

规格严格 功夫到家



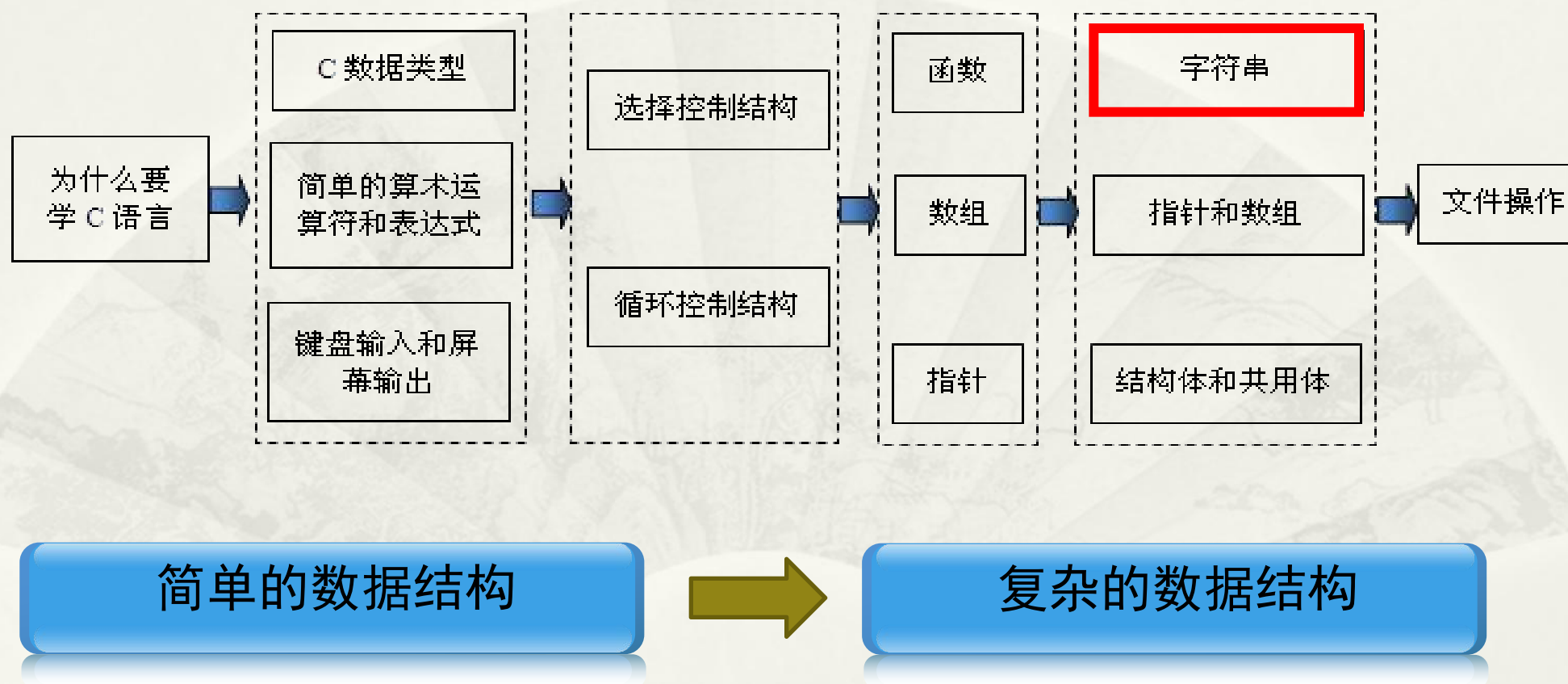
第10章 字符串

哈尔滨工业大学（深圳）
计算机科学与技术学院
刘洋

Liu.yang@hit.edu.cn

课件.版权：哈尔滨工业大学，苏小红，sxh@hit.edu.cn



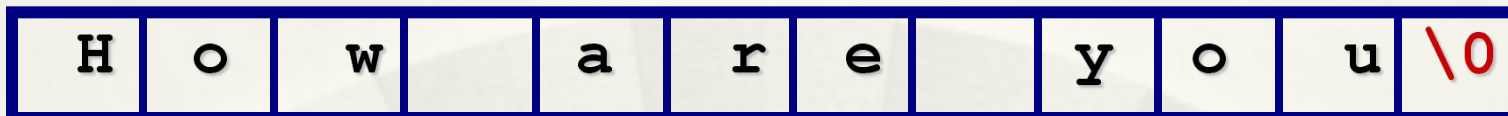


第10章 学习内容

- 字符串常量
- 字符数组和字符指针
- 字符串处理函数
- 向函数传递字符串
- 从函数返回字符串指针



10.1 字符串常量



一串以 '**\0**' 结尾的字符在C语言中被看作字符串

"How are you"

用双引号括起的一串字符是字符串常量，C语言自动为其添加 '**\0**' 结束符

10.2 字符串的存储

- C语言没有提供专门的字符串数据类型
- 使用字符数组和字符指针来处理字符串
- 字符数组

是字符数组，但不一定代表字符串

* 每个元素都是字符类型的数组

■ `char str[80];`

H	o	w		a	r	e		y	o	u	
H	o	w		a	r	e		y	o	u	\0

数组的最后一个元素必须是 '`\0`' 才表示字符串

10.2 字符串的存储

- 字符数组的初始化
- 用字符常量的初始化列表对数组初始化

```
* char str[6] = {'C','h','i','n','a','\0'};
```

- 用字符串常量直接对数组初始化

```
* char str[6] = {"China"};
```

```
* char str[6] = "China";
```

```
* char str[ ] = "China";
```


10.3 字符指针

- 还可以用字符指针来指向一个字符串
- 如果让字符指针指向一个字符串常量

```
char *pStr = "Hello China";
```

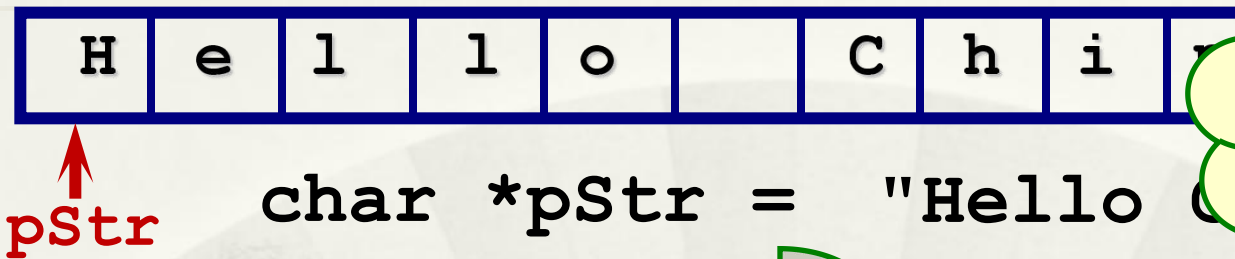


pStr

A red arrow points from the text 'pStr' to the first box of the memory array, which contains the character 'H'.

字符指针就是指向字符串首地址的指针

10.3 字符指针



```
char *pStr = "Hello China";
```

```
char *pStr;
```

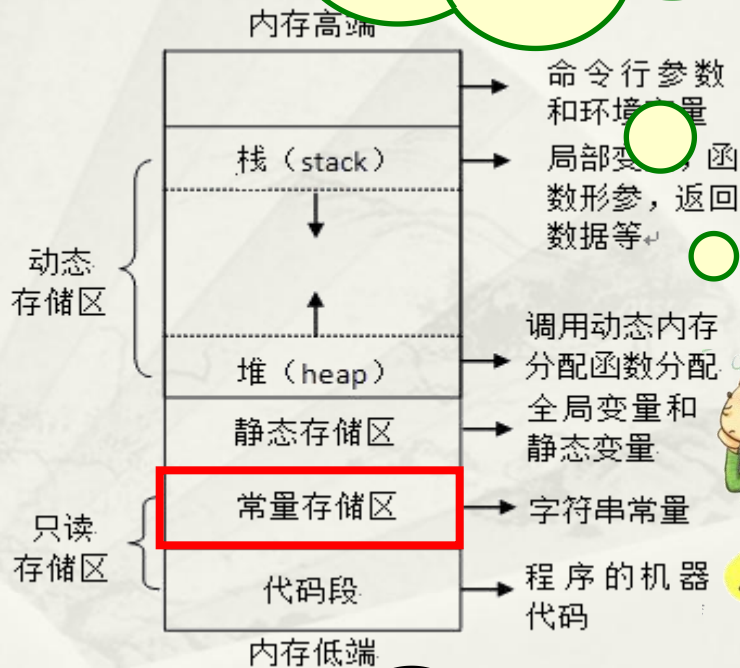
```
pStr = "Hello China";
```

pStr是一个指向常量存储区中的字符串的指针变量

字符串保存在只读的常量存储区

可修改**pStr**的值（指向），但不可以对它所指向的存储单元进行写操作

如果让字符指针指向一个**字符串数组**中的字符串呢？




***pStr** 'W' ;

10.3 字符指针

str H e l l o C h i n a \0

char str[] = "Hello China";

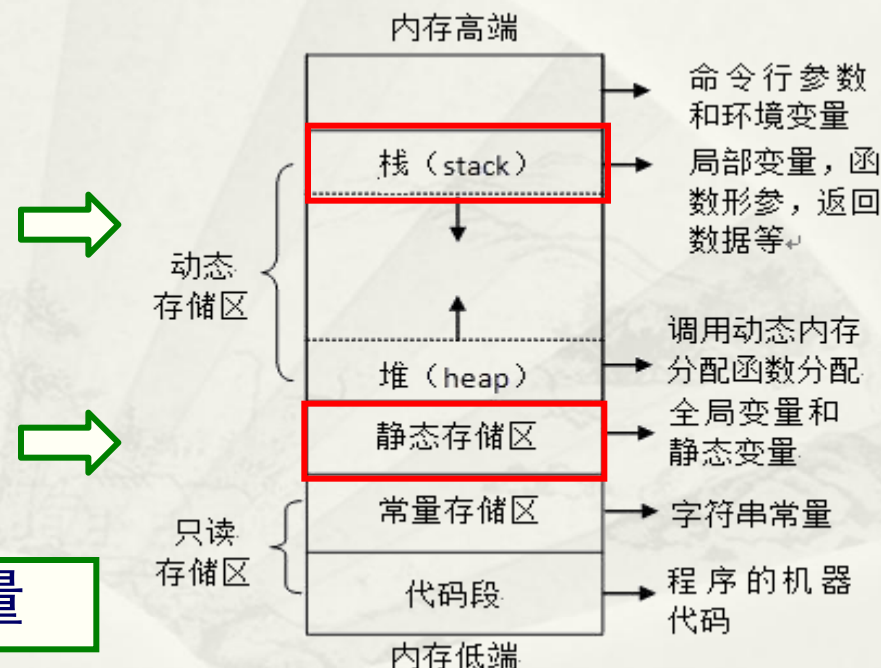
str  "Hello China";

函数内定义，保存在动态存储区

函数外定义，或定义为静态数组，字符串保存在静态存储区

数组名 **str** 的值不可修改，是地址常量

数组中存储的字符可以被修改



str[0] = 'W';

10.3 字符指针

str H e l l o C h i n a \0

pStr char str[] = "Hello China"

```
char *pStr;
```

```
pStr = str;
```

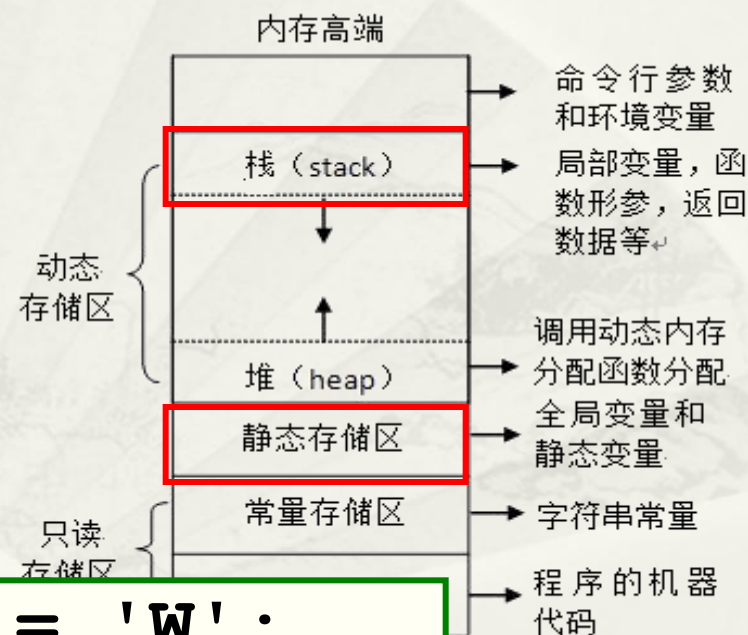
```
pStr = &str[0];
```

pStr的值（指向）可以被修改，它所指向的字符串也可以被修改

```
*pStr = 'W';
```

```
str[0] = 'W';
```

```
pStr[0] = 'W';
```



10.3 字符指针

- 正确使用和区分字符数组和字符指针
- 须牢记以下基本原则：
 - * 明确字符串被保存到了哪里
 - * 明确字符指针指向了哪里



10.4字符串的访问和输入/输出

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

按字符逐个输入/输出

```
for (i=0; str[i]!='\0'; i++)  
{  
    putchar(str[i]);  
}  
putchar('\n');
```

一般不用字符串长度控制，如 $i < 10$

10.4字符串的访问和输入/输出

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

按字符串整体输入/输出

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

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

不能输入带空格的字符串

```
gets(str);
```

```
puts(str);
```

可以输入带空格的字符串

10.4字符串的访问和输入/输出

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

```
int main()
```

```
{
```

```
    char c[80];
```

```
    printf("Input a string:");
```

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

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

```
    return 0;
```

```
}
```



用`%d`输入数字或`%s`输入字符串时，忽略空格、回车或制表符等空白字符，读到这些字符时，系统认为读入结束，因此不能输入带空格的字符串

10.4字符串的访问和输入/输出

```
#include <stdio.h>
int main()
{
    char c[80];

    printf("Input a string:");
    gets(c);
    printf("%s\n", c);

    return 0;
}
```



可输入带空格的字符串，因为空格和制表符都是字符串的一部分

此外，两个字符串输入函数对回车符的处理也不同

10.4字符串的访问和输入/输出

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

```
int main()
```

```
{
```

```
    char c[80];
```

```
    printf("Input a string:");
```

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

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

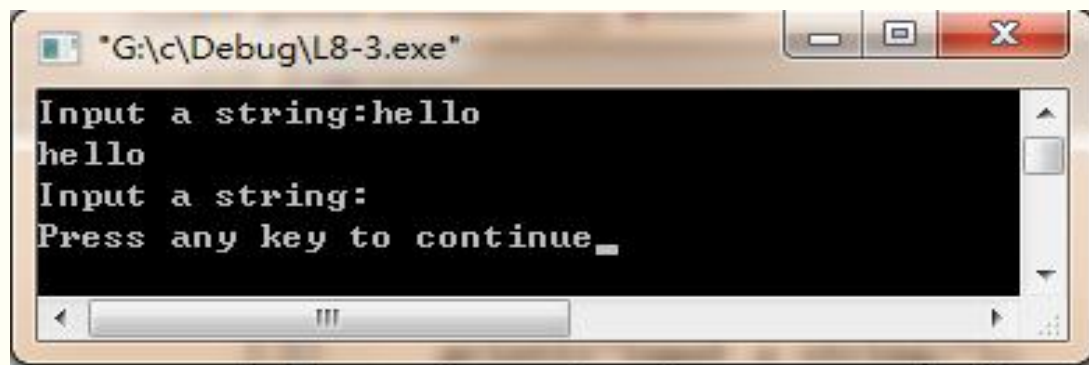
```
    printf("Input a string:");
```

```
    gets(c);
```

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

```
    return 0;
```

```
}
```



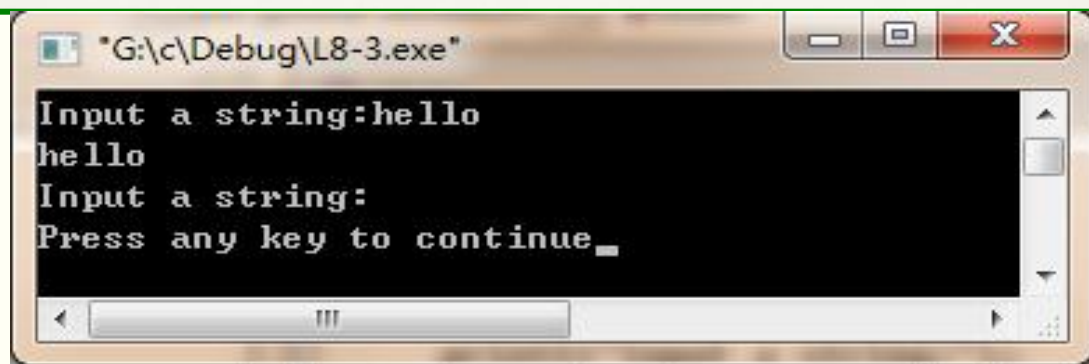
空格和制表符都是字符串的一部分，以回车作为字符串的终止符，同时将回车从缓冲区读走，但不作为字符串的一部分

10.4字符串的访问和输入/输出

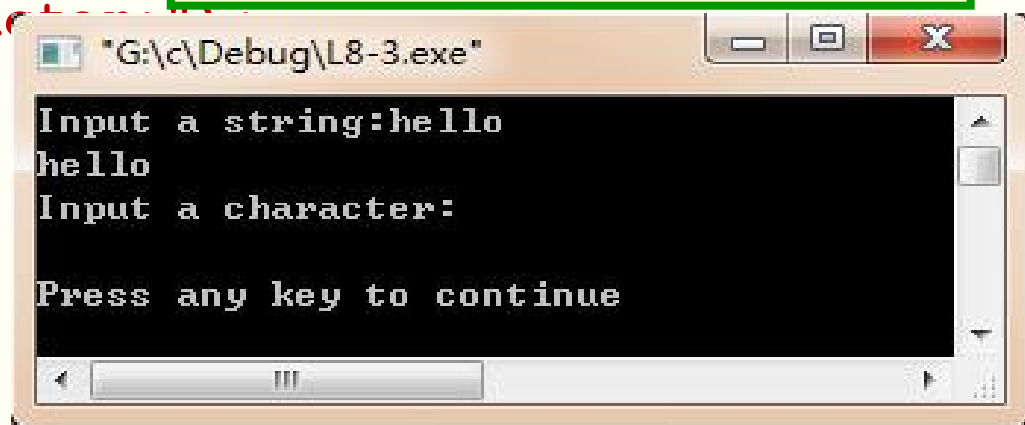
```
#include <stdio.h>
int main()
{
    char c[80], ch;

    printf("Input a string:");
    scanf("%s", c);
    printf("%s\n", c);
    printf("Input a character:");
    ch = getchar();
    printf("%c\n", ch);

    return 0;
}
```



scanf() 不读走回车，回车仍留在缓冲区中，回车被**getchar()** 读走

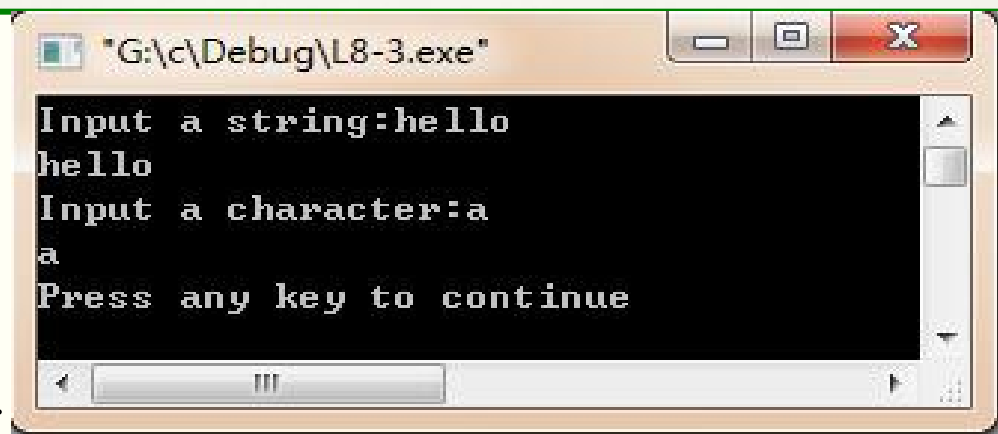


10.4字符串的访问和输入/输出

```
#include <stdio.h>
int main()
{
    char c[80], ch;

    printf("Input a string\n");
    gets(c);
    printf("%s\n", c);
    printf("Input a character:");
    ch = getchar();
    printf("%c\n", ch);

    return 0;
}
```




gets() 将回车从缓冲区读走，
所以getchar() 等待用户输入

10.4字符串的访问和输入/输出

```
#include <stdio.h>
int main()
{
    int n, ret;
    do{
        printf("Input n:");
        ret = scanf("%d", &n);
    }while (ret != 1);
    printf("n = %d\n", n);
    return 0;
}
```

假如格式不匹配，输入了非数字字符？



scanf() 按指定格式读取缓冲区中的数据，如果读取失败，则缓冲区中的非数字字符不会被读走，一直处于判断、读取、退出，判断、读取、退出，……（死机）

10.4字符串的访问和输入/输出

```
#include <stdio.h>
int main()
{
    int n, ret;

    do{
        printf("Input n:");
        ret = scanf("%d", &n);
        if (ret != 1)
        {
            while (getchar() != '\n');
        }
    }while (ret != 1);
    printf("n = %d\n", n);
    return 0;
}
```

键盘输入都被保存在输入缓冲区中，直到用户输入回车，输入函数才去读缓冲区中的数据

清空缓冲区

`while (getchar() != '\n');`

判断scanf() 返回成功读取的数据的个数，可以避免死机问题，但还需清空缓冲区

10.4字符串的访问和输入/输出

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

```
int main()
```

```
{
```

```
    char ch, c[80];
```

```
    printf("Input a string:");
```

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

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

```
    printf("Input a character:");
```

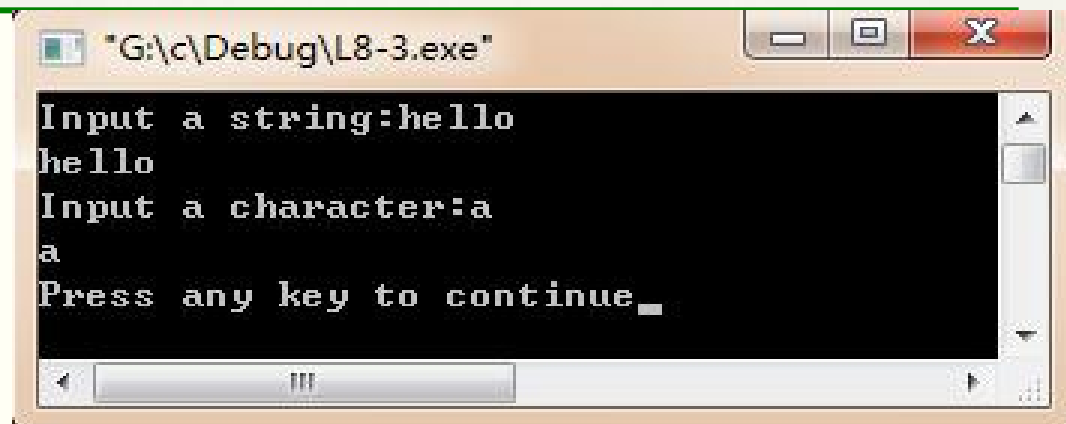
```
    scanf(" ");
```

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

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

```
    return 0;
```

```
}
```



清空缓冲区中的空白符

等价于 `scanf(" %c", &ch);`

10.4字符串的访问和输入/输出

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

```
int main()
```

```
{
```

```
    char ch, c[80];
```

```
    printf("Input a string:");
```

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

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

```
    printf("Input a character:");
```

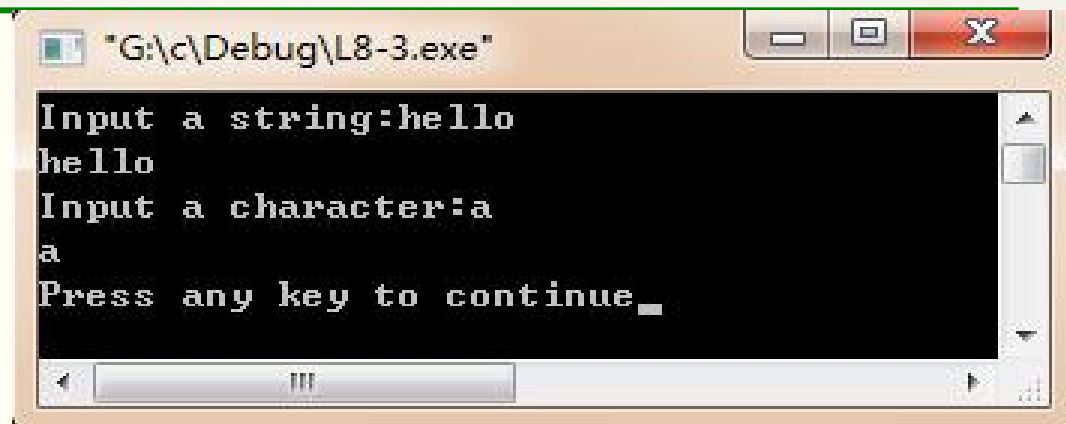
```
    scanf(" ");
```

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

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

```
    return 0;
```

```
}
```



在这里可以换成
getchar();

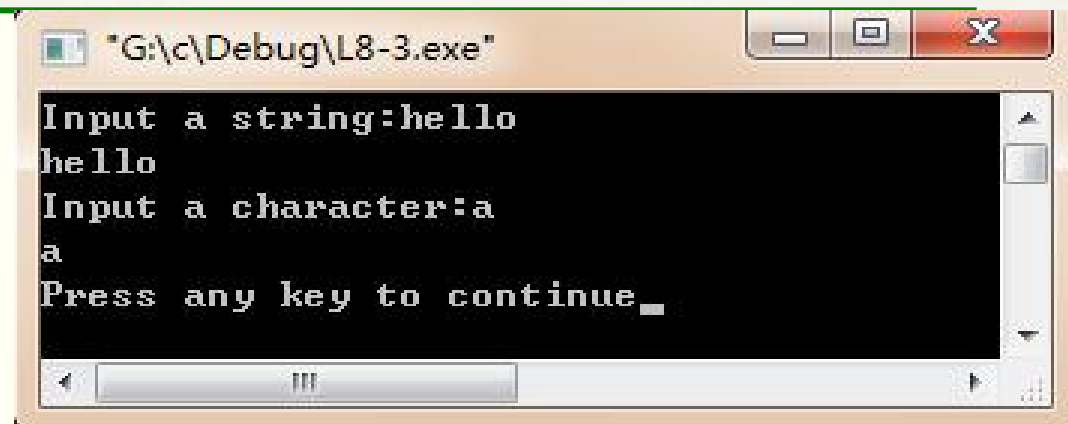
10.4字符串的访问和输入/输出

```
#include <stdio.h>
int main()
{
    char ch, c[80];

    printf("Input a string:");
    gets(c);
    printf("%s\n", c);

    printf("Input a character:");
    scanf("%c", &ch);
    printf("%c\n", ch);

    return 0;
}
```



无需再清空缓冲区

【例10.1】从键盘输入一个人名把它显示在屏幕上

```
1  #include <stdio.h>
2  #define N 12
3  int  main()
4  {
5      char  name[N];
6      printf("Enter your name(maximum 12 characters):");
7      scanf("%s", name);
8      printf("Hello %s!\n",name);
9      return 0;
10 }
```

Enter your name (maximum 12 characters):Yang✓

Hello Yang!

Enter your name (maximum 12 characters):Yang Li-wei✓

Hello Yang!

Why?



【例10.1】从键盘输入一个人名把它显示在屏幕上

```
1  #include <stdio.h>
2  #define N 12
3  int  main()
4  {
5      char  name[N];
6      printf("Enter your name(maximum 12 characters):");
7      scanf("%s", name);
8      printf("Hello %s!\n",name);
9      scanf("%s", name);          /* 读取输入缓冲区中余下的上次未被读走的字符 */
10     printf("Hello %s!\n",name);
11     return 0;
12 }
```

```
Enter your name (maximum 12 characters):Yang Li-wei✓
Hello Yang!
Hello Li-wei!
```

【例10.2】使用函数gets(), 从键盘输入一个带有空格的人名, 然后把它显示在屏幕上

```
1  #include <stdio.h>
2  #define N 12
3  int  main()
4  {
5      char  name[N];
6      printf("Enter your name(maximum 12 characters):");
7      gets(name);
8      printf("Hello %s!\n",name);
9      return 0;
10 }
```

```
Enter your name (maximum 12 characters):Yang Li-wei✓
Hello Yang Li-wei!
```


【例10.2】使用函数gets(), 从键盘输入一个带有空格的人名, 然后把它显示在屏幕上

```
1  #include <stdio.h>
2  #define N 12
3  int  main()
4  {
5      char  name[N];
6      char *ptrName = name; /* 声明了一个指向数组 name 的字符指针 ptrName */
7      printf("Enter your name(maximum 12 characters):");
8      gets(ptrName); /* 输入字符串存入字符指针 ptrName 所指向的内存 */
9      printf("Hello %s!\n", ptrName);
10     return 0;
11 }
```

```
Enter your name (maximum 12 characters):Yang Li-wei✓
Hello Yang Li-wei!
```

【例10.3】从键盘输入一个带有空格的人名，然后在显示人名的前面显示"Hello", I said to

```
1  #include <stdio.h>
2  #define N 12
3  int  main()
4  {
5      char  name[N];
6      char  str[] = "\"Hello\", I said to";
7      printf("Enter your name(maximum 12 characters):");
8      fgets(name, sizeof(name), stdin);
9      printf("%s %s.\n", str, name);
10     return 0;
11 }
```

```
Enter your name (maximum 12 characters):Yang Li-wei↵
"Hello", I said to Yang Li-wei.
```

【例10.3】从键盘输入一个带有空格的人名，然后在显示人名的前面显示"Hello", I said to

```
1  #include <stdio.h>
2  #define N 12
3  int main()
4  {
5      char name[N];
6      char *ptrName = "\"Hello\", I said to";
7      printf("Enter your name(maximum 12 characters):");
8      fgets(name, sizeof(name), stdin);
9      printf("%s %s.\n", ptrName, name);
10     return 0;
11 }
```

```
Enter your name (maximum 12 characters):Yang Li-wei✓
"Hello", I said to Yang Li-wei.
```

10.5字符串处理函数

■ `#include <string.h>`

`strlen (字符串) ;`

string length

`strcpy (目的字符串, 源字符串) ;`

string copy

`strcat (目的字符串, 源字符串) ;`

string
combination

`strcmp (字符串1, 字符串2) ;`

string
comparison

计算字符串长度

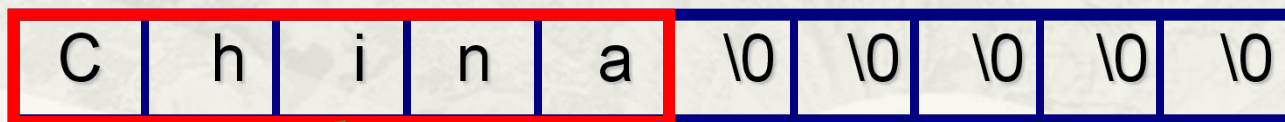
- `#include <string.h>`

`strlen(字符串);`

string length

```
char str[10] = {"China"};  
printf("%d", strlen(str));
```

打印结果是 5, 6, 还是10?



不包括\0的实际字符的个数



计算字符串长度

- `#include <string.h>`

`strlen(字符串);`

string length

```
len = strlen(str);  
for (i=0; i<len; i++)  
{  
    putchar(str[i]);  
}  
putchar('\n');
```

用长度控制字符串输出

字符串复制

- `#include <string.h>`

`strcpy(目的字符串, 源字符串);`

string copy

字符串能否用=整体复制?

`str2 = str1;`



`strcpy(str2, str1);`

注意复制的方向!
`str2`必须足够大!



字符串连接

- `#include <string.h>`

`strcat(目的字符串, 源字符串);`

string
combination

`strcat(str2, str1);`

str2必须足够大!

H	e	l	l	o	\0	\0	\0	\0	\0	\0	\0
---	---	---	---	---	----	----	----	----	----	----	----

C	h	i	n	a	\0
---	---	---	---	---	----

H	e	l	l	o	C	h	i	n	a	\0	\0
---	---	---	---	---	---	---	---	---	---	----	----

字符串比较

- `#include <string.h>`

`strcmp(字符串1, 字符串2);`

string
comparison

字符串能否用`>`,`<`,`==`比较大小?

`if (str2 == str1)`



`if (strcmp(str2, str1) == 0)`



字符串比较

- `#include <string.h>`

`strcmp(字符串1, 字符串2);`

string
comparison

字符串是如何比较大小的？

compare
computer

判断compare 小于 computer?
`strcmp(str1, str2) < 0`为真

当出现第一对不相等的字符时，就由这两个字符决定所在字符串的大小，返回其ASCII码比较的结果值



【例10.4】按奥运会 参赛国国名在字典 中的顺序对其入场 次序进行排序

```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 10 /* 字符串最大长度 */
#define N 150 /* 字符串个数 */
void SortString(char str[][MAX_LEN], int n);
int main()
{
    int i, n;
    char name[N][MAX_LEN]; /* 定义二维字符数组 */
    printf("How many countries?");
    scanf("%d", &n);
    getchar(); /* 读走输入缓冲区中的回车符 */
    printf("Input their names:\n");
    for (i=0; i<n; i++)
    {
        gets(name[i]); /* 输入n个字符串 */
    }
    SortString(name, n); /* 字符串按字典顺序排序 */
    printf("Sorted results:\n");
    for (i=0; i<n; i++)
    {
        puts(name[i]); /* 输出排序后的n个字符串 */
    }
    return 0;
}
```

```
How many countries? 5 ✓
Input their names:
America ✓
England ✓
Australia ✓
Sweden ✓
Finland ✓
Sorted results:
America
Australia
England
Finland
Sweden
```

【例10.4】按奥运会参赛国国名在字典中的顺序对其入场次序进行排序

```
void SortString(char str[][MAX_LEN], int n)
{
    int    i, j;
    char   temp[MAX_LEN];
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp(str[j], str[i]) < 0)
            {
                strcpy(temp, str[i]);
                strcpy(str[i], str[j]);
                strcpy(str[j], temp);
            }
        }
    }
}
```


缓冲区溢出攻击

- 网络黑客常常针对系统和程序自身存在的漏洞，编写相应的攻击程序
 - * 其中, 最常见的就是对缓冲区溢出漏洞的攻击
 - * 几乎占到了网络攻击次数的一半以上
- 世界上第一个缓冲区溢出攻击
 - * **Internet蠕虫**，曾造成全球多台网络服务器瘫痪
- 何谓缓冲区溢出攻击？
 - * 利用缓冲区溢出漏洞进行的攻击



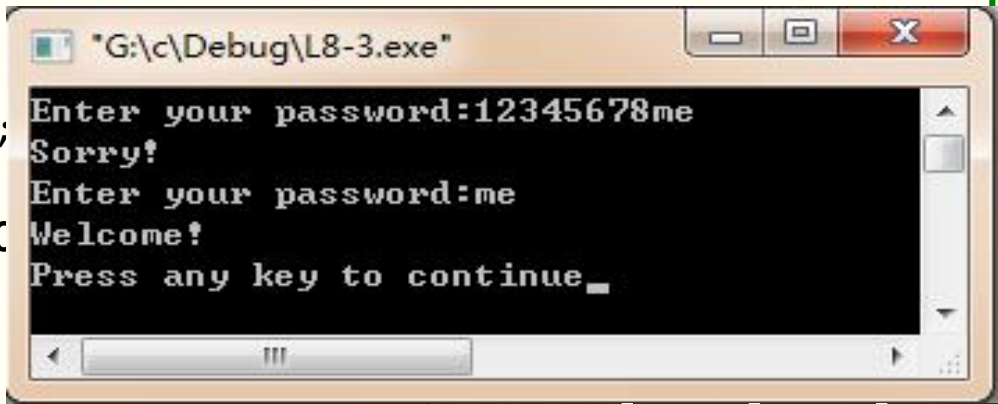
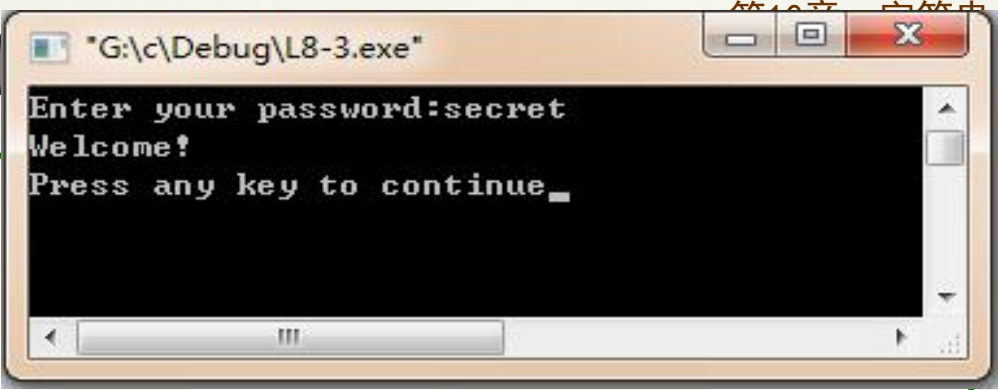
缓冲区溢出攻击

- 易引起缓冲区溢出攻击、不安全的函数
 - * `gets()`、`scanf()`、`strcpy()` 等不限制字符串长度，不对数组越界进行检查和限制，导致有用的堆栈数据被覆盖，给黑客攻击以可乘之机
- 对缓冲区溢出漏洞进行攻击的后果
 - * 程序运行失败、系统崩溃和重启等
 - * 利用缓冲区溢出，执行非授权指令，甚至取得系统特权，进而进行各种非法操作
- 防止和检测缓冲区溢出攻击
 - * 成为防御网络入侵和入侵检测的重点之一



缓冲区溢出攻击实例

```
#include <stdio.h>
#include <string.h>
int main()
{
    char password[8] = "secret", input[8];
    while (1)
    {
        printf("Enter your password:");
        gets(input);
        if (strcmp(input, password) == 0)
        {
            printf("Welcome!\n");
            break;
        }
    }
}
```



Memory

Address: Bytes:

(e.g. 0x401060, or &variable, or \$eax)

0x22ff10: 31 32 33 34 35 36 37 38 6d 65 00 72 65 74 00 00 12345678me.ret..

0x22ff20: 00 f0 fd 7f 00 00 00 00 68 ff 22 00 b6 10 40 00 .8ý!....hÿ".q.0.

原始密码串: 73 65 63 72 65 74

被覆盖后变成: 6d 65 00

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	!	64	40	@	96	60	'
^A	1	01		SOH	33	21	!"	65	41	A	97	61	a
^B	2	02		STX	34	22	"#\$	66	42	B	98	62	b
^C	3	03		ETX	35	23	"#\$%	67	43	C	99	63	c
^D	4	04		EOT	36	24	"#\$%&	68	44	D	100	64	d
^E	5	05		ENQ	37	25	"#\$%&'	69	45	E	101	65	e
^F	6	06		ACK	38	26	"#\$%&'(70	46	F	102	66	f
^G	7	07		BEL	39	27	"#\$%&'()	71	47	G	103	67	g
^H	8	08		BS	40	28	"#\$%&'()	72	48	H	104	68	h
^I	9	09		HT	41	29	"#\$%&'()	73	49	I	105	69	i
^J	10	0A		LF	42	2A	"#\$%&'()	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	"#\$%&'()	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	"#\$%&'()	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	"#\$%&'()	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	"#\$%&'()	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	"#\$%&'()	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	"#\$%&'()	80	50	P	112	70	p
^Q	17	11		DC1	49	31	"#\$%&'()	81	51	Q	113	71	q
^R	18	12		DC2	50	32	"#\$%&'()	82	52	R	114	72	r
^S	19	13		DC3	51	33	"#\$%&'()	83	53	S	115	73	s
^T	20	14		DC4	52	34	"#\$%&'()	84	54	T	116	74	t
^U	21	15		NAK	53	35	"#\$%&'()	85	55	U	117	75	u
^V	22	16		SYN	54	36	"#\$%&'()	86	56	V	118	76	v
^W	23	17		ETB	55	37	"#\$%&'()	87	57	W	119	77	w
^X	24	18		CAN	56	38	"#\$%&'()	88	58	X	120	78	x
^Y	25	19		EM	57	39	"#\$%&'()	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	"#\$%&'()	90	5A	Z	122	7A	z
^[_	27	1B		ESC	59	3B	"#\$%&'()	91	5B	[123	7B	{
^`	28	1C		FS	60	3C	"#\$%&'()	92	5C	\	124	7C	
^_	29	1D		GS	61	3D	"#\$%&'()	93	5D]	125	7D	}
^~	30	1E		RS	62	3E	"#\$%&'()	94	5E	^	126	7E	~
^^	31	1F		US	63	3F	"#\$%&'()	95	5F	_	127	7F	?

ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

字符串的安全输入方法

```
#include <stdio.h>
#include <string.h>
int main()
{
    char password[8] = "secret", input[8];
    while (1)
    {
        printf("Enter your password:");
        scanf("%7s", input);
        if (strcmp(input, password) == 0)
        {
            printf("Welcome!\n");
            break;
        }
        else
        {
            printf("Sorry!\n");
        }
    }
    return 0;
}
```

A screenshot of a Windows command prompt window. The title bar reads "G:\c\Debug\L8-3.exe". The window has standard Windows window controls (minimize, maximize, close). The command prompt shows the following text:
Enter your password:12345678me
Sorry!
Enter your password:Sorry!
Enter your password:me
Sorry!
Enter your password:
The text is displayed in a monospaced font on a black background. The window has a scrollbar on the right side.

字符串的安全输入方法

```
#include <stdio.h>
#include <string.h>
int main()
```

```
{
```

```
    char password[8] = "
```

```
    while (1)
```

```
    {
```

```
        printf("Enter your password:");
```

```
        fgets(input, sizeof(input), stdin);
```

```
        if (strcmp(input, password) == 0)
```

```
        {
```

```
            printf("Welcome!\n");
```

```
            break;
```

```
        }
```

```
    else
```

```
    {
```

```
        printf("Sorry!\n");
```

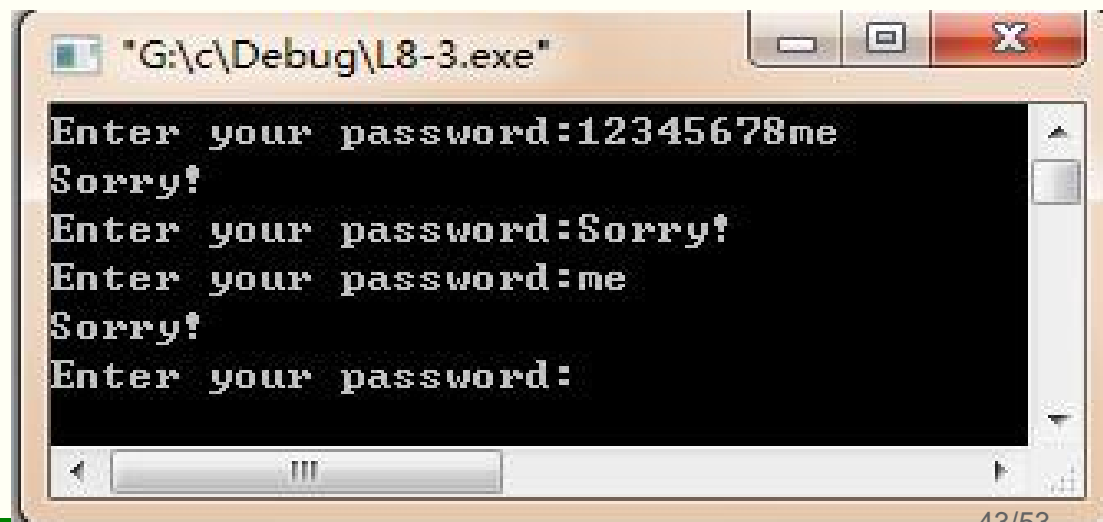
```
    }
```

```
    }
```

```
    return 0;
```

```
}
```

限制输入字符串的长度，更灵活



10.5字符串处理函数

函数功能	函数调用的一般形式	功能描述及其说明
求字符串长度	<code>strlen(str);</code>	由函数值返回字符串 <code>str</code> 的实际长度,即不包括'\0'在内的实际字符的长度
字符串拷贝	<code>strcpy(str1,str2);</code>	将字符串 <code>str2</code> 复制到字符数组 <code>str1</code> 中,这里应确保字符数组 <code>str1</code> 的大小足以存放得下字符串 2
字符串比较	<code>strcmp(str1,str2);</code>	<p>比较字符串 <code>str1</code> 和字符串 <code>str2</code> 的大小,结果分为 3 种情况:</p> <ul style="list-style-type: none"> • 当 <code>str1</code> 大于 <code>str2</code> 时,函数返回值大于 0 • 当 <code>str1</code> 等于 <code>str2</code> 时,函数返回值等于 0 • 当 <code>str1</code> 小于 <code>str2</code> 时,函数返回值小于 0 <p>字符串的比较方法为:对两个字符串从左至右按字符的 ASCII 码值大小逐个字符相比较,直到出现不同的字符或遇到'\0'为止</p>
字符串连接	<code>strcat(str1,str2);</code>	将字符串 <code>str2</code> 添加到字符数组 <code>str1</code> 中的字符串的末尾,字符数组 <code>str1</code> 中的字符串结束符被字符串 <code>str2</code> 的第一个字符覆盖,连接后的字符串存放在字符数组 <code>str1</code> 中,函数调用后返回字符数组 <code>str1</code> 的首地址。这里,字符数组 <code>str1</code> 应定义得足够大,以便能存放连接后的字符串
“n 族”字符串拷贝	<code>strncpy(str1,str2,n)</code>	将字符串 <code>str2</code> 的至多前 <code>n</code> 个字符拷贝到字符数组 <code>str1</code> 中
“n 族”字符串比较	<code>strncmp(str1,str2,n)</code>	函数 <code>strncmp(str1, str2, n)</code> 的功能与函数 <code>strcmp(str1, str2)</code> 类似,它们的不同之处在于,前者最多比较 <code>n</code> 个字符
“n 族”字符串连接	<code>strncat(str1,str2,n)</code>	将字符串 <code>str2</code> 的至多前 <code>n</code> 个字符添加到字符串 <code>str1</code> 的末尾。 <code>str1</code> 的字符串结束符被 <code>str2</code> 中的第一个字符覆盖

10.6向函数传递字符串

■ 向函数传递字符串时

- * 既可用**字符数组**作函数参数
- * 也可用**字符指针**作函数参数

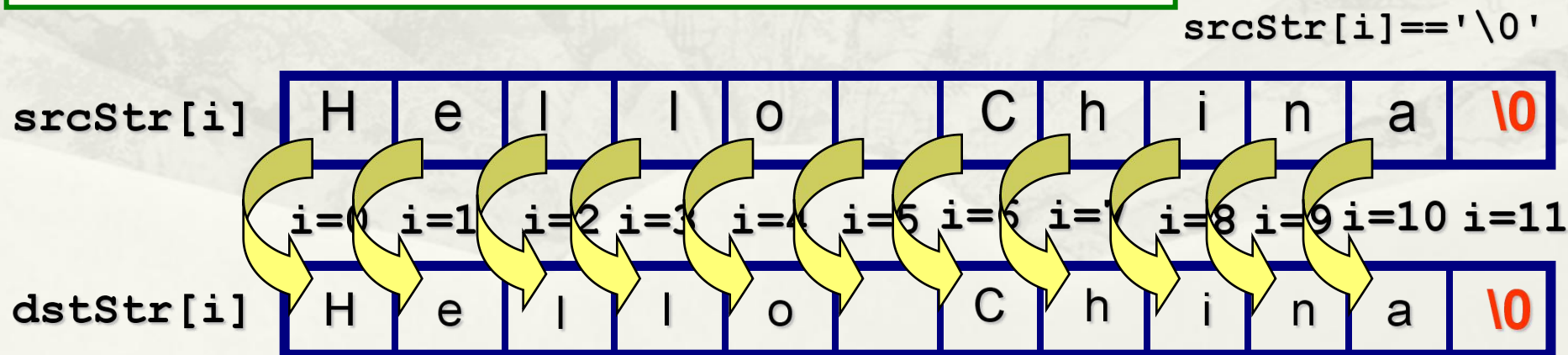
■ 传地址调用

- * 传字符串的首地址，而非字符串中的全部字符



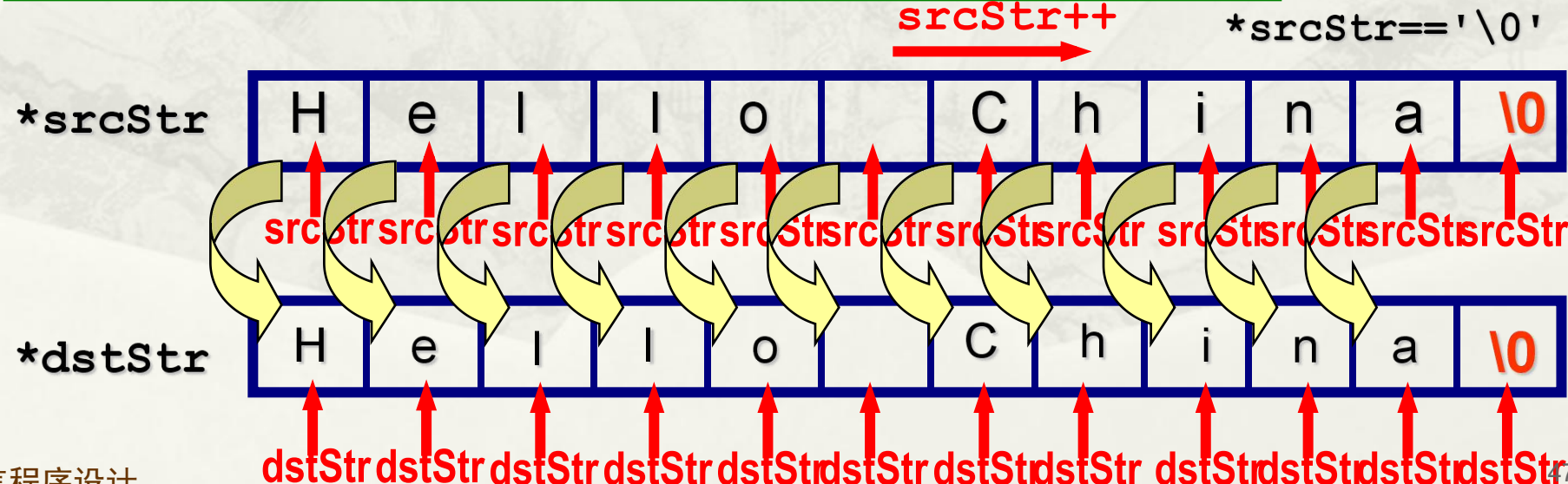
例10.5：字符串拷贝—用字符数组编程

```
void MyStrcpy(char dstStr[], char srcStr[])
{
    int i = 0;
    while (srcStr[i] != '\0')
    {
        dstStr[i] = srcStr[i];
        i++;
    }
    dstStr[i] = '\0';
}
```



例10.5：字符串拷贝—用字符指针编程

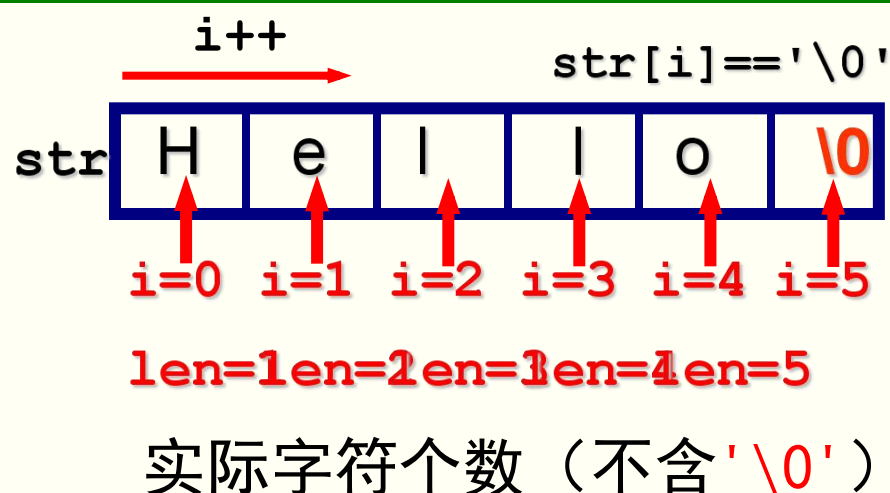
```
void MyStrcpy(char *dstStr, char *srcStr)
{
    while (*srcStr != '\0')
    {
        *dstStr = *srcStr;
        srcStr++;
        dstStr++;
    }
    *dstStr = '\0';
}
```



例10.6：计算实际字符个数

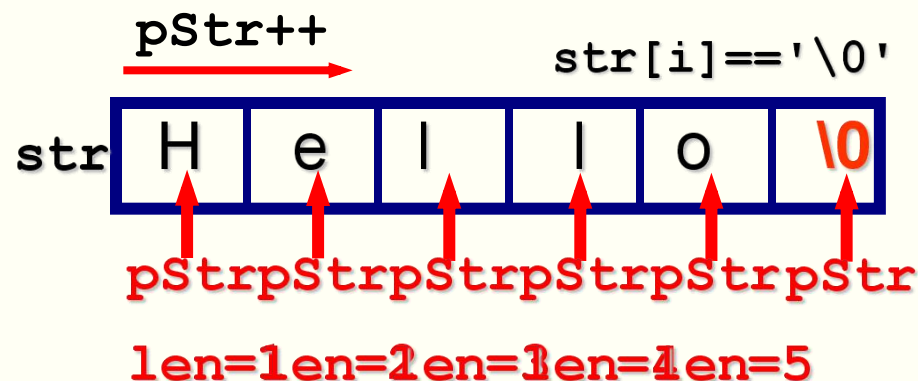
```
unsigned int  MyStrlen(char str[])
{
    int  i;
    unsigned int len = 0;
    for (i=0; str[i]!='\0'; i++)
    {
        len++;
    }
    return len;
}
```

■ 用字符数组实现



```
unsigned int  MyStrlen(char *pStr)
{
    unsigned int len = 0;
    for (; *pStr!='\0'; pStr++)
    {
        len++;
    }
    return len;
}
```

■ 用字符指针实现



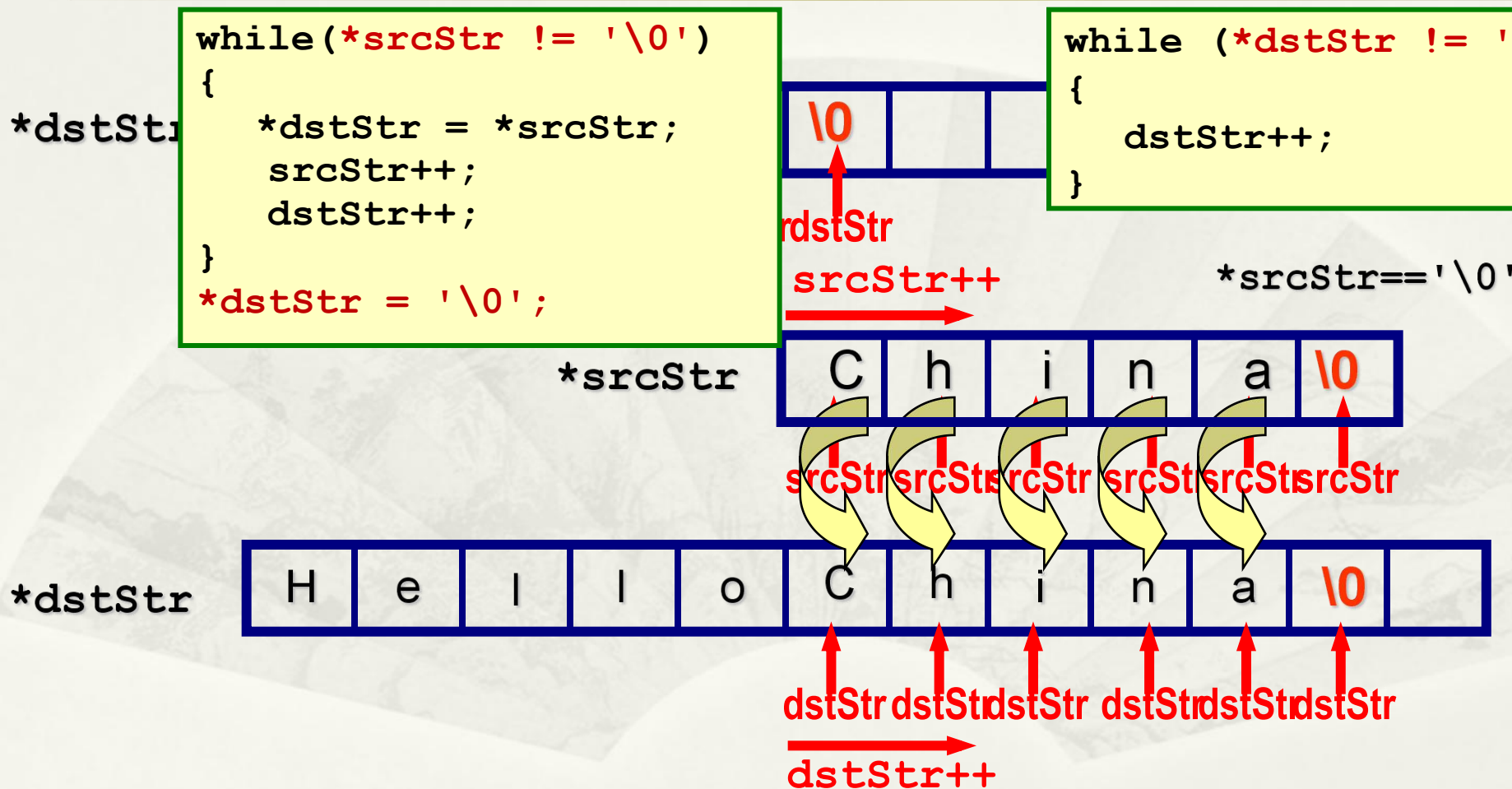
10.7从函数返回字符指针

【例10.7】编程实现strcat()的功能

```
#include <stdio.h>
#define N 80
char *MyStrcat(char *dstStr, char *srcStr);
int main()
{
    char first[2*N];    /* 这个数组应该足够大 */
    char second[N];
    char *result = NULL;
    printf("Input the first string:");
    gets(first);
    printf("Input the second string:");
    gets(second);
    result = MyStrcat(first, second);
    printf("The result is: %s\n", result);
    return 0;
}
```

```
Input the first string:Hello✓
Input the second string:China✓
The result is: HelloChina
```

10.7从函数返回字符指针



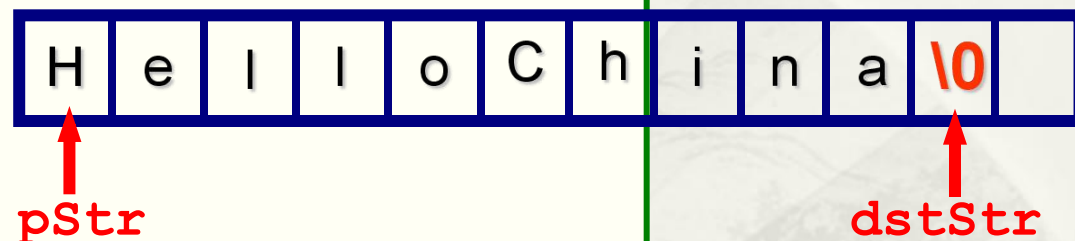
【例10.7】编程实现`strcat()`的功能

10.7从函数返回字符指针

【例10.7】编程实现strcat()的功能

```
char *MyStrcat(char *dstStr, char *srcStr)
```

```
{  
    char *pStr = dstStr;  
    while (*dstStr != '\0')  
    {  
        dstStr++;  
    }  
    while(*srcStr != '\0')  
    {  
        *dstStr = *srcStr;  
        srcStr++;  
        dstStr++;  
    }  
    *dstStr = '\0';  
    return pStr;  
}
```



返回字符串首地址

小结

- 明确字符串被保存到了哪里
- 明确字符指针指向了哪里
 - 指向字符串常量的字符指针
 - 指向字符数组的字符指针
- 向函数传递字符串
 - 向函数传递字符数组
 - 向函数传递字符指针
- 字符串处理函数



