

---

# step further

字符串

专题

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与封装（模块设计方法、函数）

表达与转换（基本操作、数据类型）

提高：

构造与访问（数组、指针、结构）

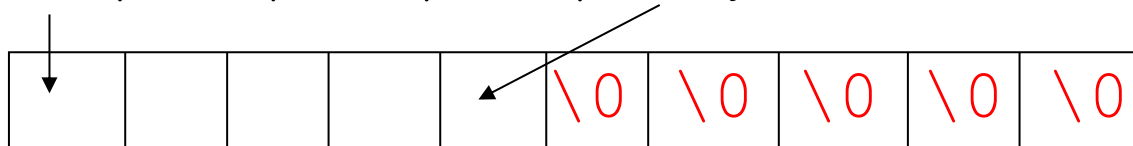
归纳与推广（程序设计的本质）

# 字符数组的定义与初始化

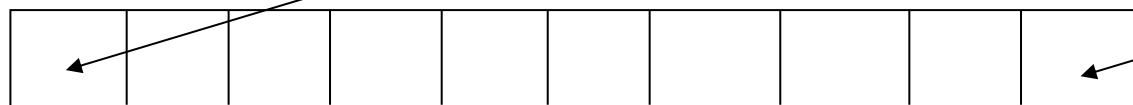
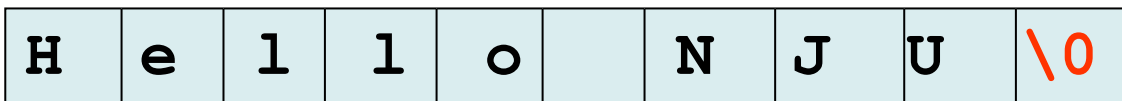
```
char str[10] = {'H','e','l','l','o',' ','N','J','U','\0'};
```



```
char str[10] = {'H','e','l','l','o'};
```



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



# 字符的输入

## ■ 方法一:

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

```
char ch;
```

```
ch = getchar( ); //可以转存输入的回车符、空格和水平制表符
```

## ■ 方法二:

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

```
char ch;
```

```
scanf("%c", &ch); //可以转存输入的回车符、空格或水平制表符
```

新标准更新为**scanf\_s**("%c", &ch, 1);

相当于C++中的

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

# 字符的输出

---

## ■ 方法一：

```
#include <stdio.h>
...
printf("%c", ch);
```

相当于C++中的

```
#include <iostream>
using namespace std;
...
cout << ch << endl;
```

## ■ 方法二：

```
#include <stdio.h>
...
putchar(ch);
```

# 字符数组的输入

□ 以下方法输入字符串，系统自动在末尾添加结束符

## ■ 方法一：

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

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

```
scanf("%s", str); //空白符不会转存到str中
```

相当于C++中的

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

新标准更新为scanf\_s("%s", str, 10); //最多可输入9个字符

## ■ 方法二：

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

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

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

相当于C++中的

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

新标准更新为gets\_s(str, 10); //最多可输入9个字符

相当于C++中的

```
#include <iostream>
using namespace std;
char str[10];
cin.getline(str, 10);
```

# 字符数组的输入（续）

□ 以下方法输入字符串，系统自动在末尾添加结束符

## ■ 方法三：

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

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

```
int i = 0;
```

```
while( (str[i] = getchar()) != '\n')  
    ++i;          //可以转存空格和水平制表符
```

相当于C++中的

```
#include <string>
```

```
string s;
```

```
getline(cin, s);
```

```
//混合输入：  
//int n;  
//char str[10];  
//  
//scanf("%d", &n);  
//getchar();      //吸收输入n之后的回车符  
//int i = 0;  
//while ( (str[i] = getchar()) != '\n')  
//    ++i;  
//  
//或  
//cin >> n;  
//getchar();      //吸收输入n之后的回车符  
//cin.getline(str, 10);
```

# 字符数组的输出

## ■ 方法一：

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

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

```
...
```

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

相当于C++中的

```
#include <iostream>
```

```
using namespace std;
```

```
cout << str << endl;
```

## ■ 方法二：

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

```
...
```

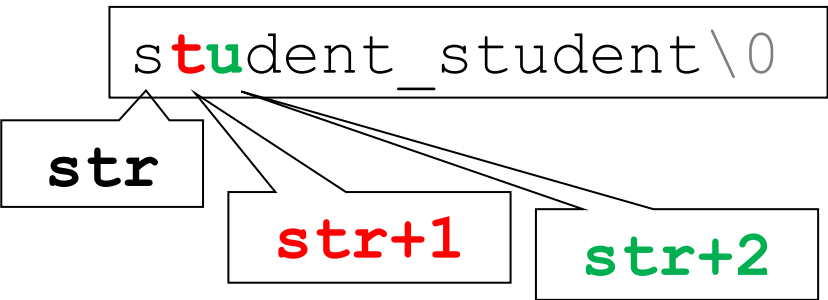
```
puts(str);    //会多输出一个回车换行符
```

□ 输出时，注意起始地址

```
char str[] = "student_student";  
printf("%s \n", str);
```

```
cout << str << endl;  
cout << str+1 << endl;  
cout << str+2 << endl;
```

```
printf("%s \n", str+1);  
printf("%s \n", str+2);
```



student\_student  
tudent\_student  
udent\_student

□ 也可以用%d或%x显示地址值

```
printf("%d. \n", str); // 按十进制整数形式显示  
printf("%x. \n", str); // 按十六进制整数形式显示
```

```
相当于 cout << (void *)str << endl;
```

4751360  
488000  
0x00488000



□ 如果没有结束符，则可能会显示若干乱码（未使用过的内存初始信号往往是“烫”、“屯”等汉字的机内码）。

■ 例如，

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

□ 编程时应防范乱码问题。

□ 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。

# 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用函数，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7.1 字符串拷贝

.....

```
int main()
```

```
{
```

```
    char str1[20], str2[20] = "Programming";
```

```
    MyStrCpy(str1, str2);
```

```
    puts(str1);
```

```
    return 0;
```

```
}
```

```
void MyStrCpy(char t[], const char s[])
```

```
{
```

```
    int i = 0;
```

```
    while( s[i] != '\0' )
```

```
    {        t[i] = s[i];
```

```
        ++i;
```

```
    }
```

```
    t[i] = s[i]; //复制 '\0'
```

```
} //该函数还利用了函数的副作用“返回”结果
```

## 例7.2 字符查找 (在一串字符中查找特定的字符)

□ 信息检索问题是一种常见的非数值计算问题。常用的算法有顺序查找、折半查找等。

□ 折半法(二分法)：假定字符数组`str`中的字符已按从小到大的顺序排列好，先确定待查区间，然后逐步缩小范围，直到搜索完为止。

- 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：

$\text{str}[\text{pmid}]$	$\left\{ \begin{array}{l} < \text{key} \\ = \text{key} \\ > \text{key} \end{array} \right.$	待查记录在 <code>str</code> 的右半部，重新计算待查区间
		查找成功，结束
		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，结束

设key='d'，则查找过程如下：

0 1 2 3 4 5 6 7 8 9 10

**str**

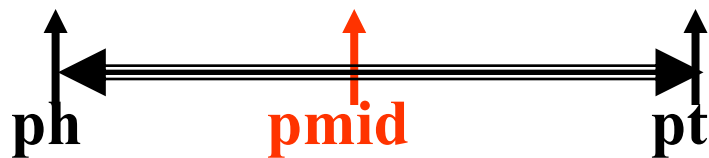
a	b	d	f	g	h	j	k	l	s	t
---	---	---	---	---	---	---	---	---	---	---

# 第1趟



**key<str[mid], key应在左半部分**

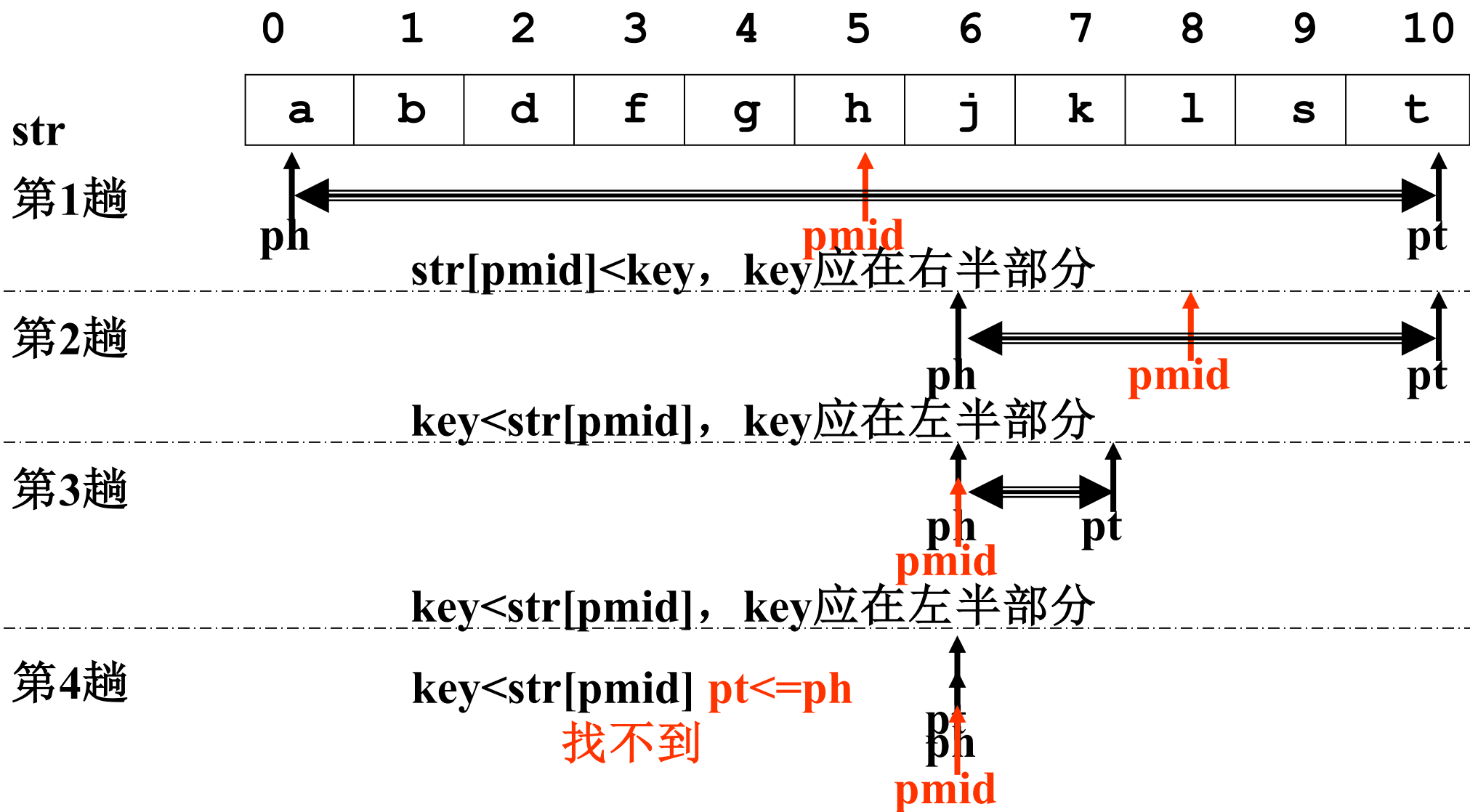
## 第2趟



**str[pmid]==key**

# 找到

设 $\text{key} = 'i'$ ，则查找过程如下：



.....

```
int main()
```

```
{
```

```
    char key, str[ ] = "abcdefghijklmnopqrst";
```

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

```
    int flag = BiSearch(key, str, 0, strlen(str)-1);
```

```
    printf("%d \n", flag);
```

```
    return 0;
```

```
}
```

也可以在被调函数里计算

```
int BiSearch(char k, char x[], int ph, int pt)
```

```
{
```

```
    while(ph <= pt)
```

```
    {        int pmid = (ph+pt)/2;
```

```
        if (k == x[pmid])    return pmid;
```

```
        else if (k > x[pmid])    ph = pmid+1;
```

```
        else    pt = pmid-1;
```

```
    }
```

```
    return -1;
```

```
}
```

# 写成递归函数

```
int BiSearch(char k, char x[], 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(k, x, pmid+1, pt);
        else
            return BiSearch(k, x, ph, pmid-1);
    }
    else
        return -1;
}
```



# 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。

# 字符指针的输出

- 通常输出的是所指向的字符串（从起始地址开始，到结束符前一个字符为止）

```
const char *pstr = "Hello";
```

```
printf("%s \n", pstr);
```

pstr



Hello\0

```
cout << pstr << endl;
```

Hello

- 也可以用%d或%x显示地址值

```
printf("%x \n", pstr); // 按十六进制整数形式显示
```

```
cout << (void *)pstr << endl;
```

- 输出指向单个字符的指针时。。。

```
char ch = 'c';
```

```
char *pc = &ch;
```

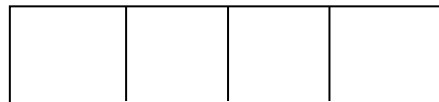
```
printf("%s \n", pc); // cout << pc << endl; 可能会输出 c烫烫烫...
```

# 字符指针 vs. 字符数组

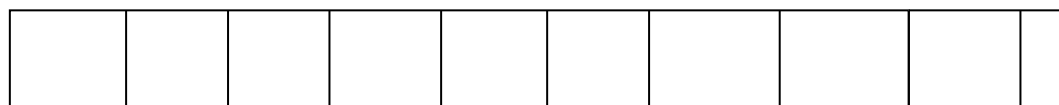
## 区别之要点:

### ■ 存储空间

`char *pstr;`



`char str[10];`



### ■ 初始化

`const char *pstr = "Hello NJU";` //用字符串常量首字符的**地址**初始化指针变量

`char str[10] = "Hello NJU";` //用字符串常量中的**字符**初始化字符数组的元素

### ■ 赋值

`pstr = "Hello NJU";` //可以将字符串常量首字符的**地址**赋给指针变量

`str = "Hello NJU";` //**×** 不可以赋值, 因为数组名代表第一个元素的地址, 是常量

### ■ 输入

`scanf("%s", str);` //可以输入

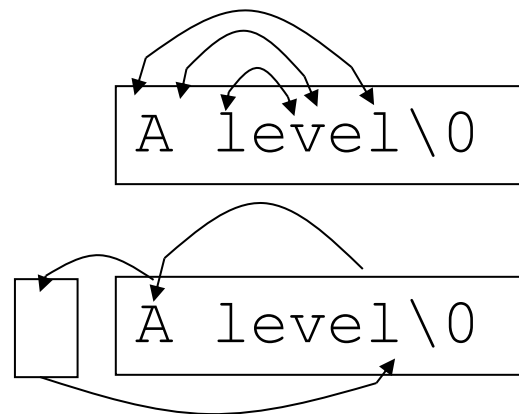
`scanf("%s", pstr);` //**?** 不一定能输入, 要看指针指向何处

<code>char *pstr = str;</code> <code>pstr = str;</code>	然后才可以对 pstr进行输入
--	--------------------

## 例7.3 字符串反转

用字符指针操控字符数组的元素

```
.....
int main()
{
    char str[ ] = "A level";
    Reverse(str);
    printf("%s \n", str);
    return 0;
}
```

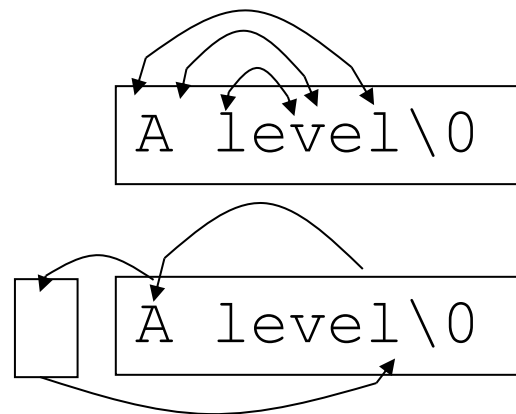


```
void Reverse(char x[])
{
    char *ph=x, *pt;
    for(pt=ph; (*pt) != '\0'; ++pt); //让pt指向结束符
    for(--pt; ph < pt; ++ph, --pt)
    {
        char temp = *ph;
        *ph = *pt;
        *pt = temp;
    } //ph、pt相向移动
}
```

## 例7.3 字符串反转

用字符指针操控字符数组的元素

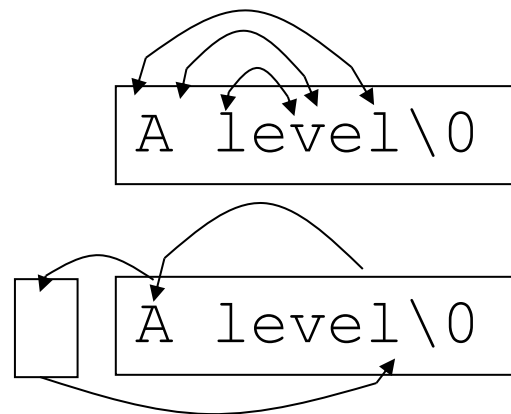
```
.....
int main()
{
    char str[ ] = "A level";
    Reverse(str);
    printf("%s \n", str);
    return 0;
}
```



```
void Reverse(char x[])
{
    char *ph=x, *pt;
    for(pt=ph; (*pt) != '\0'; ++pt) ; //让pt指向结束符
    for(--pt; ph < pt; ++ph, --pt)
    {
        char temp = *ph;
        *ph = *pt;
        *pt = temp;
    } //ph、pt相向移动
}
```

## 例7.3 字符串反转

```
.....
int main()
{
    char str[ ] = "A level";
    Reverse(str);
    printf("%s \n", str);
    return 0;
}
```



```
void Reverse(char x[])
{
    int i=0, j=0;
    while(x[j] != '\0')
        ++j;
    for(--j; i < j; ++i, --j)
    {
        char temp = x[i];
        x[i] = x[j];
        x[j] = temp;
    }
}
```

# 字符指针作为函数的参数

## □ 例7.1’ 字符串拷贝

指针移动法

```
void MyStrCpy(char *t, const char *s)  
{
```

```
    while( (*s) != '\0' )  
    {
```

```
        *t = *s;
```

```
        ++t;
```

```
        ++s;
```

```
    }
```

```
    *t = *s; //复制 '\0'
```

```
} //该函数还利用了函数的副作用 “
```

```
while( (*t++ = *s++) != '\0' );
```

```
void MyStrCpy(char t[], const char s[])  
{
```

```
    int i = 0;
```

```
    while( s[i] != '\0' )
```

```
    {        t[i] = s[i];
```

```
        ++i;
```

```
    }
```

```
    t[i] = s[i]; //复制 '\0'
```

```
} //该函数还利用了函数的副作用 “返回” 结果
```

# 早期库函数

```
char *strcpy(char *dst, const char *src)
```

```
{
```

```
    int i = 0;
```

```
    while (src[i] != '\0')
```

```
    {
```

```
        dst[i] = src[i];
```

```
        ++i;
```

```
    }
```

```
    dst[i] = '\0'; //复制 '\0'
```

```
    return dst;
```

} //该函数还利用了函数的副作用“返回”结果；又通过return返回了结果

下标法

```
strcpy(str, "NJU");
```

```
cout << str;
```

有返回值，所以可以：

```
cout << strcpy(str, "NJU");
```

```
strcpy(str, strcpy(str, "NJU"));
```

```
char *strncpy(char *dst, const char *src, int n)
```

```
{ //只拷贝前n个字符
```

```
    int i = 0;
```

```
    while (src[i] != '\0' && i < n)
```

```
        .....
```



# 常用字符串处理库函数 (**string.h**)

```
// #include <cstring>
```

## (1) 计算字符串的长度 (字符的个数)

```
unsigned int strlen(const char *s); //不包括'\0'
```

■ 例如,

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

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

5 3

## (2) 字符串复制

```
char *strcpy(char *dst, const char *src); //dst ← src
```

```
char *strncpy(char *dst, const char *src, int n);
```

■ 例如,

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

## (3) 字符串拼接

```
char *strcat(char *dst, const char *src);
```

```
char *strncat(char *dst, const char *src, int n);
```

# C11新版库函数

## ■ 字符串复制

slmax

**errno\_t** strcpy\_**s**(char \*\_Dst, **rsized\_t** \_SizeInBytes, const char \*\_Src)

**errno\_t** strncpy\_**s**(char \*\_Dst, **rsized\_t** \_SizeInBytes, const char \*\_Src, **rsized\_t** \_MaxCount)

```
strcpy_s(dstr, 6, "hello")  
strncpy_s(dstr, 5, "hello", 4)
```

## ■ 字符串拼接

**errno\_t** strcat\_**s**(char \*\_Dst, **rsized\_t** \_SizeInBytes, const char \*\_Src)

**errno\_t** strncat\_**s**(char \*\_Dst, **rsized\_t** \_SizeInBytes, const char \*\_Src, **rsized\_t** \_MaxCount)

```
strcat_s(dstr, 12, "hello")  
strncat_s(dstr, 14, "hello", 1)
```

## (4) 字符串比较

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

■ 例如,

`strcmp("student", "study")` 返回-1, 说明study大

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

■ 例如,

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

## (5) 模式匹配

```
char *strstr(char *haystack, char *needle);
```

■ 例如,

`strstr("woman", "man")` 返回 "man"

## (4) 字符串比较

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

■ 例如,

`strcmp("student", "study")` 返回-1, 说明study大

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

■ 例如,

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

## (5) 模式匹配

```
char *strstr(char *haystack, char *needle);
```

■ 例如,

`strstr("woman", "man")` 返回 "man"

## (6) 基于字符串的输入/输出

```
int sscanf(const char *buffer, const char *format, ...);
```

```
int sprintf(char *buffer, const char *format, ...);
```

■ 例如,

```
int x;
```

```
char dst[20];
```

```
scanf("%d", &x);
```

```
sprintf(dst, "%d", x); //int型变量x的值转换成字符串输出到dst中
```

```
char src[20] = "233hellonju";
```

```
char s[6];
```

```
sscanf(src, "%d%5s", &x, s); //从src中 输入233给x, 输入hello给s
```

## (7) 其他字符串处理相关库函数 (**stdlib.h**)

```
double atof(const char *nptr);    //把字符串转成double型
int atoi(const char *nptr);        //把字符串转成int型
long atol(const char *nptr);       //把字符串转成长int型
void *memset(void *s, int c, unsigned int n);
    //将字节数为n的一段内存单元全部置为变量c的值
void *memcpy(void *dst, void *src, unsigned int n);
    //将字节数为n的一段内存拷贝到dst, n不要大于dst的单元数
```

例如,

```
memset(str, '\0', sizeof(str)); //可将字符数组str的所有元素置为 \0
memcpy(dst, src, sizeof(dest)); //dst、src是非空类型数据的地址
```

# 几点说明：

- 关于字符串处理问题：如果是输入或输出，不管用什么方式（系统提供了两大类方式），只要是 oj 或 VS2019 支持的 就可以直接使用；如果不是输入或输出，需要自己写代码来实现。
- 所谓系统提供的两大类方式 指的是：一类是基于过程式程序设计方法所设计的函数库，由C语言标准规定，C++语言标准兼容，一般C/C++环境都支持；一类是基于对象式程序设计方法所设计的类库，由C++语言标准规定，仅C++环境支持。
- 如果使用标准输入/输出库函数，需先 `#include <stdio.h>`（C环境或C++环境）或 `#include <cstdio>`（C++环境）；如果使用输入输出类库，需先 `#include <iostream>` 及 `using namespace std;`
- 如果使用其他字符串处理库函数，需先 `#include <string.h>`（C环境或C++环境）或 `#include <cstring>`（C++环境）；如果使用其他字符串处理类库，需先 `#include <string>`（C++环境）。本课程一般不建议甚至不允许使用，具体看题目要求。

# 二维字符数组与字符指针

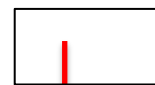
□ 可以用来处理多个字符串。比如，

```
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 \*p = b[0];** // 或 ... = &b[0][0];

```
void StrOut(char *p, int row, int col)  
{  
    char *ptemp = p;  
    for(int i=0; i < row; ++i)  
    {  
        while(*ptemp != '\0')  
        {  
            printf("%c", *ptemp);  
            ++ptemp;  
        }  
        printf("\n"); //输出一行  
        ptemp = (p += col);  
    }  
}
```



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



# 二维字符数组与字符数组的指针


□ 可以用来处理多个字符串。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 (\*q)[5] = b; // 或 ... = &b[0];**

```
void StrOut(char (*q)[5], int row)  
{  
    for(int i=0; i < row; ++i)  
    {  
        printf("%s\n", *q);  
        ++q;  
    } // 输出多个字符串  
}
```



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

# 二维字符数组与字符指针数组

□ 可以用来处理多个字符串。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], b[1], b[2], b[3]};  
// 或 ... = {&b[0][0], &b[1][0], &b[2][0], &b[3][0]};
```

```
char *x[] = bp;
```

```
void StrOut(char *x[], int row)  
{  
    for(int i=0; i < row; ++i)  
    {  
        printf("%s\n", x[i]);  
    } // 输出多个字符串  
}
```

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

- 
- 用**字符指针数组**处理多个字符串不需要事先知道字符串的最大长度，比用二维字符数组（或字符数组的指针）处理多个字符串更为方便.

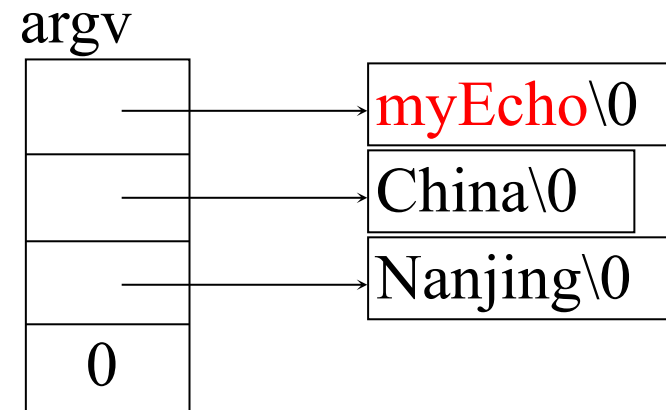
## 例7.4 多个字符串的排序程序(将若干姓氏的拼音按字典顺序重排)

```
...
#include <string.h>
const int N = 4;
int main( )
{
    const char *name[] = {"Zhao", "Qian", "Sun", "Li"};
    for(int i = 0; i < N - 1; ++i)
    {
        int min = i;
        for(int j = i + 1; j < N; ++j)
            if(strcmp(name[min], name[j]) > 0) min = j;
        if(min != i)
        {
            const char *temp = name[i];
            name[i] = name[min];
            name[min] = temp;
        }
    }
    for(int 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;
}
```



命令提示符

```
Microsoft Windows [版本 10.0.18363.1440]
(c) 2019 Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>cd..

C:\Users>cd..

C:\>myEcho China Nanjing
Nanjing
China

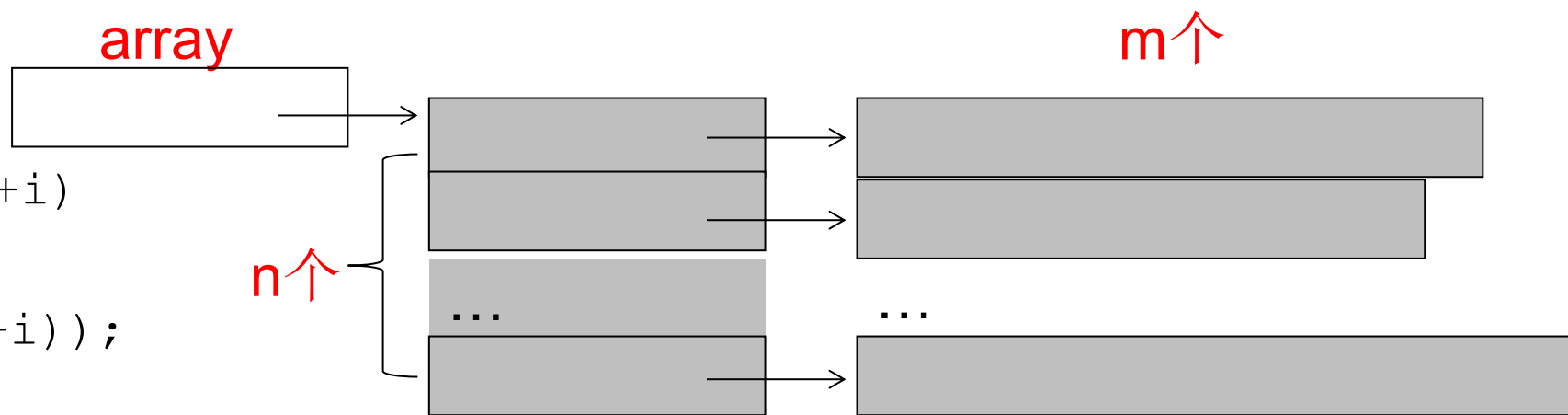
C:\>
```

# 行列都是动态的“二维字符数组”（每行第一个元素与前行最后一个元素未必连续）

```
int n, m;  
cin >> n;  
  
char **array = (char **)malloc(n * sizeof(char *)); //先创建动态字符指针数组  
for(int i=0; i < n; ++i)  
{  
    cin >> m;  
    if(array) //再分别创建动态一维字符数组  
        array[i] = (char *)malloc(m * sizeof(char));  
}
```

.....

```
for(int i=0; i < n; ++i)  
    if(array)  
        free(* (array+i));  
free(array);
```

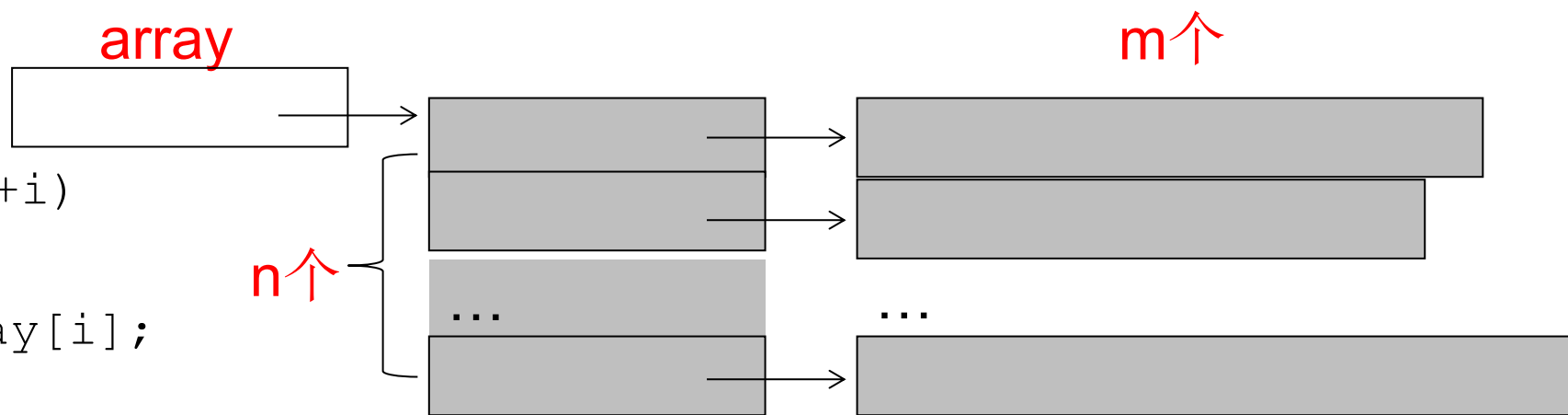


# 行列都是动态的“二维字符数组”（每行第一个元素与前行最后一个元素未必连续）

```
int n, m;  
cin >> n;  
char **array = new char *[n]; //先new一个动态字符指针数组（一维）  
for(int i=0; i < n; ++i)  
{  
    cin >> m;  
    if(array) //再分别new动态一维字符数组  
        array[i] = new char[m];  
}
```

.....

```
for(int i=0; i < n; ++i)  
    if(array)  
        delete []array[i];  
delete []array;
```



# 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