

# 6 复杂数据的描述—构造数据类型

## 6.4 字符串

郭延文

2019级计算机科学与技术系

# 字符串

---

- ▶ 字符数组
  - ▶ 字符数组的定义和初始化
  - ▶ 字符数组的输入/输出
  - ▶ 字符数组作为函数参数
  - ▶ 用字符指针操纵字符数组
- ▶ 字符串常量的访问
- ▶ 常用字符串处理库函数
- ▶ 基于字符的信息检索
- ▶ 字符型指针数组与带形参的main函数\*\*

# 字符串

- ▶ 实际应用中，字符串是程序里常见的数据存在形式，通常用来表示文本数据。
- ▶ C语言本身没有提供字符串类型，一般通过构造基类型为char的数组类型，即字符数组来存储和处理字符串。
- ▶ 字符数组具有特殊性：通过引入结束符\0可以忽略数组长度的判断等。
- ▶ 一个字符串常量，比如“This is a C program.”（普通字符序列）或“Please input \ “Y\” or \ “N\” :”（含转义符的字符序列），可以复制到字符数组中加以处理，也可以用指针进行访问。

# 字符数组

---

## ▶ 字符数组的定义

```
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）。

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

- ▶ 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。

- 
- ▶ 如果是如下的不完全初始化形式:

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

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

- ▶ 系统会为str分配10个字节的内存空间，其前6个元素被初始化为'H'、'e'、'l'、'l'、'o'、'\0'，没有被初始化的元素均默认为'\0'。

- ▶ 如果是如下的初始化形式:

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

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

- ▶ 字符数组的长度省略后，系统为str只分配6个字节的内存空间

- ▶ 如果写成

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

- ▶ 则系统为str只分配5个字节的内存空间，不会自动存储'\0'

- 
- ▶ 如果是如下的不完全初始化形式:

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

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

- ▶ 系统会为str分配10个字节的内存空间，其前6个元素被初始化为'H'、'e'、'l'、'l'、'o'、'\0'，没有被初始化的元素均默认为'\0'。

- ▶ 如果是如下的初始化形式:

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

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

- ▶ 字符数组的长度省略后，系统为str只分配6个字节的内存空间

- ▶ 如果写成

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

- ▶ 则系统为str只分配5个字节的内存空间，不会自动存储'\0'

---

▶ ‘A’ 与 “A” 的区别：

▶ sizeof( ‘A’ )                      1

01000001
----------

▶ sizeof( “A” )                      2

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



# 字符数组的输入/输出

- ▶ 用 `cin>>`, `cout<<` (C++)

```
char* str;
```

```
cin>>str;
```

当键入的字符串为: Object\_Oriented Programming!

结果是: str指向的字符串为: Object\_Oriented

- ▶ 通过 `scanf/printf` 函数 (C), 借助格式符 `%s` 输入/输出 (一个 `%s` 对应一个字符数组名):

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

```
scanf( "%s" , str);      //输入一个字符串至str
```

```
printf( "The string is %s. \n" , str); //将str里的字符串输出
```

用 `cin/scanf` 的输入, 遇到空白字符为止!

# 字符数组的输入/输出: 用字符串相关的库函数!

---

- ▶ 用库函数gets/puts输入/输出一个字符串:

```
gets(str); / puts(str);
```

以回车符作为输入的结束标志, 也就是说回车符不会转存到str中, 不过gets可以读取空格符! (解决cin和scanf的空格后不能输入的问题)

```
getchar(str[i]); / putchar(str[i]);
```

单字符的输入和输出

---

## ▶ 对于字符数组

`char str[10];`

- ▶ 如果输入转存的字符个数 $n$ 小于9，则字符数组尚存在 $9-n$ 个多余内存单元，多余空间里的信号是乱码。
- ▶ 如果输入转存的字符个数 $n$ 大于9，则多余的 $n-9$ 个字符不仅会占用字符串结束符的存储空间，还会占用紧随字符数组之后的其他内存空间，从而造成内存冲突错误。

- 
- ▶ 如果输出没有结束符的字符串，则在字符串的后面会显示若干乱码（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("The string is %s. \n", str); //通常会显示Hello烫烫...
```

- ▶ 用格式符%s输出str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（内存中总有一些单元里的信号是0（‘\0’的ASCII码））。应防范出现乱码问题！

## 例：从键盘输入一个字符串，然后把该字符串逆向输出（反转）

```
#include <iostream>
```

```
#include <cstring>
```

```
using namespace std;
```

```
int main()
```

```
{ const int MAX_LEN=100;
```

```
char str[MAX_LEN];
```

```
gets(str);
```

```
int len = strlen(str); // str中的字符个数（\0之前）
```

```
for (int i=0,j=len-1; i<len/2; i++,j--) // 两个循环变量，也可以一个
```

```
{
```

```
    char temp;
```

```
    temp = str[i];
```

```
    str[i] = str[j];
```

```
    str[j] = temp;
```

```
}
```

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

```
return 0;
```

比如：“Program” 转换为 “margorP”。

对于for循环等，  
需要注意循环变量的边界条件  
(还记得排序的双重循环，  
循环变量的循环条件?)

```
}
```

## 例：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]; // 逐个输入字符
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("digit: %d; space: %d; other: %d \n", nDigit, nSpace, nOther);
```

## 例：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;  // 用字符串
printf("Enter string line\n");

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

printf("digit: %d; space: %d; other: %d \n", nDigit, nSpace, nOther);
```

## 例：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;
```

```
void Stat1(char string[])
```

```
{  
    int i = 0;  
    while(string[i] != '\0')  
    {  
        if((string[i]) >= '0' && (string[i]) <= '9') nDigit++;  
        else if((string[i]) == ' ') nSpace++;  
        else nOther++;  
        i++;  
    }  
}
```

```
int main()
```

```
{ char str[80];
```

```
    gets(str);
```

```
    Stat1(str);
```

```
    printf("digit: %d; space: %d; other: %d \n", nDigit, nSpace, nOther);
```

```
    return 0;
```

```
}
```

用函数的形式实现统计个数的功能



```
void Stat2(char string[], int *nDigit, int *nSpace, int *nOther)
```

```
{
```

```
    int i=0;
```

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

```
    {    if((string[i])>='0'&&(string[i])<='9') (*nDigit)++;
```

```
        else if((string[i])==' ') (*nSpace)++;
```

```
        else (*nOther)++;
```

```
        i++;
```

```
    }
```

```
}
```

```
int main()
```

```
{ char str[80];
```

```
    int dgt=0,spc=0,othr=0;
```

```
    gets(str);
```

```
    Stat2(str, &dgt, &spc, &othr);
```

```
    printf("digit: %d; space: %d; other: %d \n", dgt, spc, othr);
```

```
    return 0;
```

```
}
```

用字符串+函数的  
形式实现统计个数的  
功能

## 例：统计输入的字符行中各个数字、空格及其他字符的个数

```
int k, nSpace=0, nOther=0;
char ch;
int nDigit[10];           // 10个数字符号的计数器，类似于“标志位”
for(k = 0; k < 10; k++) nDigit[k] = 0;
printf("Enter string line\n");

while ((ch= getchar()) != '\n')
{
    if (ch >= '0' && ch <= '9')
        ++nDigit[ch - '0'];    // 对应计数器增1
    else if (ch == ' ')
        ++nSpace;
    else
        ++nOther;
}
for(k = 0; k < 10; k++)
    printf( "%d: %d \n", k, nDigit[k]);
printf("space:%d; other:%d \n", 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，通常需要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：字符串的大小写转换（小写转大写）

---

```
#include <stdio.h>
void to_upper(char s[]);
int main()
{   char str[10];
    printf("Please input a string: \n");
    gets(str);
    to_upper(str);
    printf("The new string is %s. \n", str);
}

void to_upper(char s[])
{   int i;
    for(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 OR: -32
    }
}
```

## 例：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

## 补充：字符串转化为数字的方法

---

```
char str[]="1234321";
```

```
int a;
```

```
sscanf(str,"%d",&a);
```

```
-----
```

```
char str[]="123.321";
```

```
double a;
```

```
sscanf(str,"%lf",&a);
```

```
-----
```

另外也可以使用atoi(),atol(),atof()等库函数



## 补充：数字转换为字符串的方法

---

```
char str[10];  
int a=1234321;  
sprintf(str,"%d",a);
```

-----

```
char str[10];  
double a=123.321;  
sprintf(str,"%0.3lf",a);
```

-----

```
char *itoa(int value, char* string, int radix);
```

同样也可以将数字转字符串，不过itoa()这个函数是平台相关的（不是标准里的）

---



# 总结1: C字符串三种输入/赋值方法

---

1. 声明字符串时直接赋值
2. 用scanf函数 (%s格式符)
3. strcpy函数给字符串赋值, 比较常用

char a[10] = "123"; 或

#include <string.h>

... ..

char a[10];

strcpy(a, "123");





# 关于字符串赋值

---

```
struct  
{  
    char name[20];  
    ...  
}s;
```

```
s.name = "Joe" ; // 错误
```

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

```
str = "Hello" ; // 错误
```

```
// error ...: "=" : 无法从 "const char [..]" 转换为 "char [..]"
```



## 总结2: C++字符串输入方法

---

### ➤ 使用cin输入字符串的相关问题

#### ➤ cin 使用空白（空格、制表符和换行符）来定字符串的界

这意味着cin在获取字符数组输入时只读取一个单词（遇到空格即止），在读取该单词后，cin将该字符串放到数组中，并自动在结尾添加空字符

### ➤ 用gets()库函数



# 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：从键盘输入一个字符串，然后把该字符串逆向输出（用指针实现）

▶ 比如：“Program” 转换为 “margorP”。

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

```
int main( )
```

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

```
  char *ph = str, *pt = ph;
```

```
  for( ; (*pt) != '\0'; pt++);
```

```
  for(pt--; ph < pt; ph++, pt--)
```

```
  {
```

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

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

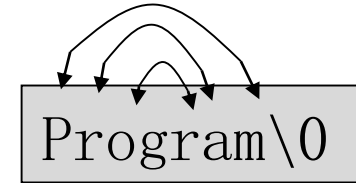
```
    *pt = temp;
```

```
  }
```

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

```
  return 0;
```

```
}
```



//ph指向第一个字符

//循环结束时，pt指向'\0'

//ph、pt相向移动，注意边界！

- 
- ▶ 对于字符指针，可以用%s格式符输出它所指向的字符串。

比如，

```
char *pstr, str[6] = "Hello";
```

```
pstr = str;
```

```
printf("The string is %s.\n", pstr); //显示Hello
```

- ▶ 也可以用其他格式符显示地址值，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
```

- 
- ▶ 如果要保存字符指针变量自身内存单元的地址，则需要定义一个字符型的二级指针变量，比如，

```
char *pstr, str[6] = "Hello";
```

```
pstr = str;
```

```
char **ppstr;
```

```
ppstr = &pstr;
```

- ▶ 用%s或%c格式符，可以输出二级指针变量所指向的字符串或某个单字符：

```
printf("%s \n", *ppstr);
```

```
//相当于printf("%s \n", pstr); 输出结果为Hello
```

```
printf("%c \n", **ppstr);
```

```
//相当于printf("%c \n", *pstr); 输出结果为H
```

# 字符串常量的初始化：采用指针（第二种方法）

## ▶ 第一种方法

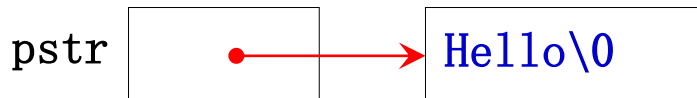
▶ `char astr[] = "Hello";`    `astr` Hello\0

*//astr在栈区占6个字节空间，存储该字符串常量的副本*

## ▶ 第二种方法

▶ `char *pstr = "Hello";`    Hello\0

或者



▶ `char *pstr;`  
`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*

```
char astr[] = "Hello";  
char *pstr = "Hello";
```

---

▶ 然后通过数组或指针访问字符串常量：

```
printf("The first character is %c \n", astr[0]); //访问第一个字符  
printf("The first character is %c \n", *pstr);    //访问第一个字符  
printf("The first character is %c \n", pstr[0]);  //访问第一个字符
```

```
printf("The first character is %c \n", astr[2]); //访问第三个字符  
printf("The third character is %c \n", *(pstr+2)); //访问第三个字符  
printf("The first character is %c \n", pstr[2]); //访问第三个字符
```

▶ 对于字符数组，其元素的值可修改：

▶ `astr[0] = 'h'`；

▶ 对于字符指针，不提倡通过指针变量修改字符串常量的值

▶ 比如一些编译器不允许 `(*pstr) = 'h'` 或 `pstr[0] = 'h'`；

▶ 通常会加一个 `const` 关键词，即 `const char *str = "Hello"`；  
以防止修改字符串常量中的字符。



# 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。

# C标准库中的字符串处理函数

(头文件 `cstring` 或 `string.h` )

---

## ▶ 计算字符串的长度

- ▶ `int strlen(const char s[]);`

## ▶ 字符串复制

- ▶ `char *strcpy(char dst[],const char src[]);`

- ▶ `char *strncpy(char dst[],const char src[],int n);`

## ▶ 字符串拼接

- ▶ `char *strcat(char dst[],const char src[]);`

- ▶ `char *strncat(char dst[],const char src[],int n);`

## ▶ 字符串比较

- ▶ `int strcmp(const char s1[],const char s2[]);`

- ▶ `int strncmp(const char s1[],const char s2[],int n);`

## ▶ 模式匹配

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

---

▶ ▶ ...

---

## (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（从特定位置开始）的个数，不包括 ‘\0’

- ▶ 比如，

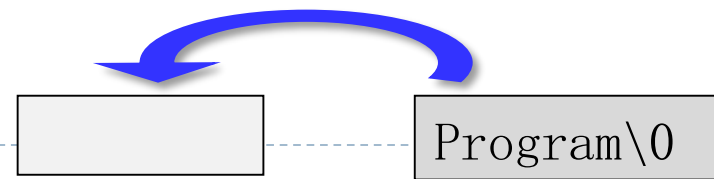
```
char str[] = "Hello" ;  
printf("%d\n", strlen(str));  
printf("%d\n", strlen(str+2));
```

5  
3

# 字符串反转（调用strlen求长度）

---

```
#include <string.h>
...
void strReverse(char *pstr)
{
    char *ph = pstr, *pt = pstr + strlen(pstr)-1;
    for(; ph < pt; ph++, pt--)
    {
        char temp = *ph;
        *ph = *pt;
        *pt = temp;
    }
}
```



## (2) 字符串复制

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

- ▶ 功能：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
- ▶ 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);`

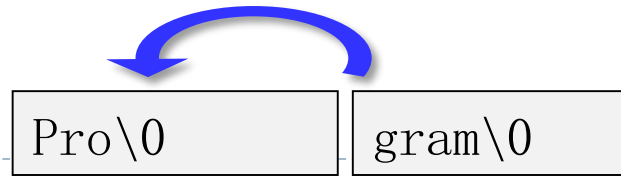
//库函数strcpy和strncpy的返回值均为s1的内存首地址

## 例：编程实现字符串拷贝函数

- ▶ 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。

```
void myStrcpy(char *t, char *s)
{
    while( (*s) != '\0' )
    {
        *t = *s;
        t++;
        s++;
    } //逐个字符复制
    *t = *s; //复制 '\0'
}

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



### (3) 字符串拼接

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

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

- ▶ 功能：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);`

---

- ▶ 功能：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。
- ▶ 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



0  
-20  
4

- ▶ `strncmp("student","study",4)`
- ▶ `strncmp("student","study",5)`
- ▶ `strncmp("program", "process", 5)`

$'e' - 'y' = -20$

$'g' - 'c' = 4$

## 例：判断用户输入密码的正确性（提供三次机会）

```
#include <stdio.h>
#include <string.h>
#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;
}
```

---

## (5) 模式匹配

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

- ▶ 求一个字符串在另一个字符串中首次出现的位置，即子串的查找操作，是一种模式匹配。
- ▶ 功能：从haystack中寻找needle第一次出现的位置，不比较结束符。返回指向第一次出现needle位置的指针，返回NULL表示没找到。

思考：如何通过该指针知道到底是第几个位置？

---

## (6) 其他操作

- ▶ 标准库中还有一些函数用于从字符串到数值类型的转换，它们的说明信息位于头文件stdlib.h中。

`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

## atof应用:

---

```
#include<stdlib.h>
```

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

```
int main()
```

```
{
```

```
    double d;
```

```
    char*str="12345.67";
```

```
    d=atof(str);
```

```
    printf( "string=%s, double=%lf\n",str,d);
```

```
    return 0;
```

```
}
```

```
#include<stdlib.h>
```

```
int main()
```

```
{
```

```
    char*a="-100.23";
```

```
    char*b="200e-2";
```

```
    double c;
```

```
    c=atof(a)+atof(b);
```

```
    printf( "c=%.2lf\n" ,c);
```

```
    return 0;
```

```
}
```

---



## 其他可应用于字符串处理的库函数

---

`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，比如  
`memset(str, '\0', sizeof(str));`

//将字符数组str的所有元素置为空字符

`void *memcpy(void *s1, void * s2, unsigned int n);`

- ▶ 可以实现字节数为n的一段内存的拷贝，n不要大于s1的元素个数，比如

`memcpy(dest, src, sizeof(dest));`

//将字符数组src的元素拷贝到数组dest中，

//dest、src是非空类型数据的地址

## strcpy V. S. memcpy

---

- ▶ strcpy和memcpy都是标准C库函数，它们有下面的特点
  - ▶ strcpy提供了字符串的复制。即strcpy只用于字符串复制，并且它不仅复制字符串内容之外，还会复制字符串的结束符。
  - ▶ 已知strcpy函数的原型是：`char* strcpy(char* dest, const char* src);`  
memcpy提供了一般内存的复制。即memcpy对于需要复制的内容没有限制，因此用途更广。
-

## 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 ss[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 ss[4][5] = {"Zhao", "Qian", "Sun", "Li"}; 或
```

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

```
char a[4][5];
```

```
char *ssp[4] = {a[0], a[1], a[2], a[3]}; 或
```

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



---

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

`// 注意与数组指针char(*ps)[4]的区别`

▶ 用字符型指针数组表示多个字符串不需要预先确定每个字符串的最大长度，比用二维字符型数组表示多个字符串更为方便！

## 举 例 (二维字符型数组) :

weekday	S u n d a y \0
weekday+1	M o n d a y \0

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

- 定义了一个7行10列的字符矩阵，并用7个字符串常量对其进行了初始化，其中：
- `weekday`是二维数组名，表示第一行的地址（假设其值为0x0065fde4，可用格式符%lx显示）；
- `weekday + 1`表示第二行的地址（假设其值为0x0065fdee（= 0x0065fde4 + 10））；
- `weekday[0]`表示第一行第一列元素的地址（其值为0x0065fde4），等价于一个一维字符型数组名，用格式符%s输出结果为Sunday，这是因为二维数组`weekday`可以看成是一个含7个元素（`weekday[0]~weekday[6]`）的特殊的一维数组；
- `weekday[0] + 1`表示第一行第二列元素的地址（其值为0x0065fde5），用%s输出结果为 ；
- `weekday[0][0]`表示第一行第一列的字符，用格式符%c输出结果为S；
- `weekday[0][0] + 1`表示将第一行第一列的字符ASCII码加1，用%c输出结果为 ；

## 举 例

### (字符型指针数组) :

week	0X2002
week+1	0X2012

0X2002	S u n d a y \0
0X2012	M o n d a y \0

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

- ▶ 定义了一个含7个元素的一维数组，每个元素都是一个字符型指针，并用7个字符串常量对其进行了初始化，其中：
- ▶ week是一维数组名，表示第一个元素week[0]的地址（假设其值为0x0065fdf0，可用格式符%lx显示）；
- ▶ week+1表示第二个元素week[1]的地址（假设其值为0x0065fdf4（=0x0065fdf0 + 4））；
- ▶ week[0]表示第一个元素，其值为一个字符型地址（假设其值为0x00002002），用%s输出结果为Sunday；
- ▶ week[0] + 1表示第一个元素的值加1（其值为0x00002003），仍然是一个字符型地址，用%s输出结果为 ；
- ▶ week[0][0]表示第一个元素所指向字符串的第一个字符，用%c输出结果为S；
- ▶ week[0][0] + 1表示将第一个元素所指向字符串的第一个字符ASCII码加1，用%c输出结果为 ；

# 例：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 <stdio.h>
```

```
#include <string.h>
```

```
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;
```

```
    }
```

```
}
```

用选择排序法

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

# 基于字符的信息检索程序

---

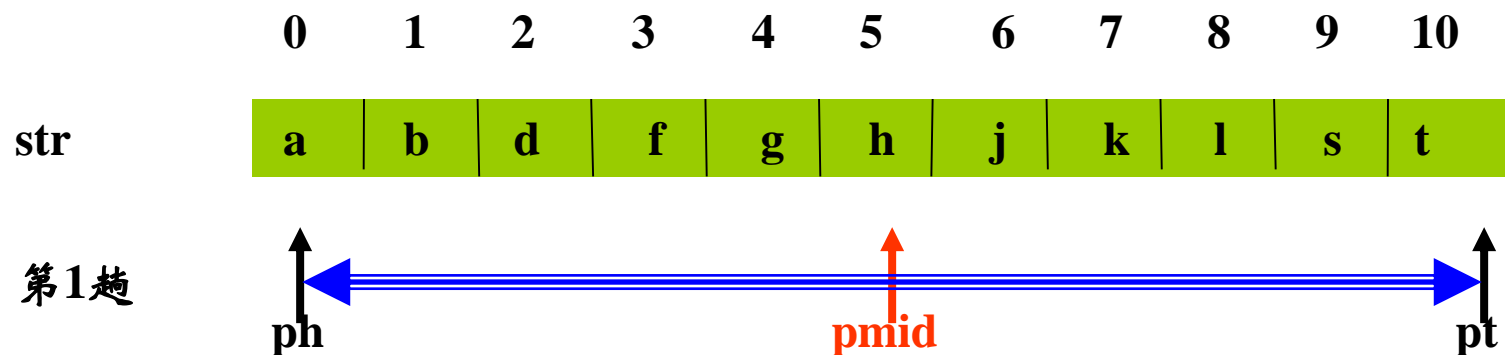
- ▶ 信息检索问题是一种常见的非数值计算问题。
- ▶ 本节以基于字符数组的字符查找为例，介绍简单的信息检索方法。
- ▶ 常用的数据查找算法有顺序查找、折半查找等。

# 查 找

I am the King!

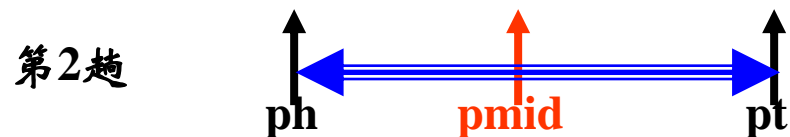
- ▶ 假设字符数组st中的字符已按从小到大的顺序排列好，现在从键盘上输入一个字符key，用折半法在str中查找此字符：找到，输出它在数组中的位置；否则给出相应提示。
- ▶ 分析：采用折半法(二分法)查找（要求待查数据排列有序）先确定待查数据范围(区间)，然后逐步缩小范围，直到搜索完为止。

设key='d', 则查找过程如下:



key < str[pmid], key 应在左半部分

---



str[pmid] == key

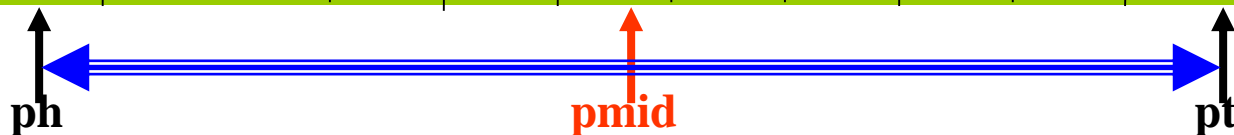
找到

设key='k'，则查找过程如下：

0    1    2    3    4    5    6    7    8    9    10

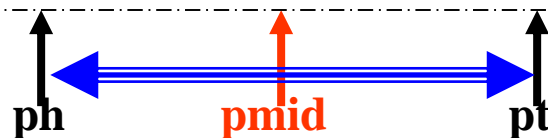
str    a    b    d    f    g    h    j    k    l    s    t

第1趟



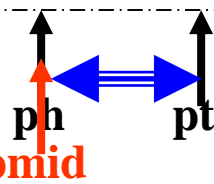
$\text{str}[\text{pmid}] < \text{key}$ ，key应在右半部分

第2趟



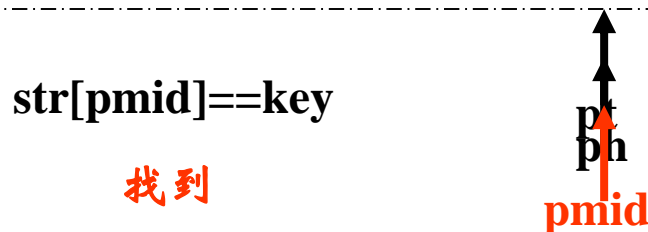
$\text{key} < \text{str}[\text{pmid}]$ ，key应在左半部分

第3趟



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

第4趟



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

找到



设key='i'，则查找过程如下：

0    1    2    3    4    5    6    7    8    9    10

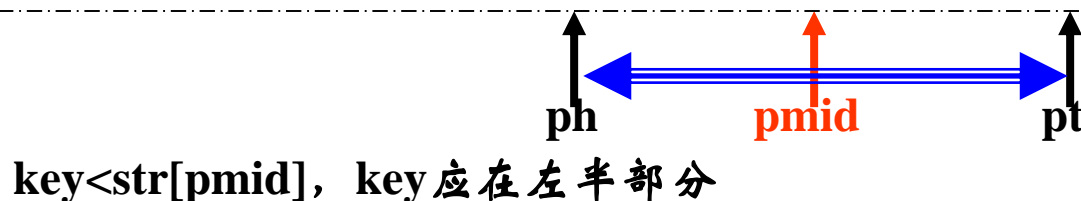
str    

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

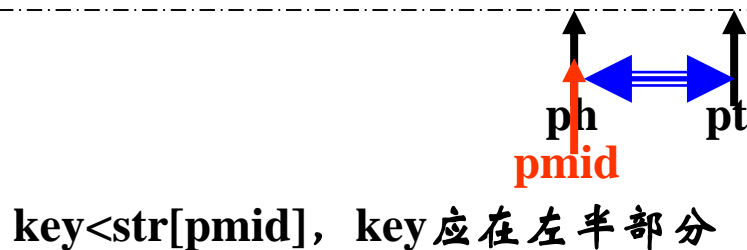
第1趟



第2趟



第3趟



第4趟



# 算法设计

---

- ▶ 计算待查区间的中间元素下标 $\text{pmid}$ ;
- ▶ 将 $\text{key}$ 与 $\text{str}[\text{pmid}]$ 比较, 得到三种结果并分别处理:

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

整个区域查找完毕, 没查到

```
int main()  
{ char key, str[ ] = "abcdefghijklmnopqrst";  
  int ph=0, pt, pmid, flag=0;
```

```
    return 0;
```

```
}
```



```
int main()
{ char key, str[ ] = "abcdefghijklmnopqrst";
  int ph=0, pt, pmid, flag=0;

  printf("please input a letter:");
  scanf("%c", &key);
  pt = strlen(str)-1;

  return 0;
}
```

```
int main()
{ char key, str[ ] = "abcdefghijklmnopqrst";
  int ph=0, pt, pmid, flag=0;

  printf("please input a letter:");
  scanf("%c", &key);
  pt = strlen(str)-1;

  while(ph <= pt)
  {
      .....
  }
  if(flag==1) cout << pmid << endl;

}
```

```
int main()
{ char key, str[ ] = "abcdefghijklmnopqrst";
  int ph=0, pt, pmid, flag=0;
```

```
  printf("please input a letter:");
  scanf("%c", &key);
  pt = strlen(str)-1;
```

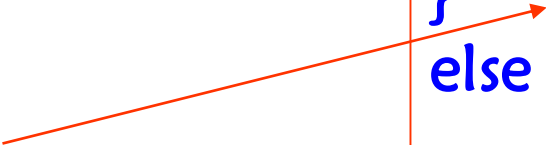
```
  while(ph <= pt)
  {
      .....
  }
```

```
  if(flag == 1) printf("%d \n", pmid);
  else printf("\n not found \n");
```

```
  return 0;
```

```
}
```

```
    pmid = (ph+pt)/2;
    if (key == str[pmid])
    {        flag = 1;
              break;
    }
    else if (key > str[pmid])
              ph = pmid+1;
    else
              pt = pmid-1;
```



# 写成独立的函数

```
int main()
{  char key,str[ ] = "abcdefghijklmnopqrst";
   printf("please input a letter:");
   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])
        else if(k < x[pmid])

    }
    else           return -1;
}
```



# 二分法（折半法）查找字符串

```
cp = Bin(temp, "sun", 4, &cn);
```

```
char *Bin(
{
    int left, right, mid;
    left = 0;
    right = n-1;
    while(left <= right)
    {
        ...
    }
    return 0;
}

mid = (left+right)/2;
if (strcmp(str, sp[mid]) < 0)
    right = mid-1;
else if(strcmp(str, sp[mid])>0)
    left = mid+1;
else
{
    *addr = mid;
    return(sp[mid]);
}
```

通过参数传递值只能实参→形参

而通过指针指向实参，则可修改实参

---

```
char *Bin(char *sp[ ], char *str, int n, int *addr)
```

```
int main( )
{
    int cn;
    char *cp;
    char *temp[] = {"li", "qian", "sun", "zhao"};
    cp = Bin(temp, "sun", 4, &cn);
    if(cp != 0)
        printf("%d\n", cn+1);
    return 0;
}
```

# 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

# 带形参的main函数\*\*

► 实际上，定义main函数时可以定义参数，比如，

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

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

```
{
```

```
    while(argc > 1)
```

```
    {
```

```
        ++argv;
```

```
        printf("%s \n", *argv);
```

```
        --argc;
```

```
    } //从第二个元素开始
```

```
    //分行输出所有元素的值
```

```
    return 0;
```

```
}
```

假设该代码存储在程序文件echo.c中，  
操作系统按如下命令执行该程序：

c:\>echo China Nanjing

即命令中包括三个字符串，  
于是形参argc自动获得字符串的个数3，  
形参argv是一个字符型指针数组，  
每个元素获得一个字符串，  
argv[0]获得“echo”，  
argv[1]获得“China”，  
argv[2]获得“Nanjing”。

程序执行结果为：

China

Nanjing

- 
- ▶ 再比如，一个文件拷贝程序`copy.c`，可以按“`copy file1 file2`”的命令形式来执行文件`file2`至文件`file1`的拷贝。
  - ▶ 一般情况下，程序不需要调用者（比如操作系统）提供参数，定义`main`函数时不用定义形参，如果程序需要用到调用者提供的参数，则可以在定义`main`函数时给出形参的定义。

# 小 结

---

## ▶ 字符串：

- ▶ 用字符数组或字符指针来表示和处理
- ▶ 与一般的数组和指针类型数据相比，字符型数组和指针具有特殊性

## ▶ 要求：

- ▶ 掌握字符数组和字符指针的定义、初始化和操作方法
- ▶ 能够自行实现常用字符串处理函数
- ▶ 掌握基本的字符串库函数用法

## Q & A

---



# 内容回顾

## ▶ 指针及其应用-1

### ▶ 指针的基本概念

- ▶ 指针类型的构造
- ▶ 指针变量的定义与初始化

```
int *  
float *  
double *  
char *
```

```
int *pi = &i;  
float *pf = &f;  
double *px = &x;  
char *pch = &ch;
```

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

### ▶ 指针数组

```
int ** pp = &pi;
```

### ▶ 多级指针变量

```
void *
```

```
void *pv = 0;
```

### ▶ 通用指针与void类型

### ▶ 指针类型相关的基本操作

\* (与&“互逆”)

=

> < >= <=

== !=

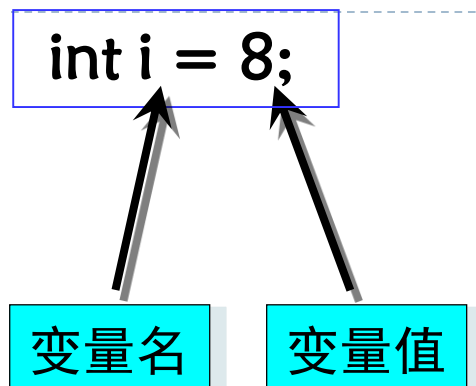
++ --

-



# 内容回顾

## “举个形象的例子”



```
int i, *p;
```

```
p = &i;
```

```
i = 南京大学;
```

```
// p 的值: 南京市栖霞区仙林大道163号
```

```
// *p 的值: 南京大学
```

```
*p = 国立中央大学; // i 也变为了国立中央大学
```

```
p++; // 在 p 的值基础上加其指向的基类型的字节数
```

# 内容回顾

## ▶ 指针及其应用-2

### ▶ 用指针操纵数组

### ▶ 用指针在函数间传递数据

#### ▶ 传址调用

#### ▶ const

#### ▶ 指针型函数

### ▶ 用指针访问动态变量

#### ▶ 动态变量的创建、访问与撤销

#### ▶ 内存泄露与悬空指针\*\*

### ▶ 用指针操纵函数\*\*

```
int a[10];  
int b[5][10];  
int * p = a;  
int (*q)[10] = &a;  
int (*r)[5][10] = &b;
```

```
int * ap[3] = {&i, &j, &k};  
// 注意ap与q的区别
```

```
Fun(a, ...);  
void Fun(int *pa)  
{ *pa...pa++...}
```

```
Fun(&n);  
void Fun(int *pn)  
{ *pn = ...}
```

```
void Fun(const int *pa)  
{ *pa...pa++...} ×  
*pa = ...
```

```
int * Fun(...)  
{...return &...}
```

# 内容回顾

## 指针操纵二维数组，显示其元素

---

```
#include<iostream>
using namespace std;
#define M 3
#define N 3
int main() {
    int a[M][N] = {1,2,3,4,5,6,7,8,9}; //待打印数组
    int (*pp)[N]; //按行循环指针。二级指针pp应当定义为长度为N的指针数组，
    //这样pp++会移动sizeof(int)*N,即一行的长度,如果定义为
    //int** pp,则pp++的时候移动的是一个指针的长度(32位机上4个字节),
    int *p; //每次移动一个数，定义为一维指针
    for(pp = a; pp < a+M; pp++){ //外层循环是按行移动，应当用一个二级
    //指针进行移动，
        for(p = *pp; p < *pp+N; p++){ //内层循环在每一行内按列移动
            cout << *p<<" "; //打印输出
        }
        cout << endl;//打印换行符
    }
    return 0;
}
```

---

# 内容回顾

## “理解两点”

---

▶ 当定义了一个指针后：

▶ \* 和 & “互反”

▶ 对于数组元素取地址

▶ & 和 [0] “互抵消”

▶ 例如：

```
double *p, a[10], b[10][10];
```

```
p = a; // p=&a[0];
```

```
// *p 即是a[0]
```

```
p= b[0]; // p = &b[0][0];
```

```
double (*q)[10]; // 可存放一个长度为10的数组的地址
```

```
q= b; // q = &b[0];
```

---



# 内容回顾

## ▶ 指针及其应用-2

- ▶ 用指针操纵数组
- ▶ 用指针在函数间传递数据

- ▶ 传址调用

- ▶ const

- ▶ 指针型函数

- ▶ 用指针访问动态变量

- ▶ 动态变量的创建、访问与撤销

- ▶ 内存泄露与悬空指针\*\*

- ▶ 用指针操纵函数\*\*

```
int * Fun(...)  
{...return &...}
```

//注意Fun与pfun1的区别

```
int n; ...  
int q = new int [n];  
delete []q;
```

q自身消亡或指向了别处，但q指向的堆区未撤销；  
q自身未消亡，也未指向别处，而q指向的堆区已撤销。

```
int (*pfun1) () = &Fun1;  
double (*pfun2)(int) = &Fun2;
```

# 指针内容小结

---

## ▶ 指针：

- ▶ 一种派生数据类型，
- ▶ 地址是一种特殊的整数，将地址专门用指针类型来描述，可以限制该类型数据的操作集，从而得以保护数据。

## ▶ 要求：

- ▶ 掌握指针的定义、初始化和操作方法
- ▶ 掌握指针类型的特征及其典型应用场合
  - ▶ 可以在函数间高效地传递数据
  - ▶ 用指针操纵数组
  - ▶ 可以用来操作动态数据。
    - 动态变量和动态数组需要在程序中创建与撤销，使用过程中要注意避免内存泄露和悬浮指针等问题。
  - ▶ 此外，本章还介绍了用指针操纵函数的方法，以及指针数组、多级指针、void类型、const类型等概念
- ▶ 继续保持良好的编程习惯

# 内容回顾: 字符串

## ▶ 通常以字符数组来存储和处理字符串

- ▶ 以 ‘\0’ 为结束标志!

## ▶ 输入输出函数: gets(str), puts(str)

- ▶ 输出还可以用C: printf("%s", str);

- ▶ C++: cout<<str;

## ▶ 字符串常量初始化

- ▶ char str[] = "Hello";

- ▶ char \*pstr = "Hello";

用字符型指针表示多个字符  
不需要预先确定字符串的最  
大长度, 比用字符型数组表  
示字符串更为方便!

## ▶ 字符数组作为函数参数

- ▶ 由于 ‘\0’, 形参可省去表示数组长度的参数

# 内容回顾

---

- ▶ 例题：字符串反转
- ▶ 例题：统计输入的字符行中各个数字、空格及其他字符的个数
- ▶ 例题：字符串的大小写转换（小写转大写）

`str[i] = str[i] - 'a' + 'A';`

- ▶ 例题：数字字符串到整数的转换: `int StrToInt(char *str);`

`int i=str[i]-'0'; // 或str[i]-48`

- ▶ 例题：整数转化为字符串: `char *IntToStr(int num, char str[])`

`str[i]=i+'0'`

---



# 内容回顾

例：从键盘输入一个字符串，然后把该字符串逆向输出（反转）

```
#include <iostream>
```

```
#include <cstring>
```

```
using namespace std;
```

```
int main()
```

```
{ const int MAX_LEN=100;
```

```
char str[MAX_LEN];
```

```
gets(str);
```

```
int len = strlen(str); // str中的字符个数（\0之前）
```

```
for (int i=0,j=len-1; i<len/2; i++,j--) // 两个循环变量，也可以一个
```

```
{
```

```
    char temp;
```

```
    temp = str[i];
```

```
    str[i] = str[j];
```

```
    str[j] = temp;
```

```
}
```

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

```
return 0;
```

比如：“Program” 转换为 “margorP”。

对于for循环等，  
需要注意循环变量的边界条件  
(还记得排序的双重循环，  
循环变量的循环条件?)

```
}
```

## 思考

---

1. 下列那一项是C/C++语言中的合法字面常量，请选择 ( A )。

A) 3e-8    B) 0239    C) A    D) 'abc'

2. 如何表示 abc ?

如何表示 I am the king ?

