的转换，比如“365”转换为 $((3*10)+6)*10+5$ 。

$$0*10 + 3 = 3$$

$$3*10 + 6 = 36$$

$$36*10 + 5 = 365$$

```
int str_to_int(char s[ ])
{
    if(s[0] == '\0') return 0;    //空字符串转换为0
    int i, n = 0;                  //n用于存储转换结果，初始化为0
    for(i = 0; s[i] != '\0'; i++) //循环处理各位数字
        n = n * 10 + (s[i] - '0');
    return n;
}
```

//通过数字字符与字符'0'的ASCII码差值计算数字字符对应的数值

//即'3'-'0' == 3, '6'-'0' == 6, '5'-'0' == 5

用指针操纵字符数组*——字符指针

```
char str[10];
```

```
char *pstr = str;    //相当于char *pstr = &str[0];
```

- **pstr先存储str[0]的地址，不妨设为0x00002000 (简作2000)，然后，pstr的值可以变化为2001，2002，2003，2004，2005，2006，2007，2008，2009，于是可以通过pstr来操纵字符数组str的各个元素。**

例4 字符数组里字符串的反转，比如“Program”转换为“margorP”。

...

```
int main( )
```

```
{ char str[10] = "Program";
```

```
char *ph = str, *pt = ph; //ph指向第一个字符
```

```
for( ; (*pt) != '\0'; pt++); //循环结束时，pt指向最后一个字符
```

```
for(pt--; ph < pt; ph++, pt--) //ph、pt相向移动
```

```
{ char temp = *ph;
```

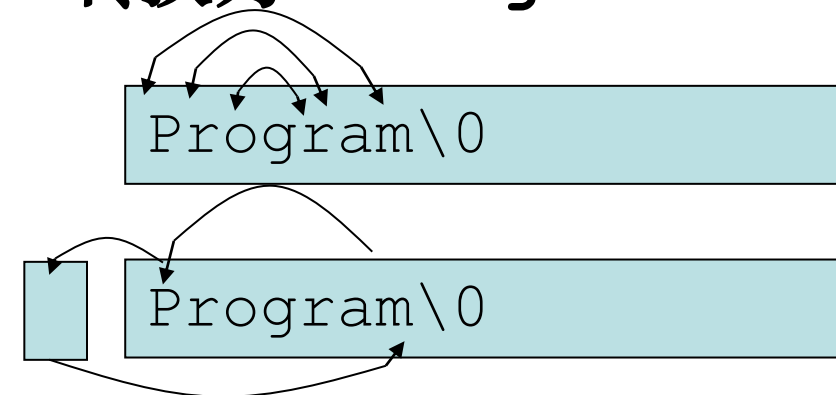
```
    *ph = *pt;
```

```
    *pt = temp;
```

```
}
```

```
printf("The reverse string is %s \n", str);
```

```
return 0; }
```



❁ 输出字符指针，可以输出它所指向的字符串。比如，

➤ `char *pstr = "Hello";`

➤ `printf("%s \n", pstr);` `//显示Hello, cout << pstr << endl;`

❁ 也可以显示地址值，比如：

➤ `printf("The string is %d. \n", pstr);`

`//按十进制整数形式显示一个地址值`

➤ `printf("The string is %x. \n", pstr);`

`//按16进制整数形式显示一个地址值`

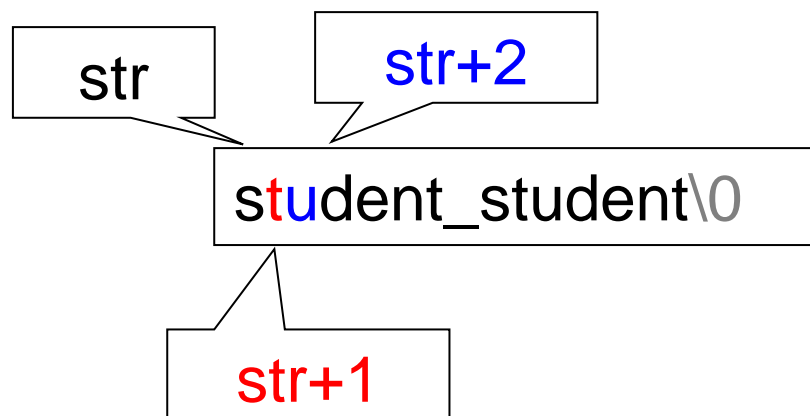
`0x22ff50`

`cout << (void *)str;` `//输出地址`

输出字符型数组名，也可以输出字符串

```
char str[] = "student_student";  
printf("%s \n", str); //输出student_student, cout << str << endl;
```

```
cout << str+1 << endl; //printf("%s \n", str+1);  
cout << str+2 << endl; //printf("%s \n", str+2);
```



```
C:\WINDOWS\system32\cmd.exe  
student_student  
tudent_student  
udent_student  
请按任意键继续. . .
```

也可以显示地址值

➡ ...

- 如果输出没有结束符的字符串，则输出时会显示若干乱码（未使用过的内存初始信号往往是汉字“烫”的机内码）。

➤ 比如，

```
char str[] = {'H', 'e', 'l', 'l', 'o'};  
printf("%s", str); // cout << str; 可能会显示Hello烫烫...
```

- 内存中总有一些单元里的信号是0（'\0'的ASCII码），于是输出时，会将str数组里的字符以及其后的若干乱码全部输出，直至遇到0为止。程序员应防范乱码问题。

```
char ch = 'c';  
char *pc = &ch;  
cout << pc << endl;  
//printf("%s \n", pc);  
//输出以c开头的乱码字符串
```

● 字符指针也常用作函数的参数，以提高字符型数据的传递效率。

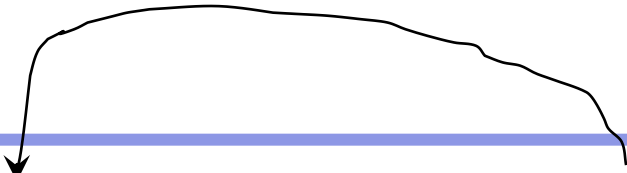
● 例5 字符串拷贝函数。

const

指针移动法

```
void myStrcpy(char *t, char *s)
{
    while( (*s) != '\0' )
    {
        *t = *s;
        t++;
        s++;
    } //逐个字符复制
    *t = *s; //复制 '\0'
} //该函数还利用了函数的副作用“返回”函数处理的结果

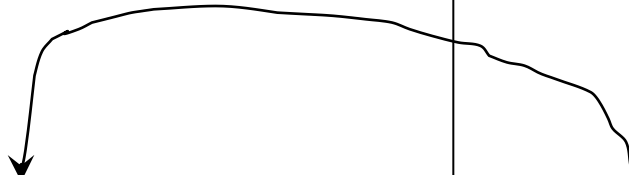
int main()
{
    char str[20];
    myStrcpy(str, "C Language");
    puts(str);
    return 0; }
```



```
char *strcpy(char *dst, const char *src)
{
    int i;
    for (i = 0; src[i] != '\0'; ++i)
        dst[i] = src[i];
    dst[i] = '\0';
    return dst;
}
```

下标法

约定从右往左，以便调用



```
char *strncpy(char *dst, const char *src, int n)
{
    int i;
    for (i = 0; i < n && src[i] != '\0'; ++i)
        dst[i] = src[i];
    dst[i] = '\0';
    return dst;
}
```

用字符数组处理字符串 vs. 用字符指针处理字符串

用字符串常量初始化字符数组

➤ `char str[] = "Hello";`

//str在栈区占6个字节空间，存储该字符串常量的副本

用字符串常量初始化字符指针

➤ `char *pstr = "Hello";`

//pstr在栈区占4个字节空间，存储该字符串常量的首地址

直接将字符串常量赋给字符指针（这其实是将字符串常量的首地址存储在字符指针中）

➤ `char *pstr;`

➤ `pstr = "Hello";`

//注意比较对于字符数组str，不可以“`str = "Hello";`”

然后可以通过数组或指针访问字符串常量

```
printf("%c \n", str[0]);           //访问第一个字符
printf("%c \n", *pstr);           //访问第一个字符
printf("%c \n", pstr[0]);         //访问第一个字符
printf("%c \n", str[2]);          //访问第三个字符
printf("%c \n", *(pstr+2));       //访问第三个字符
printf("%c \n", pstr[2]);         //访问第三个字符
```

```
str[0] = 'h';    //数组元素的值可修改
(*pstr) = 'h';  //? 标准未规定
pstr[0] = 'h';  //? 标准未规定
```

```
const char *pstr = "Hello";
           //确保字符串常量中的字符不被修改, C++
```

区别之要点:

```
char str[10] = "Hello NJU"; //可以初始化  
str = "Hello NJU"; // 不可以赋值  
scanf("%s", str); //可以输入  
printf("The string is: %s \n", str); //可输出
```

```
const char *pstr = "Hello NJU"; //可以初始化  
pstr = "Hello NJU"; // 可以赋值  
scanf("%s", pstr); //不可以输入  
printf("The string is: %s \n", pstr); //可输出
```


常用字符串处理库函数

- 字符串处理是非数值计算任务中的常见环节，在程序设计中占有重要地位。
- C语言标准库中提供了一些常用字符串处理函数，它们通常默认'\0'为字符串的结束标志，这些函数的说明信息位于头文件string.h中。

(1) 计算字符串的长度

```
unsigned int strlen(const char * s);
```

➤ 计算字符串s中有效字符的个数，不包括'\0'

➤ 比如，

```
char str[] = "Hello";
```

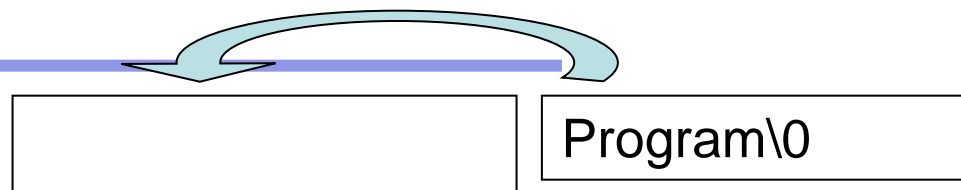
```
printf("%d \n", strlen(str));
```

```
printf("%d \n", strlen(str+2));
```

5

3

(2) 字符串复制



```
char *strcpy(char * s1, const char * s2);
```

- 把字符串s2复制到s1所指向的内存空间中。调用该库函数须保证s1所指向的内存空间足以存储s2
- 在s2长度未知的情况下，可以用库函数strncpy把字符串s2中的前n个字符复制到s1所指向的内存空间中

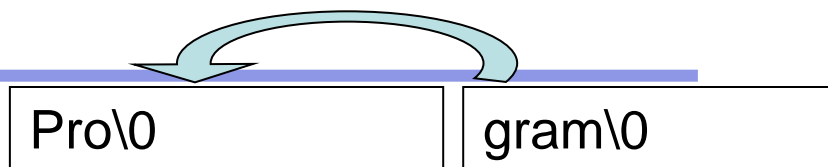
```
char *strncpy(char * s1, const char * s2, int n);
```

- 比如，

```
char *str = strcpy(str, "nju", 2);
```

//库函数strcpy和strncpy的返回值均为s1的内存首地址

(3) 字符串拼接



```
char *strcat(char * s1, const char * s2);
```

```
char *strncat(char * s1, const char * s2, int n);
```

- 这两个库函数的功能是把字符串s2追加到s1所指向的内存中字符串的后面，s1所指向的内存中原字符串的结束符被覆盖，新拼接的长字符串结尾有'\0'，其他特征与字符串复制库函数类似。

(4) 字符串比较

```
int strcmp(const char *s1, const char *s2);
```

```
int strncmp(const char *s1, const char *s2, int n);
```

- 这两个库函数的功能是比较两个字符串的大小，即两个字符串在字典中的前后关系，越靠后的越大，比如，study比student大，worker比work大。
- 这两个库函数的返回值为字符串s1与s2中对应位置第一个不同字符的ASCII码差值，比如，strcmp("study", "student")的结果为20，其他特征与字符串复制库函数类似。

- 负数表示s2大

student\0

study\0

- 如果s1与s2中没有不同字符，则返回值为0，表示两个字符串相等(两个字符串相等的充要条件是长度相等且各个对应位置上的字符都相等)

stu\0

stu\0

- 正数表示s1大

study\0

student\0

- 比如，

if(strncmp(str, "nju", 2) == 0)说明 str 前两个字符为 nj

例6 判断用户输入密码的正确性 (提供三次机会)

```
...
#include <string>
#define PASSW "X1BYU4KR"
int main()
{ char cmark='n', word[25];
  int i;
  for(i = 0; i < 3; i++)
  { printf("Please enter the password:");
    scanf("%s", word);
    if(strcmp(word, PASSW) == 0)
    { printf("Password is correct.");
      cmark = 'y';
      break;
    }
  }
```

```
    else
        printf("Password is incorrect!");
    }
    if(cmark == 'n')
        return -1;
    else
        ...
    return 0; }
```

新版标准及支持的开发环境 (eg. VS) 对一些字符串处理库函数进行了更新:

➤ `errno_t strcpy_s(char *_Dst, rsize_t _SizeInBytes, const char *_Src)`

【strcpy_s(目标字符数组或指针, 结果字节数 (含'\0'), 源串)】

➤ `errno_t strncpy_s(char *_Dst, rsize_t _SizeInBytes, const char *_Src, rsize_t _MaxCount)`

【strncpy_s(目标字符数组或指针, 结果字节数 (含'\0'), 源串, 前几个字符)】

➤ `errno_t strcat_s(char *_Dst, rsize_t _SizeInBytes, const char *_Src)` (参数含义同strcpy_s)

➤ `errno_t strncat_s(char *_Dst, rsize_t _SizeInBytes, const char *_Src, rsize_t _MaxCount)` (参数含义同strncpy_s)

`gets_s(str, 5)` //最多可输入4个字符

`scanf_s("%s", str, 5)` //最多可输入4个字符

拷贝：覆盖

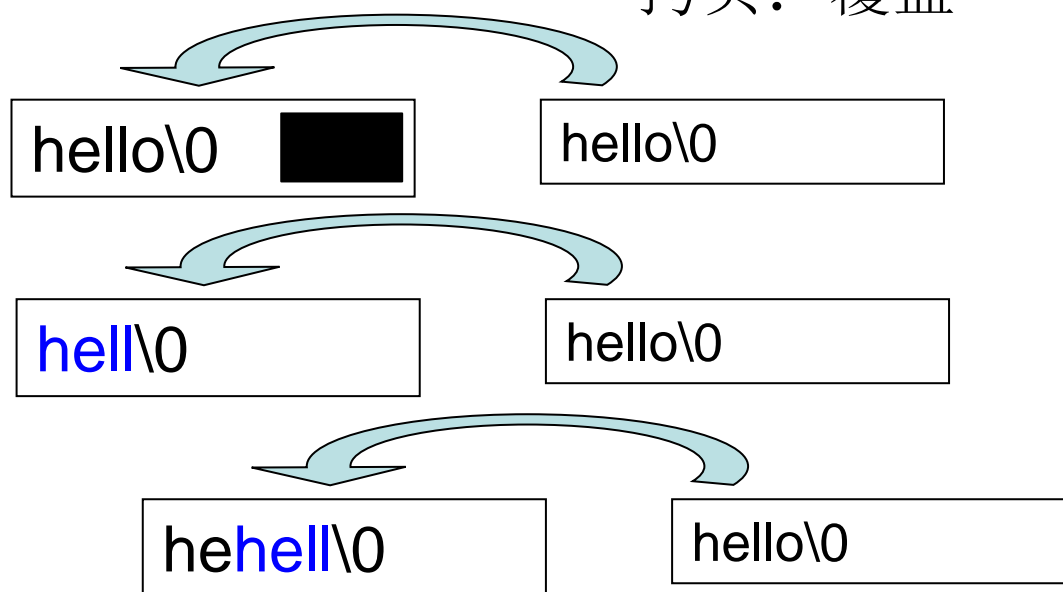
$\geq \text{strlen}(\text{"hello"}) + 1$

❶ `strcpy_s(dst, 6, "hello")`

$\geq 4 + 1$

❷ `strncpy_s(dst, 5, "hello", 4)`

❸ `strncpy_s(dst+2, 7, "hello", 4)`



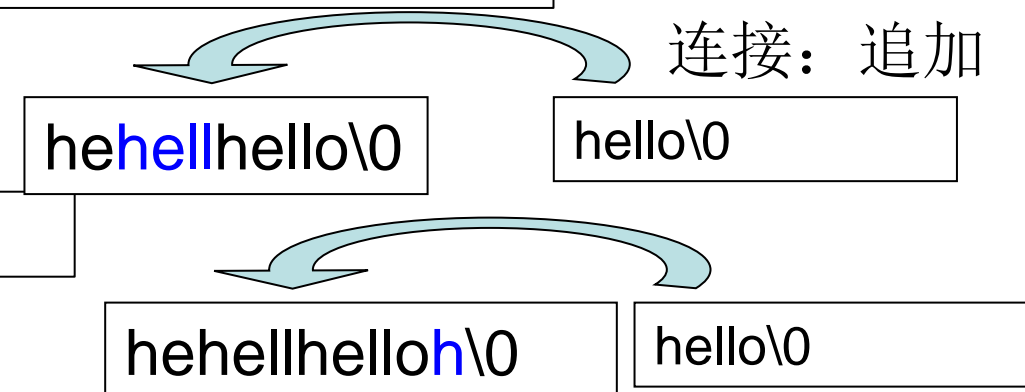
$\geq \text{strlen}(\text{dst}) + \text{strlen}(\text{"hello"}) + 1$

❹ `strcat_s(dst, 12, "hello")`

$\geq \text{strlen}(\text{dst}) + 1 + 1$

❺ `strncat_s(dst, 14, "hello", 1)`

连接：追加



二维字符型数组

🌈 可以用来表示多个字符串。比如，

```
char b[4][5] = { {'Z', 'h', 'a', 'o', '\0'},  
                  {'Q', 'i', 'a', 'n', '\0'},  
                  {'S', 'u', 'n', '\0'},  
                  {'L', 'i', '\0'} };
```

➡ 也可以写成：

```
char b[4][5] = { "Zhao", "Qian", "Sun", "Li" };
```

用指针操纵二维数组-1

- 对于二维数组，可以通过不同级别的指针变量来操纵。注意，二维数组名表示第一行的地址。

➤ 比如，

```
char b[5][10];
```

```
char *p;
```

```
p = &b[0][0];    //或 “p = b[0];”
```

//一级指针变量可以存储数组b某一元素的地址

- 然后通过**指针移动法**指定某个元素地址，**再取值**指定该元素，比如 `p++`，`*p`;

```
char b[4][5] = {"Zhao", "Qian", "Sun", "Li"};
```

```
char *p = b[0];
```


```
for(int i=0; i < 20; i++)
```

```
{
```

```
    printf("%c", *p);
```

```
    p++;
```

```
} //输出20个字符
```



Z	h	a	o	\0
Q	i	a	n	\0
S	u	n	\0	
L	i	\0		

用指针操纵二维数组-2

➤ 或,

```
char (*q) [10];
```

```
q = &b[0]; //或 “q = b;”
```


//二级指针变量可以存储数组b某一行的地址

- 然后通过**下标法**指定某个元素，比如`q[i][j]`（相当于`b[i][j]`）；
- 也可以通过**指针移动法**指定某一行的地址，**再取值**指定某一行，比如`q++`，`*q`，或某一行的首元素，`**q`；
- 还可以通过**偏移量法**指定某个元素

```
char b[ ][5] = {"Zhao", "Qian", "Sun", "Li", "\0"};
```

```
char (*q)[5] = b;
```

```
while(strcmp(*q, "\0"))  
{  
    printf("%s\n", *q);  
    q++;  
} //输出四个字符串
```



Z	h	a	o	\0
Q	i	a	n	\0
S	u	n	\0	
L	i	\0		
\0				

指针数组

- 如果数组的每一个元素都是一个指针类型的数据，则该数组叫做指针数组。

➔ 比如，

```
int i = 0, j = 1, k = 2;
```

```
int *ap[3] = {&i, &j, &k};
```

//ap这个指针数组的长度是3，各个元素的类型是int *

- 指针数组一般用于多个字符串的处理

字符型指针数组

- 对于每一个元素都是一个字符型指针的一维数组，可以用来表示多个字符串。比如，

```
char b[4][5] = { {'Z', 'h', 'a', 'o', '\0'},  
                 {'Q', 'i', 'a', 'n', '\0'},  
                 {'S', 'u', 'n', '\0'},  
                 {'L', 'i', '\0'} };
```

➤ 也可以写成：

```
char b[4][5] = { "Zhao", "Qian", "Sun", "Li" };
```

```
char *bp[4] = {&b[0][0], &b[1][0], &b[2][0], &b[3][0]};
```

或

```
char *bp[4] = {b[0], b[1], b[2], b[3]};
```

```
char *bp[4] = {b[0], b[1], b[2], b[3]};
```

- 这里，字符型指针数组bp是4个地址数据群体，每个地址指向一个字符串常量，也可以直接初始化成：

```
char *bp[4] = {"Zhao", "Qian", "Sun", "Li"};
```

//注意与数组指针char(*q)[5]的区别

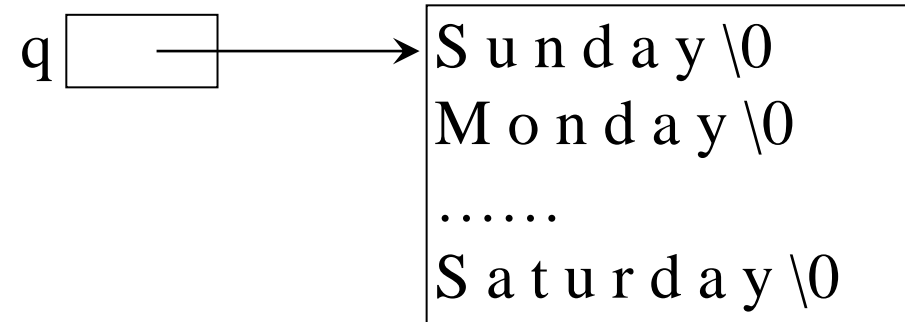
-
- 用**字符型指针数组**表示多个字符串不需要事先知道字符串的最大长度，比用二维字符型数组（或数组指针）表示多个字符串更为方便。

二维数组 vs. 数组的指针 vs. 指针数组

二维字符型数组

```
char weekday[7][10] = {"Sunday", "Monday", "...", "Saturday"};
```

```
cout << weekday[6] << endl;
```



数组的指针

```
char (*q)[10] = weekday;
```

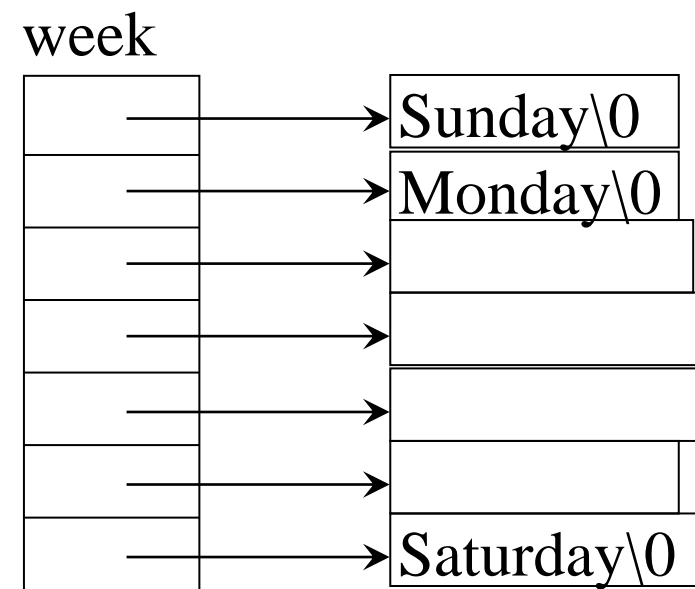
```
q += 6;
```

```
cout << *q << endl;
```

字符指针数组

```
char *week[7] = {"Sunday", "Monday", "...", "Saturday"};
```

```
cout << week[6] << endl;
```



例7 多个字符串的排序程序（将百家姓的拼音按字典顺序重排）。

...

```
#include <string>

int main( )
{ char *temp, *name[ ] = {"Zhao", "Qian", "Sun", "Li"};
  int n=4, i, min;
  for(i = 0; i < n - 1; i++)
  { min = i;
    for(int j = i + 1; j < n; j++)
      if(strcmp(name[min], name[j]) > 0) min = j;
    if(min != i)
    { temp = name[i];
      name[i] = name[min];
      name[min] = temp;
    }
  }

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

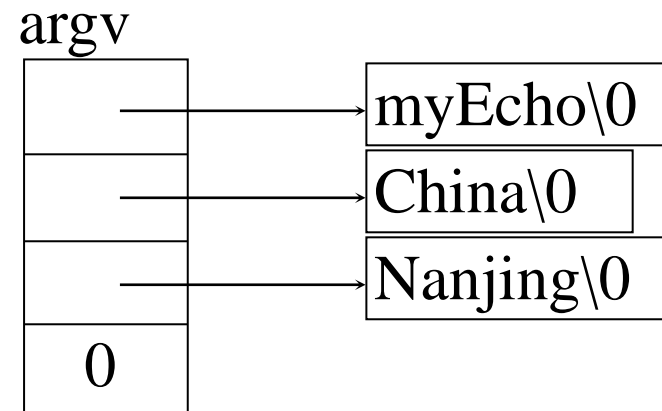
带形参的 main 函数*

```
#include <stdio.h>

int main(int argc, char *argv[ ])
{
    while(argc > 1)
    {
        printf("%s \n", argv[argc-1]);
        --argc;
    } //从第二个元素开始分行逆序输出所有元素
    return 0;
}
```

程序执行结果为:

Nanjing
China



假设该代码存在源文件**myEcho.c**中，
执行环境按如下命令执行该程序：

c:\>myEcho China Nanjing

即命令中包括三个字符串，
形参**argc**会自动获得字符串的个数**3**，
形参**argv**是一个字符型指针数组，
每个元素会获得一个字符串：

argv[0] 获得 “myEcho”

argv[1] 获得 “China”

argv[2] 获得 “Nanjing”

**

不要求掌握

行列都是动态的“二维数组”（第二行第一个元素与第一行最后一个元素未必连续）：

```
int n, m;
```

```
cin >> n;
```

```
char **array = new char *[n];    //先new一个动态指针型数组（一维）
```

```
for(int i=0; i < n; ++i)
```

```
{  cin >> m;
```

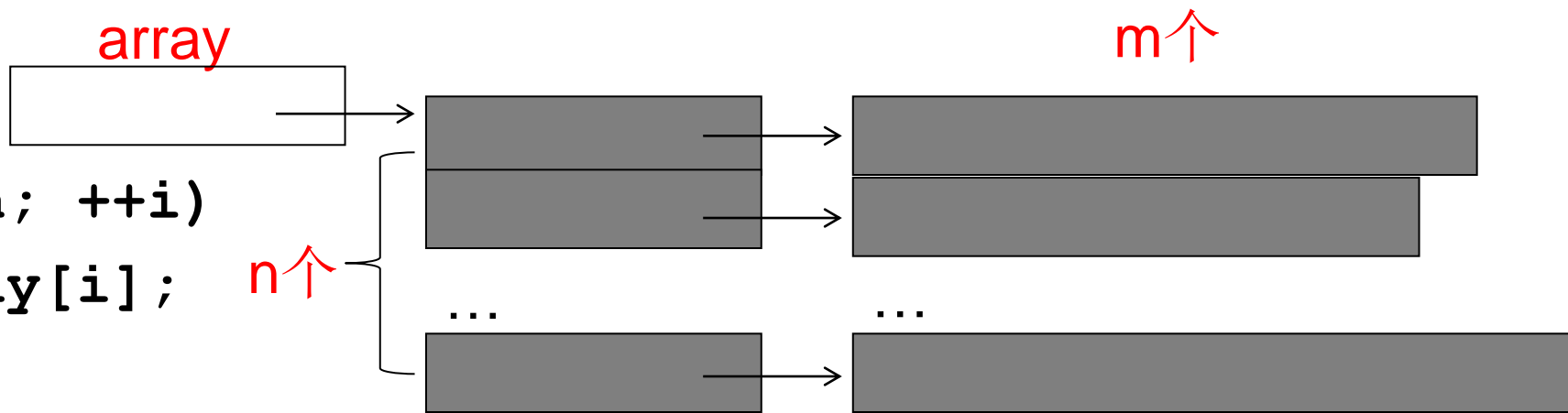
```
    array[i]= new char[m];    //再分别new数组的每一个元素指向的数组
```

```
}
```

```
for(int i=0; i < n; ++i)
```

```
    delete []array[i];
```

```
delete []array;
```



基于字符数组的信息检索程序

- 信息检索问题是一种常见的非数值计算问题。本节以基于字符数组的字符查找为例，介绍简单的信息检索方法。常用的数据查找算法有顺序查找、折半查找等。

查找

- 有一字符数组`str`，其中的字符已按从小到大的顺序排列好，现在从键盘上输入一个字符`key`，用折半法在`str`中查找此字符。找到，输出它在数组中的位置；没找到，给出相应提示。
- 分析：折半法(二分法)查找，要求待查数据排列有序——`str`已符合要求。先确定待查数据范围(区间)，然后逐步缩小范围，直到搜索完为止。
- 设计：
 - 计算待查区间的中间元素下标`pmid`；
 - 将`key`与`str[pmid]`比较，得到三种结果并分别处理：

$\text{str}[\text{pmid}] < \text{key}$	待查记录在 <code>str</code> 的右半部，重新计算待查区间
$\text{str}[\text{pmid}] = \text{key}$	查找成功，结束
$\text{str}[\text{pmid}] > \text{key}$	待查记录在 <code>str</code> 的左半部，重新计算待查区间

整个区域查找完毕，没查到

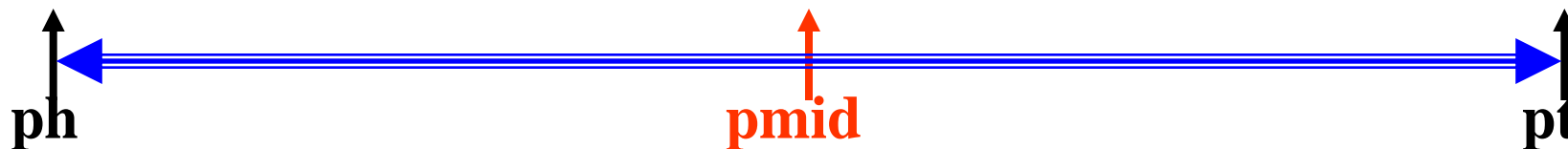
设 $\text{key} = 'd'$ ，则查找过程如下：

0 1 2 3 4 5 6 7 8 9 10

str

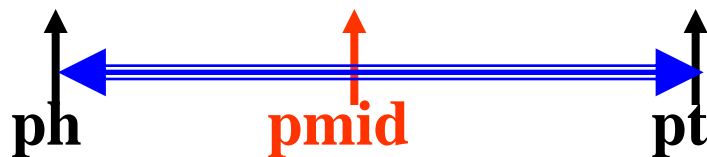
a	b	d	f	g	h	j	k	l	s	t
---	---	---	---	---	---	---	---	---	---	---

第1趟



$\text{key} < \text{str}[\text{pmid}]$ ，key应在左半部分

第2趟



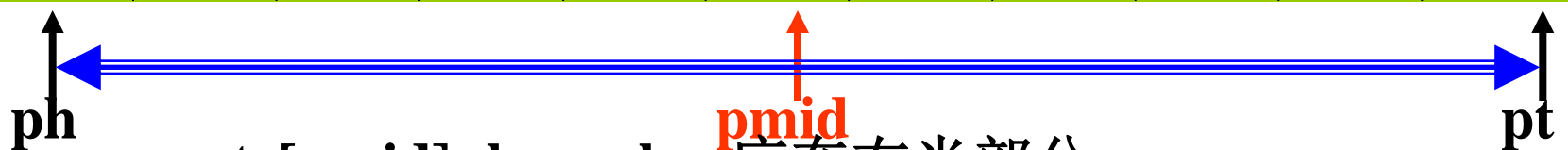
$\text{str}[\text{pmid}] == \text{key}$

找到

设key='d'，则查找过程如下：

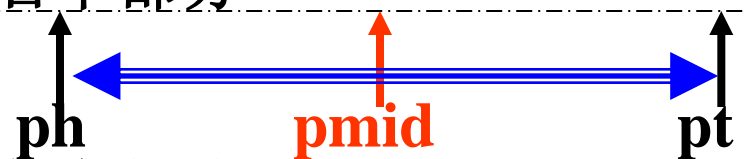
	0	1	2	3	4	5	6	7	8	9	10
str	a	b	d	f	g	h	j	k	l	s	t

第1趟



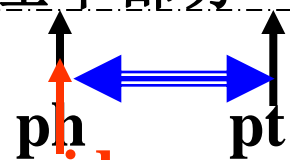
$\text{str}[\text{pmid}] < \text{key}$, key应在右半部分

第2趟



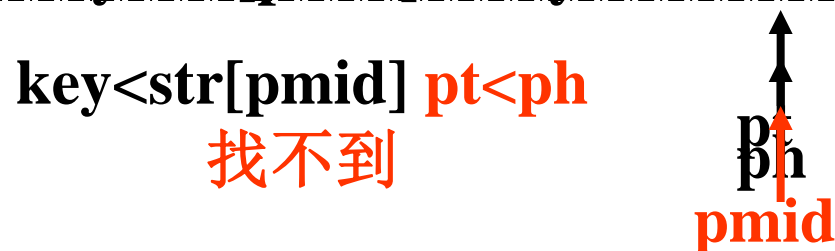
$\text{key} < \text{str}[\text{pmid}]$, key应在左半部分

第3趟



$\text{key} < \text{str}[\text{pmid}]$, key应在左半部分

第4趟



```

int main()
{
    char key, str[ ] = "abcdefghijklmnopqrst";
    scanf("%c", &key);
    int flag = BiSearch(str, key, 0, strlen(str)-1);
    if(flag == -1)    printf("\n not found \n");
    else    printf("%d \n", flag);
    return 0;
}

int BiSearch(char x[ ], char k, int ph, int pt)
{
    int pmid;
    while(ph <= pt)
    {
        pmid = (ph+pt)/2;
        if (k == x[pmid])    break;
        else if (k > x[pmid])    ph = pmid+1;
        else    pt = pmid-1;
    }
    if(ph > pt) pmid = -1;
    return pmid;
}

```

写成递归函数

```
int BiSearch(char x[ ], char k, int ph, int pt)
{
    if(ph <= pt)
    {
        int pmid = (ph+pt)/2;
        if(k == x[pmid])
            return pmid;
        else if(k > x[pmid])
            return BiSearch(x, k, pmid+1, pt);
        else if(k < x[pmid])
            return BiSearch(x, k, ph, pmid-1);
    }
    else
        return -1;
}
```

算法分析

顺序查找

- 最好情况：比较1次
- 最坏情况：比较N次
- 平均情况： $(1+2+\dots+N)/N = (N+1)/2$ 次

二分法查找

- 最好情况：比较1次
- 最坏情况：比较 $\log_2(N+1)$ 次
- 平均情况： $\log_2(N+1) - 1$ 次

小结

🌈 字符串：

- 用字符数组或字符指针来表示和处理
- 与其他类型数组和指针相比，字符型数组和指针具有特殊性（结束符）

🌈 要求：

- 掌握字符数组和指针的定义、初始化和操作方法
- 能够自行实现常用字符串处理库函数
 - 一个程序代码量 \approx 100行
- 继续保持良好的编程习惯

Thanks!

