字符数组

- 即数组中每一个元素的类型都是字符型，其主要用于存储一串连续的字符。
- 格式：**char** 数组名[字符个数];
- 例：char c[8];
- 引用及赋值：c[0]='O'; c[1]='K'; ……

字符数组初始化

- 为数组中的每个元素指定初值
  - 例：char c[5] = { ‘H’,‘e’,‘l’,‘l’,‘o’ } ;
  - 此时也可省略数组长度5。

H	e	l	l	o

- 初值个数小于数组长度时，多余元素自动赋’\0’
  - 例：char c[7] = { ‘H’,‘e’,‘l’,‘l’,‘o’ } ;

H	e	l	l	o	\0	\0

- 初值个数不能大于数组长度。
- 二维字符数组定义和初始化
  - char c[2][3] = {{‘a’, ‘b’, ‘c’}, {‘A’, ‘B’, ‘C”}};

字符串

- 前面已经介绍过，字符串是指位于一对双引号中的任何字符。编译器会自动在双引号里字符的结尾处添加空字符'\0' 作为字符串的结束。\0'也要占用一个字节。
- 例：“hello”被存储时占用几个字节？
- 在C中，字符串的处理是用字符数组完成的。

字符数组处理字符串

- 定义一个字符数组存放字符串时，必须明确指定数组大小。
  - char c[6] = { “Hello” } ;
  - char c[6] = “Hello” ;
  - char c[6] = { ‘H’,‘e’,‘l’,‘l’,‘o’,‘\0’ } ;

  } 三种形式等价

  - 注意，最后一种结尾必须有’\0’，否则就是字符数组，而不是字符串。
- 指定数组大小时，一定要确保数组元素个数比字符串长度至少多1。
- 初始化字符数组时，可省数组大小，由编译器决定。在处理字符串时，这种方式会很方便。
  - 例： char c[ ] = “Hello” ;

字符串的输入

- 要把一个字符串读到程序中，必须首先预留足够大的存储区来存放希望读入的字符串，然后使用输入函数获取这个字符串；千万不要指望计算机在读入的时候会先计算字符串的长度，然后为字符串分配空间。
- 例：char name[20]; 要事先指定name的大小。现在name是一个已经分配20个字节存储块的地址。
- 为字符串预留空间后，就可以读取字符串了。C库提供了三个读取字符串的函数：scanf( )、gets( )和fgets( )。

gets( )函数

- gets( )函数从系统标准输入设备（通常是键盘）获得一个字符串。
- gets( )读取换行符（\n，即回车）之前的所有字符，并在这些字符后添加一个空字符\0。而读到的换行符会被gets( )丢弃。
- 为使用该函数，应包含头文件stdio.h
- 常用方式： gets(存放字符串的地址);
- 不足：该函数不检查预留存储区是否能够容纳实际输入的数据。当输入字符数目大于字符数组的长度时， 多出的字符则会溢出到相邻的内存区。

gets( )函数示例

```
//例：读取一个名字
#include  <stdio.h>
#define  MAX   20
int  main(void)
{
      char  name[MAX];
      printf("What's your name?");
      gets(name);
      printf("name: %s.\n", name);
      return 0;
}
```

fgets( )函数

- 因gets( )函数不检查目标数组是否能够容纳输入，所以不安全。
- fgets( )函数改进了gets( )函数的不足，它需要指定最大读入的字符数。所以fgets( )函数比gets( )函数安全。
- 该函数是为文件I/O设计的，目前暂时不讲，在后面我们会学习它。

scanf( )函数

- scanf( )函数可以使用%s来读入一个字符串。
- 与gets( )主要的区别在于它们如何决定字符串何时结束。
  - scanf( )使用%s格式读入字符串时，以遇到的第一个非空白字符开始读取，以读到（但不包括）下一个空白字符（比如空格、制表符或换行符）结束；
  - 或者，如果指定了字段宽度，如%10s，scanf( )就会读入10个字符或直到第一个空白字符终止输入。
  - 而gets( )会读取所有字符，直到遇到第一个换行符终止。
  - 相比而言gets( )常用，而scanf( )更适合输入一个单词。

scanf( )函数示例

```
#include <stdio.h>
int main(void)
{
    char name1[10], name2[10];
    printf("Please enter 2 names: \n");
    scanf("%5s %s", name1, name2);
    printf("name1: %s  and name2: %s.\n", name1, name2);
    return 0;
}
```

- 注意：用格式%s 输入字符串时，字符数组变量name1, name2前不必加取地址符&，因为数组名本身代表数组的首地址

字符串的输出

- 相应于字符串的输入，字符串的输出在C库中也有三个标准函数：puts( )、fputs( )和printf( )。

puts( )函数

- 使用方式：puts(要输出的字符串起始地址);
- 一次输出一个字符串，输出时将遇到的'\0'自动转换成换行符。
- 注意：gets( )丢掉输入里的换行符，但是puts( )为输出添加换行符。

- 例：有如下程序片段

```
char str[20] = "I love china!";
puts("My name is lucy.");
puts(str);
puts(&str[7]);
```

- 显示结果如下：

```
My name is lucy.
I love china!
china!
```

fputs( )函数

- 与puts( )不同，fputs( )并不为输出自动添加换行符。
- fputs( )函数是为文件I/O设计的。目前暂时不讲，在后面我们会学习它。

printf( )函数

- 使用%s输出字符串时，printf( )需要一个字符串地址作为参数。
- printf( )使用起来没有puts( )方便，但它可以格式化多种数据类型，因而更通用。
- 和puts( )不同之处是printf( )并不自动在新行上输出每一个字符串；如果需要，必须明确指明\n。
  - 例：char str[20] = “I love china!”;

    printf(“%s\n”, str); 和 puts(str); 效果一样。

自定义字符串输入输出

- 不一定非要使用系统提供的库函数对字符串进行输入和输出，我们可以使用getchar( )和putchar( )完成对字符串的输入和输出。

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    char  str[20];
    while ((str[i++]=getchar( )) !='\n' ) ;
    str[i-1] = '\0';
    for (i=0; str[i]!='\0'; i++)
        putchar(str[i]);
    return 0;
}
```

字符串数组

- 字符串数组就是数组中的每一个元素是一个字符串。
- 因字符串本身就是一个字符数组，所以字符串数组实际上是一个二维字符数组。
- 例：char str[5][20];
- 该二维字符数组的第一维表示字符串的个数，第二维表示每个字符串的存储长度。

字符串函数

- 示例：

  char s1[5]="abc", s2[3], s3[8];

  s2 = "abc";　　//错，　赋值与初始化不同

  s3 = s1;　　//错，对s2，s3的赋值都是非法的

- C语言不允许用赋值表达式对字符数组整体赋值。
- 由于字符串有其特殊性，所以系统提供了许多专门处理字符串的函数对其操作。
- 这些函数都包含在头文件string.h中。因此使用这些函数时，应加#include <string.h>

strcpy( )函数

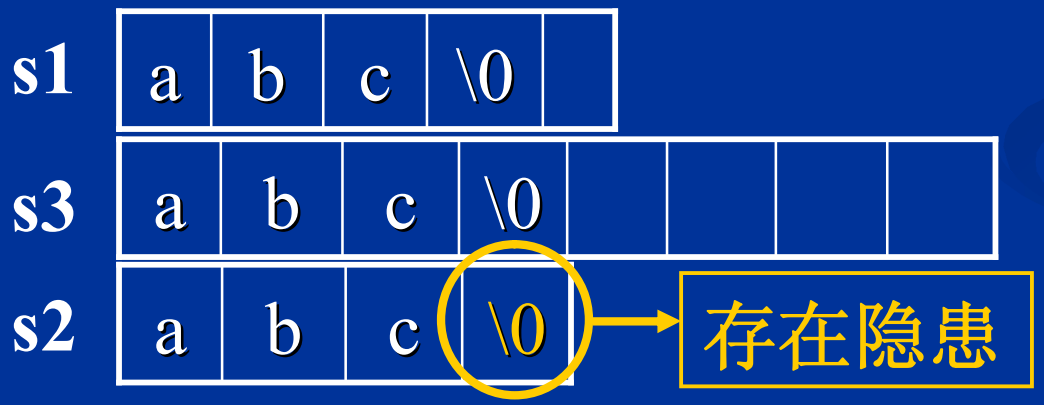
- 字符串拷贝函数
  - 格式： strcpy (target, source)
  - 作用：将source(源)中的字符串复制到target中(目标)
- 注意：
  - target必须自己保证足够大，以容下源字符串中的内容
  - source可以是字符串常量，或是字符数组名
  - 拷贝时‘\0’也一起拷贝
  - 它和赋值语句的顺序一样，目标字符串在左边，便于记忆

strcpy( )示例

char s1[5]="abc" , s2[3], s3[8];

strcpy (s3, "abc") ;

strcpy (s2, s1) ;

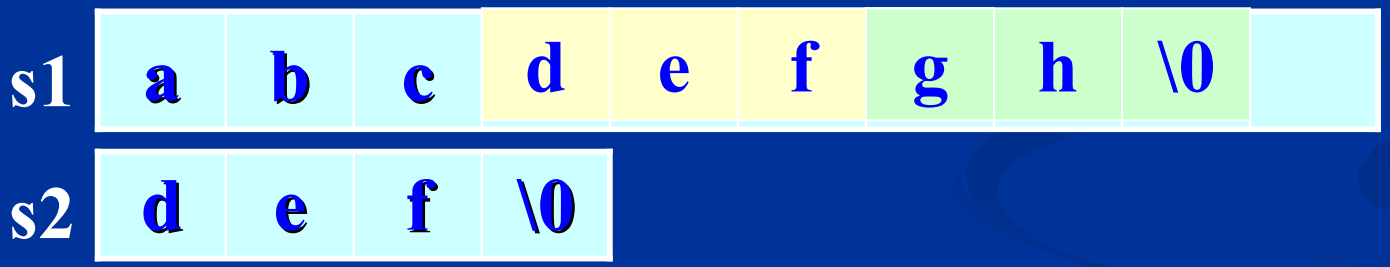


strcat( )函数

- 字符串连接函数
  - 格式:　strcat (字符串1 , 字符串2)
  - 作用: 将字符串2中的字符连接到字符串1 的后面，从而使第一个字符串成为一个新的组合字符串，第二个字符串并没有改变
- 说明:
  - 连接时将字符串1 末尾的‘\0’将去掉，而在连接后的新字符串末尾添加‘\0’
  - 注意：字符数组1要足够大，以容纳连接后的新字符串

strcat( )函数示例

例：char s1[10] = “abc” , s2[ ] = “def” ;
strcat (s1, s2) ;
strcat (s1, “gh”) ;



strcmp( )函数

- 字符串比较函数
  - 格式：strcmp（字符串1，字符串2）
  - 作用：比较两个字符串的大小
- 说明：
  - 两个字符串可能是字符串常量或字符数组变量
  - 两个字符串比较时，从字符串中的第一个字符开始逐个比较其机器编码的（ASCII码）值， 直到出现不同字符或出现‘\0’为止
  - 比较的结果由函数值带回

strcmp( )函数示例

- 该函数的返回值如下：
  - str1 等于 str2，函数值为0
  - str1 大于 str2，函数值为正数
  - str1 小于 str2，函数值为负数

所有对两个字符串比较，不能用以下形式：

```
if (“study” == “student”)
    printf(“yes”);
```

而只能用：

```
if (strcmp(“study”, “student”) == 0)
    printf(“yes”);
```

strlen( )函数

- 测字符串长度函数
  - 格式:　strlen (字符串)
  - 作用: 测出字符串中实际字符的个数(不包括'\0')

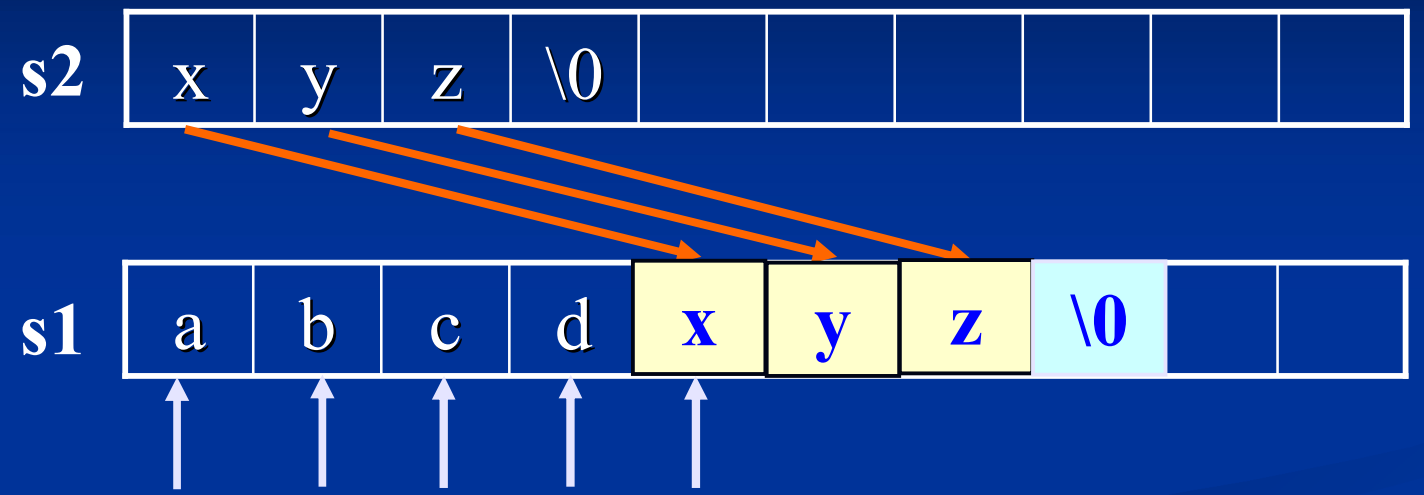
例 :　int　len1, len2 ;

```
char　 s[10] ;
len1 = strlen( "computer") ;
gets(s) ;
len2 = strlen(s) ;
```

strlwr( )和strupr( )函数

- 字符串中大、小字母转换
  - strlwr (字符串)：将字符串中大写字母换成小写字母
  - strupr (字符串)：将字符串中小写字母换成大写字母

- 例：编程实现两个字符串的连接，但不能使用strcat函数



目标：将字符串s2 连接到字符串s1后面

- 步骤：
  - 让数组s1 的下标指向字符串的末尾, 即‘\0’的位置
  - 依次将数组s2 的字符赋给s1, 它们的下标都加1
  - 最后数组s1 的末尾赋值为‘\0’

```
#include  <stdio.h>
int main(void)
{  char s1[80], s2[40];
   int  i=0, j=0;
   printf("Input string1:");
   gets(s1);
   printf("Input string2:");
   gets(s2);
   while(s1[i]!='\0')
         i++;
   while(s2[j]!='\0')
      s1[i++]=s2[j++];
   s1[i]='\0';
   printf("The new string is: %s", s1);
   return 0;
}
```

→ i 和j分别是s1 和s2的下标

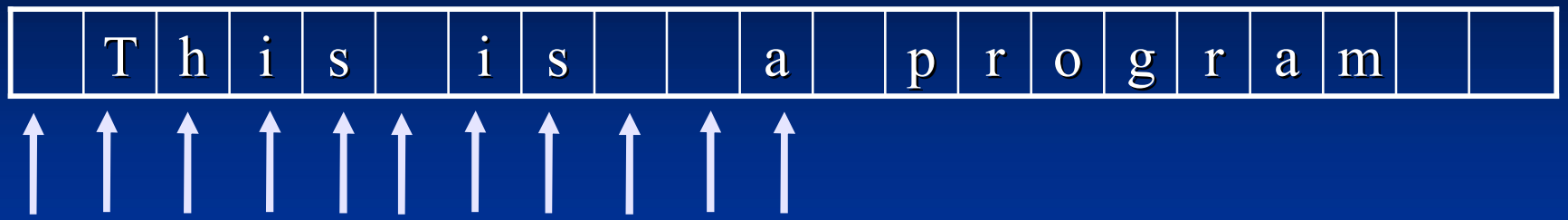
→当元素i不是'\0' 时，让i加1，指向下个元素

→依次将s2的元素j赋给s1的元素i

```
s1[i]=s2[j];
i++;  j++;
```

→S1末尾赋'\0'

- 例：输入一行字符， 统计其中有多少个单词



word 1

- 分析:

  用一个字符数组来保存一行字符，因为单词是由空格分开的, 所以统计单词个数关键在于判断某个字符是否为空格

- 具体方法:

  设置一个标志变量word, 如果当前字符是空格, 则word=0; 如果当前字符不是空格， 则word=1

```
#include <stdio.h>
int main(void)
{  char string[81];
   int  i, num=0, word=0;
   char c;
   gets(string);
   for(i=0; (c=string[i]) != '\0'; i++)
       if (c==' ')  word=0;
       else if (word == 0)
       {     word = 1;
             num++;
       }
   printf ("There are %d words.\n", num);
   return 0;
}
```

num用来统计单词个数，word是判别是否为单词的标志，若word=0表示未出现单词，如出现单词word就置成1

c不是空格，若c前面的字符是空格，表明这是一个新单词的开始，则word=1，num加1；若c前面的字符不是空格，则不作任何处理

数组与指针

- 回忆：数组是由一组具有相同数据类型的元素组成。
- 数组元素在内存中是连续存放的，而数组名就表示这段连续存储单元的首地址。
- 指针变量就是用来存放地址的变量，它和数组的关系非常密切。

一维数组及元素的地址表示

int a[5] = { 1, 2, 3, 4, 5 }; 数组名a表示数组的首地址

元 素	地 址	地 址	元 素
a[0]	&a[0]	a	*a
a[1]	&a[1]	a+1	*(a+1)
a[2]	&a[2]	a+2	*(a+2)
a[3]	&a[3]	a+3	*(a+3)
a[4]	&a[4]	a+4	*(a+4)

重要结论：

除了优先级外，下标引用和间接访问完全相同。

例：

下列两表达式等同

array[下标]

*(array + (下标))

指向一维数组元素的指针变量

```
int   *p,  a[5] = { 1, 2, 3, 4, 5 };
p=&a[0];    //把元素a[0]的地址赋给指针变量p
p=a;        //数组名代表数组的首地址，p指向元素a[0]
```

C语言规定：若p指向数组中的一个元素，则p+1指向数组的下一个元素

地址	元素	元素
p	*p	p[0]
p+1	*(p+1)	p[1]
p+2	*(p+2)	p[2]
p+3	*(p+3)	p[3]
p+4	*(p+4)	p[4]

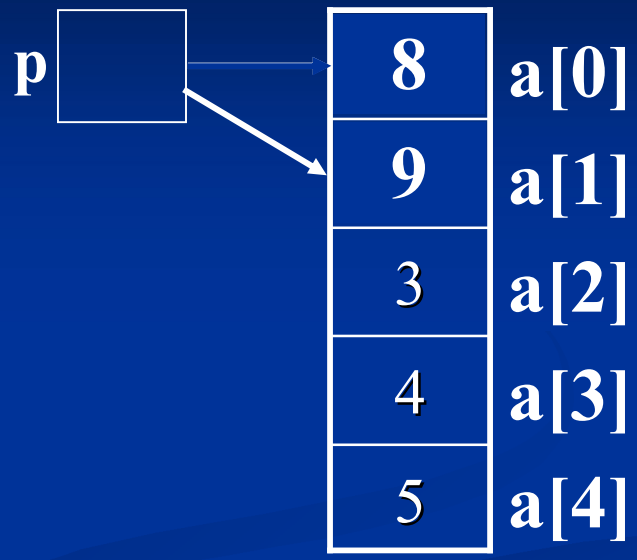
还能如何表示元素？

通过指针引用数组元素

例：int *p, a[5] = { 1, 2, 3, 4, 5 };

p=a; *p=8;

p=p+1; *p=9;



若p=a，或p=&a[0];

则p+i和a+i是元素a[i]的地址，

而*(p+i)和*(a+i)就是数组元素a[i]

数组元素的访问方法
- 下标法：数组名[下标]
- 指针法：*(数组名+(下标)) 或 *指针变量

```
for(i=0; i<5; i++)
    printf("%d", *(a+i));
```

```
for(p=a; p<(a+5); p++)
    printf("%d", *p);
```

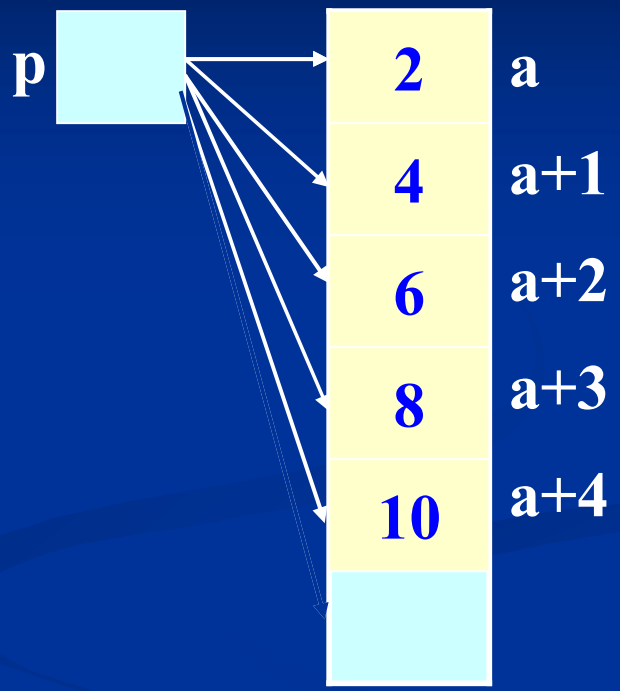
- 例：用三种方式输出数组元素

```
#include <stdio.h>
int  main(void)
{
    int   a[5]={1, 3, 5 ,7 ,9 }, *p , i ;
    for( i=0; i<5; i++)
          printf("%3d", a[i]);
    printf("\n") ;
    for(i=0; i<5; i++)
          printf("%3d", *(a+i));
    printf("\n") ;
    for (p=a; p<a+5; p++)
          printf("%3d", *p);
    printf("\n") ;
    return  0;
}
```

注意：可以用p++ , 但不能用a++, 因为a 代表数组的首地址, 它是地址常量, 不能改变, 而p是一个指针变量, 可以改变

- 例: 将数组a中全部元素加1，再输出a

```
#include <stdio.h>
int  main(void)
{
    int   a[5] = {1, 3, 5 ,7 ,9 },
    int   *p , j ;
    for ( p=a ; p<a+5 ; p++ )
        printf("%3d", *p);
    printf("\n") ;
    for ( j=0 ; j<5 ; j++)
        a[j]=a[j]+1 ;
    p=a ;
    for ( j=0 ; j<5 ; j++)
        printf("%3d", *(p+j)) ;
    printf("\n") ;
    return  0;
}
```



使用指针变量要注意它的当前值

数组和指针完全相等吗？

- 通过前面的学习，你会认为数组和指针是相等的吗？
- 考虑如下声明：

```
int a[5];
int *p;
```

a和p可以互换吗？

- 结论：虽然它们都具有指针值，都可以进行间接访问和下标引用操作，但它们还是存在相当大的区别。
  - 内存空间分配不同，指向位置不同
  - *a合法，*p非法
  - p++可以通过编译，a++却不行

数组名作函数参数

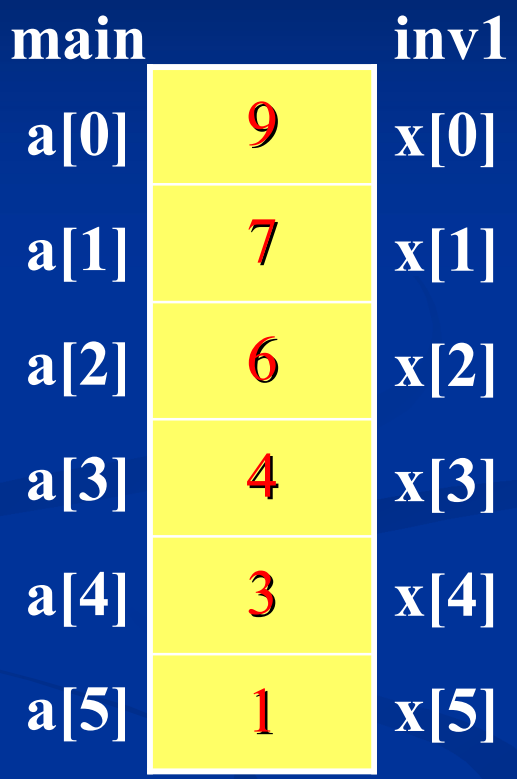
- 当一个数组名作为参数传递给一个函数时会发生什么情况呢？
- 数组名的值就是一个指向数组第一个元素的指针，作为函数参数时，它传递给函数的是一份该指针的拷贝。
- 因此，函数如果执行下标引用，实际上是对这个指针（形参）执行间接访问操作，并且通过这种间接访问，函数可以访问和修改主调函数的数组元素，而不会影响主调函数的实参值本身（但可能修改它所指向的内容）
- 具体内容看如下示例

例：编写函数实现数组元素逆序

①实参和形参都用数组名

```
#include <stdio.h>
void inv1( int x[ ] , int n )
{   int  temp, i , j , m=(n−1)/2;
    for( i=0 ; i<=m ; i++)
    {      j = n−1−i ;
           temp = x[i] ;
           x[i] = x[j] ;
           x[j] = temp ;
    }
}
int main(void)
{   int  i , a[6]={ 1, 3, 4, 6, 7, 9 };
    inv1(a, 6 );
    for( i=0; i<6; i++ )
          printf("%3d", a[i] );
    printf("\n");
    return 0;
 }
```

无需指定数组长度，其本质是个地址。若想操作具体数目的元素，需指定另一个参数



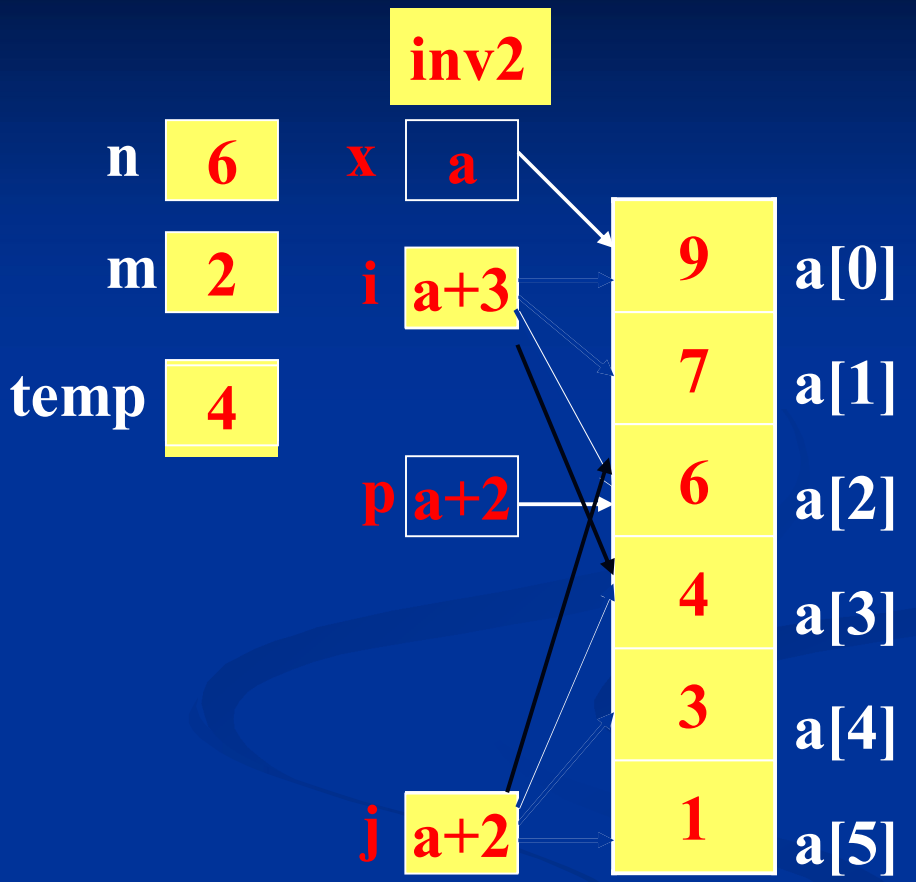
- 如果把一个数组名作为实参传递给函数，正确的函数形参应该是什么？指针or数组？
- 通过前面的例子可以看出，函数的形参用的是数组。其实调用函数时实际传递的是一个指针，所以函数的形参实际上是个指针，但用数组名作形参更容易新手理解。因此下面两个函数原型是相等的：

```
void  inv1( int x[ ],  int n );
void  inv1( int *x,  int n );
```

- 注意：
  - 这两个声明只是在特定的上下文环境中是相等的。
  - 这两个声明哪个“更加准确”呢？答案是：指针。因为实参虽然是数组名，但实际上是个指针，而不是数组。
  - 进一步明白：
    - 为什么函数定义中的一维数组形参无需写明它的元素数目，因为函数并不为数组参数分配内存空间。形参只是一个指针，它指向的是已经在其他地方分配好内存的空间。
    - 函数确实无法知道数组长度，如果想知道，它必须用另一个参数显式传递给函数

②实参用数组名,形参用指针变量

```
#include <stdio.h>
void inv2(int *x, int n)
{ int temp, m=(n-1)/2;
  int *p, *i, *j ;
  j=x+n-1;
  p=x+m;
  for( i=x; i<=p; i++, j-- )
  {  temp=*i ;
     *i=*j ;
     *j=temp ;
  }
}
int main(void)
{ int i , a[6]={ 1, 3, 4, 6, 7, 9 };
  inv2(a, 6 );
  for( i=0; i<6; i++ )
      printf("%3d", a[i] );
  printf("\n");
  return 0;
}
```



③实参用指针变量，形参用数组名

```
#include <stdio.h>
void inv3(int x[ ] , int n);
int main(void)
{
    int *p, a[6]={1, 3, 4, 6, 7, 9};
    p = a;
    inv3( p, 6 ) ;
    for( p=a ; p<a+6 ; p++ )
        printf("%3d", *p );
    printf("\n");
    return 0;
}
void inv3(int x[ ], int n)
{   int temp, i, j, m=(n-1)/2;
    for( i=0; i<=m; i++)
    {     j = n-1-i;
          temp = x[i];
          x[i] = x[j];
          x[j] = temp;
    }
}
```

④实参和形参都用指针变量

```
#include <stdio.h>
void inv4(int *x, int n);
int main(void)
{
    int *p, a[6]={1, 3, 4, 6, 7, 9};
    p = a;
    inv4( p, 6 );
    for( p=a; p<a+6; p++ )
        printf("%3d", *p );
    printf("\n");
    return 0;
}
void inv4(int *x , int n)
{
    int temp, m=(n-1)/2;
    int *p, *i , *j ;
    j = x+n-1;
    p = x+m;
    for( i=x; i<=p ; i++, j-- )
    {   temp = *i;
        *i = *j ;
        *j = temp;
    }
}
```

小结：如果有一个实参数组，想在函数中改变此数组的元素的值，实参与形参的对应关系有四种

	实参	形参
1	数组名	数组名
2	数组名	指针变量
3	指针变量	数组名
4	指针变量	指针变量

字符串与指针

- 定义指向字符的指针变量
  - 形式：char *变量名
  - 例： char *p = “China”;
- 说明
  - 这里没有定义字符数组，但字符串在内存中还是以数组形式存放的，字符串在内存中占有一片连续的存储单元， 以‘\0’结束。
  - 赋值只是把字符串的首地址赋给p，而不是把字符串赋给 p， p 是一个指针变量，它不能存放一个字符串，只能存放一个地址。

字符串的输出

例：　char　*p = “China”;

　　printf(“%s\n” , p) ;

输出字符串时，先输出p 指向的第一个字符，然后系统自动执行p+1，使之指向下一个字符，再输出该字符，直到遇到‘\0’ 为止。

也可用循环逐个输出字符串中的字符:

例：　char　*p = “China”;

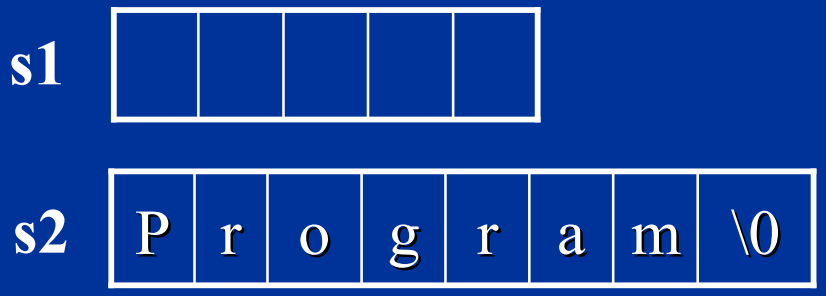
　　for (; *p!=‘\0’; p++)

　　　printf(“%c”, *p) ;

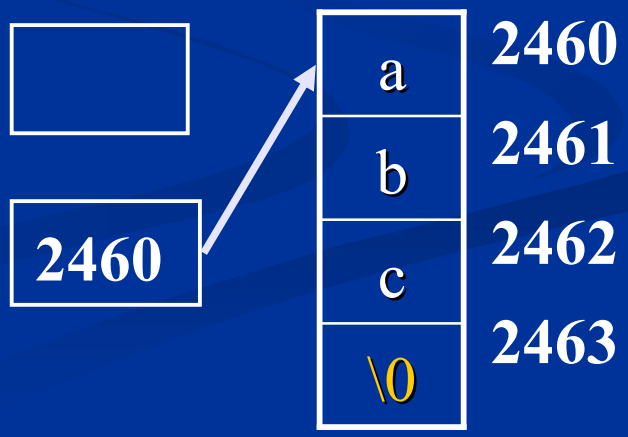
字符数组与字符指针变量的区别

- 存储方式不同:
  - 字符数组在定义时，不论是否进行初始化，都会为其分配存储空间，用来存放字符串中的字符和‘\0’。
  - 字符指针是分配一个指针变量的存储单元，用于存放地址。如果字符指针定义时没有进行初始化，则编译器不为任何字符串分配存储空间；如果定义时进行了初始化，则编译器还要分配一块连续内存空间存储字符串，并将存储空间的起始地址赋给字符指针。

char s1[5], s2[8] =“Program”;



char *p1, *p2=“abc”;



- 赋值方式不同：
  - 字符数组可以初始化，可以给每个元素赋值，但不能整体赋值。若想整体赋值，需借助strcpy( )函数实现。
  - 字符指针可以初始化，也可以整体赋值。

```
char   s[8], str[ ]= “good” ;   //对
s[0]=‘c’;  s[1]=‘h’;            //对
s[8] = “China” ;                //错
s = “China” ;                   //错
```

应该用：strcpy(s, “China”);

```
char  *p1;
char  *p2 = “abcd”;             //对
p1 =  “China” ;                 //对
```

- 数组名表示字符串的首地址，但数组名是一个地址常量，它的值是不能变的，而指针变量的值是可以改变的。

```
char   *p =“Program”;
for ( ; *p!=‘\0’; p++ )
   printf(“%c”, *p ) ;   对
```

```
char  s[10] =“Program” ;
for ( ; *s!=‘\0’; s++ )
      printf(“%c”, *s ) ; 错
```

- 若没有对字符指针变量赋值，该指针变量的值是不确定的，此时不应该对字符指针变量进行操作，否则可能出现异常。

```
char   s1[5], s2[10];
scanf(“%s”, s1);
strcpy(s2, “Hello”);
```

} 这种用法完全正确

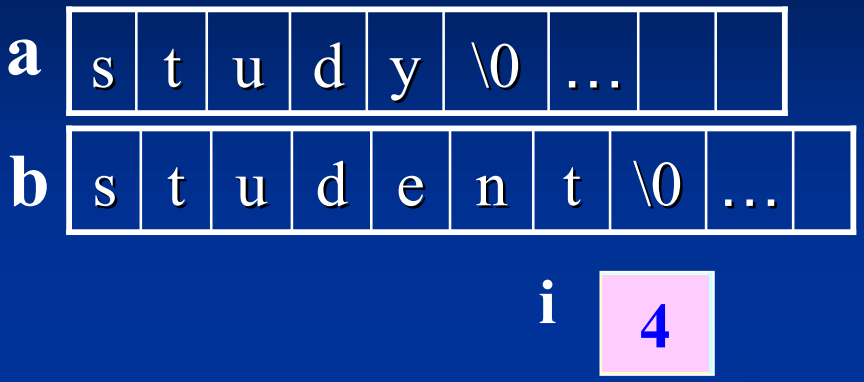
```
char   *p1, *p2;
scanf(“%s”, p1);
strcpy(p2, “Hello”);
```

} 这种用法危险，因为p1，p2未赋值，它们都是随机值，无法确定它们指向内存中的具体位置，执行操作后可能导致异常。

- 例：按字典排列法比较两个单词的大小

方法1：用字符数组实现

```
#include<stdio.h>
int  main(void)
{
    char  a[20], b[20];
    int  i = 0;
    gets(a);
    gets(b);
    while((a[i] == b[i]) && a[i]!= '\0')
        i++;
    if(a[i] == '\0' && b[i] == '\0')
        printf("%s = %s\n", a, b);
    else if(a[i] > b[i])
        printf("%s > %s\n", a, b);
    else
        printf("%s < %s\n", a, b);
    return  0;
}
```



方法2：　用字符指针实现

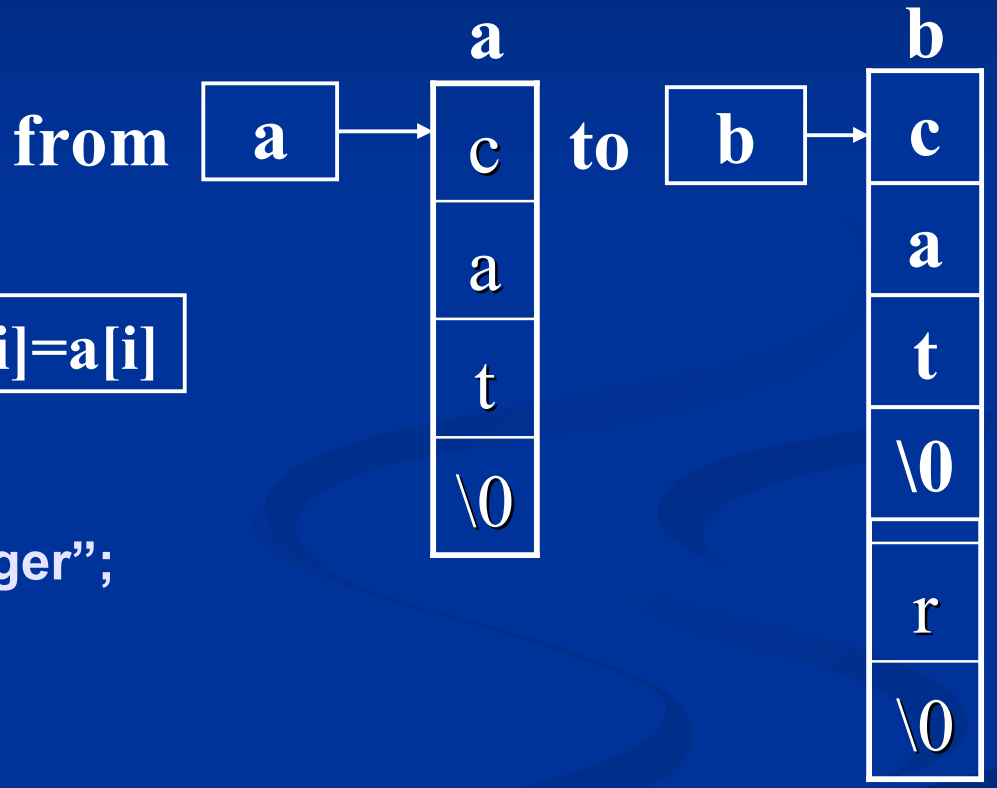
```
#include<stdio.h>
int  main(void)
{
   char  a[20], b[20],  *sa,  *sb;
   sa = a;
   sb = b;
   gets(sa);
   gets(sb);
   while((*sa == *sb) && *sa != '\0')
   {
          sa++;
          sb++;
   }
   if(*sa == '\0' && *sb == '\0')
       printf("%s = %s\n", sa, sb);
   else  if(*sa > *sb)
          printf("%s > %s\n", sa, sb);
   else
        printf("%s < %s\n", sa, sb);
   return  0;
}
```

字符指针作函数参数

例：实现字符串复制

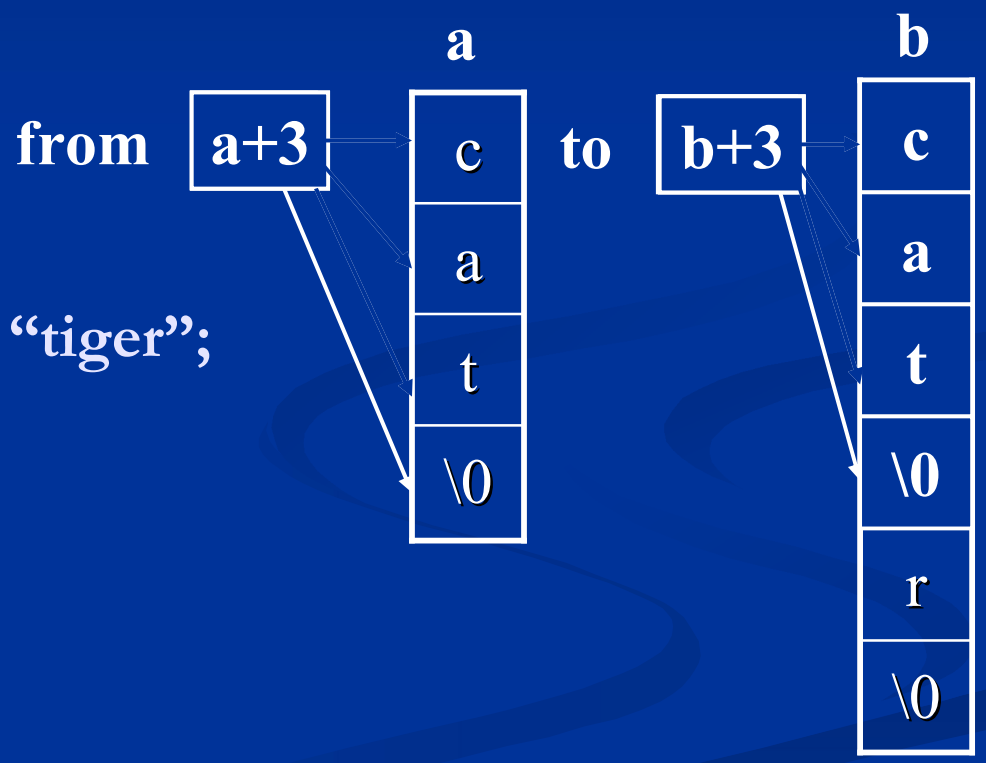
```
void   copystr(char  from[ ] , char  to[ ])
{   int   i = 0 ;
    while ( from[i] != '\0' )
    {    to[i] = from[i] ;
         i++;
    }
    to[i] = '\0';
}
int  main(void)
{
   char  a[ ] = "cat" ,  b[ ] = "tiger";
   puts(a);
   puts(b);
   copystr(a , b);
   puts(a);
   puts(b);
   return  0;
}
```

相当于b[i]=a[i]



例:用指向字符的指针作形参

```
void  copystr(char *from , char  *to)
{
     for(  ; *from!='\0' ; from++ , to++ )
          *to = *from ;
    *to = '\0';
}
int  main(void )
{
    char  *a = "cat" , b[6] = "tiger";
    puts(a);
    puts(b);
    copystr(a , b);
    puts(a);
    puts(b);
    return  0;
}
```



小结

- 一维数组的定义、初始化和引用
- 二维数组的定义、初始化和引用
- 用数组解决相关问题的算法
- 字符数组的定义、初始化和引用
- 常用字符串处理函数
- 数组与指针的使用方法
- 字符串与指针的使用方法


Class is over