



清华大学

腾讯会议 ID: 638 164 433

*Tsinghua University*

# 计算机程序设计基础

## 第13讲 字符串

沈瑜 (010-62782951)

shenyu@tsinghua.edu.cn

清华大学电机系

2022.12.6





# 主要内容

- 若干知识点
- 字符串输入输出及转换
- 例子：成语接龙
- 强调：程序调试思路

参考教材：





## 知识点1

### • 字符串常量

- 字符串常量：一对双引号括起来的字符序列。
  - “Hello,World”
- **C**在每个字符串结尾处加上字符‘\0’
  - 作为字符串结束标志
  - 作用：传递参数时，不用指明字符串长度
- 举例
  - “a” 包括两个字符：‘a’, ‘\0’





## 知识点2

### • 字符串存储

- 字符串采用字符数组来存储，以字符'\0'作为结束标志。因此，字符串是一种特殊的字符数组。
- strlen求字符串长度时不包括'\0'.

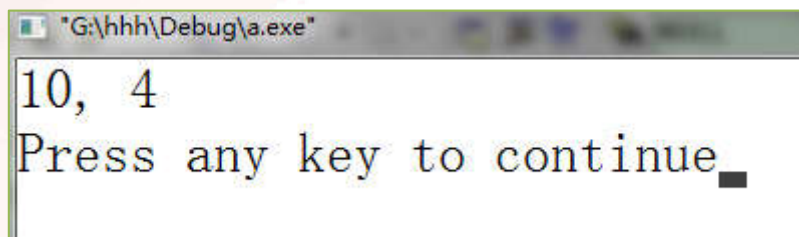




# 例

```
void main() {  
    char a[] = "587691234";  
    char *a2 = "587691234";  
    printf("%d, %d\n", sizeof(a), sizeof(a2));  
}
```

例1： 程序输出是？



"G:\hhh\Debug\a.exe"  
10, 4  
Press any key to continue.





# 例

```
char digit_to_hex_char(int digit)
{
    return "0123456789ABCDEF"[digit];
}
```





## 知识点3

### • 汉字字符

- 一个汉字，2个字符
- 汉字数目很多，需要用两个字节才能表示

```
char *p  = "一马当先";  
char *p2 = "abcdefgh";
```

**Visual Studio**对未初始化的内存的赋值为**0xCC**，  
两个字节**0xCCCC**恰好是“烫”字





## 13.1 字符串输入输出及转换

### 1. 相关函数

作用	函数
字符串输入	<b>scanf, fscanf, gets, fgets</b>
字符串输出	<b>printf, fprintf, puts, fputs</b>
从字符串转换	<b>sscanf, atoi, atof</b>
转换为字符串	<b>sprintf</b>

**scanf** 系列，以读到**空格**视为字符串结束

**gets**系列，以读到**回车符**视为字符串结束





## 2.从字符串转换

### 使用sscanf 函数

		fscanf	sscanf
作用类似		将字符串转换成其他类型数据后输入	
用法类似		<b>fscanf( f1, “ %d,%d”, &amp;k1, &amp;k2);</b> <b>sscanf( s1, “ %d,%d”, &amp;k1,&amp;k2);</b>	
区别	参数类型	第一个参数是文件指针	第一个参数是字符指针
	作用	从文件读入	从字符数组读入

**File \* f1;**

**char s1[100] = “6,7”;**

- 更加简单的函数
  - **atof, atoi**系列
  - **stdlib.h**

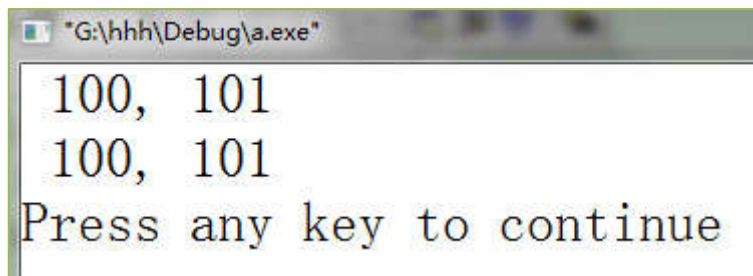
作用: 将字符串转换为浮点数, 整数  
使用举例:

```
char temp[100] = "123.4";  
float fTemp = atof(temp);
```



### 例3: sscanf, fscanf 用法对比

```
void main() {  
    char p[100] = "100, 101";  
    int i=0, j=0;  
    sscanf(p, "%d,%d", &i, &j);  
    printf(" %d, %d\n", i, j);  
  
    FILE *f = fopen("d:\\s.dat", "w+");  
    fprintf(f, "%s", p);  
    rewind(f);  
    fscanf(f, "%d,%d", &i, &j);  
    fclose(f);  
    printf(" %d, %d\n", i, j);  
}
```



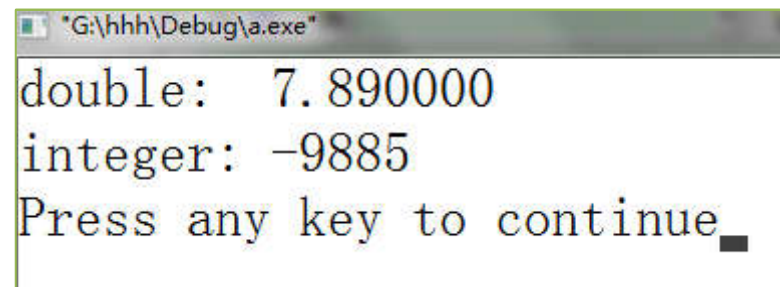
```
G:\hhh\Debug\a.exe  
100, 101  
100, 101  
Press any key to continue
```

## 例4: atof,atoi 举例

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

void main( void ) {
    char *s = "7.89";
    double x = atof( s );
    printf( "double: %lf\n", x );

    s = " -9885 pigs";
    int i = atoi( s );
    printf( "integer: %d\n", i );
}
```



```
"G:\hhh\Debug\a.exe"
double: 7.890000
integer: -9885
Press any key to continue_
```

### 3. 转换为字符串

#### 使用**sprintf** 函数

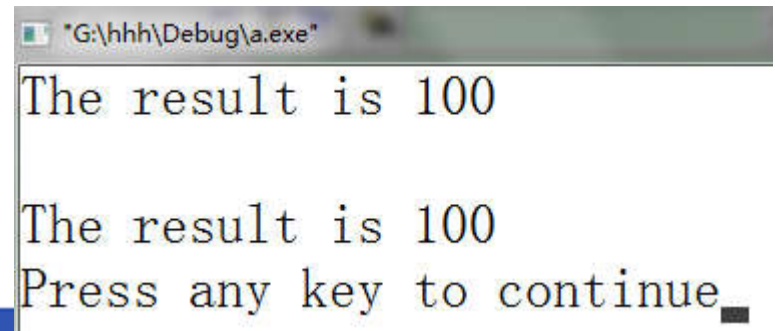
		<b>fprintf</b>	<b>sprintf</b>
作用类似		将其他类型数据转换成字符串格式后输出	
用法类似		<b>fprintf( f1, “ %d,%d”, k1,k2);</b> <b>sprintf( s1, “ %d,%d”, k1,k2);</b>	
区别	参数类型	第一个参数是文件指针	第一个参数是字符指针
	作用	输出到文件	输出到字符数组



## 例6: sprintf, fprintf 使用对比

```
#include <stdlib.h> //函数system
void main() {
    char p[100];
    int i=100;
    sprintf(p, "The result is %d\n",i);
    puts(p);

    FILE *f = fopen("d:\\s.dat","w");
    fprintf(f, "The result is %d\n",i);
    fclose(f);
    system(" type d:\\s.dat");
}
```



```
"G:\hhh\Debug\a.exe"
The result is 100

The result is 100
Press any key to continue.
```



## 4.字符串操作1——最常用

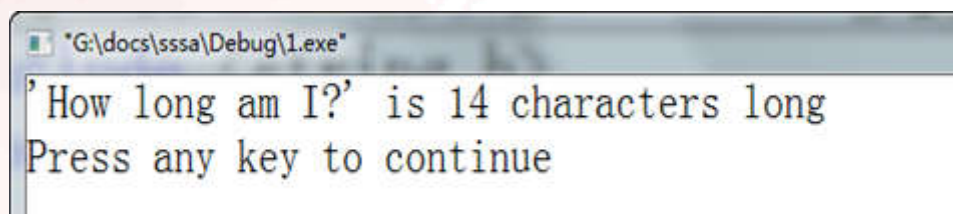
作用	函数	备注
字符串长度	<b>strlen</b>	<b>String length</b>
字符串拷贝	<b>strcpy, strncpy</b>	<b>String copy</b>
字符串拼接	<b>strcat, strncat</b>	<b>string catenate</b>
字符串比较	<b>strcmp, strncmp</b> <b>stricmp, strnicmp</b>	<b>String compare</b> <b>n</b> ,number, 指明比较字符个数 <b>i</b> ,指明不区分字母大小写 <b>case insensitive</b>
大小写转换	<b>strlwr,strupr</b>	<b>Lower,upper</b>



## 例1: strlen

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

void main( void )
{
    char buffer[61] = "How long am I?";
    int len;
    len = strlen( buffer );
    printf( "'%s' is %d characters long\n",
            buffer, len );
}
```



A screenshot of a Windows command prompt window. The title bar shows the file path "G:\docs\sssa\Debug\1.exe". The window contains the output of the program: "'How long am I?' is 14 characters long" followed by a prompt "Press any key to continue".

## 例2: strcat strcpy

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

void main( void )
{
    char string[80];
    strcpy( string, "Hello world from " );
    strcat( string, "strcpy " );
    strcat( string, "and " );
    strcat( string, "strcat!" );
    printf( "String = %s\n", string );
}
```

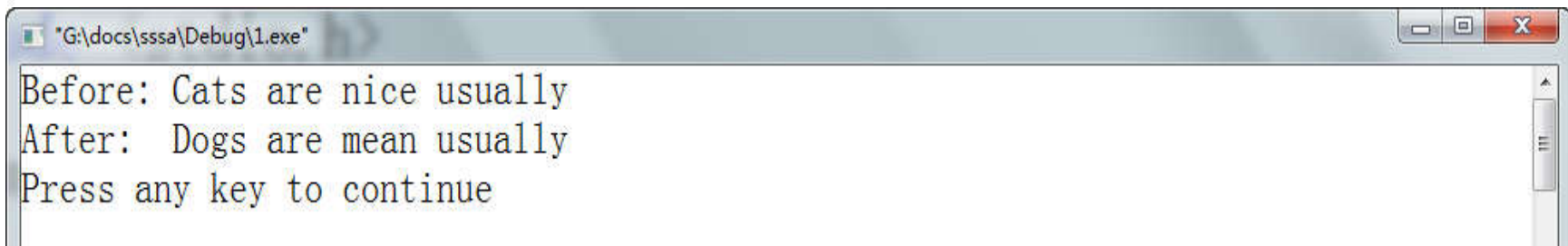
"G:\hhh\Debug\1.exe"

String = Hello world from strcpy and strcat!  
Press any key to continue

### 例3: **strncpy**

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

void main( void )
{
    char string[100] = "Cats are nice usually";
    printf ( "Before: %s\n", string );
    strncpy( string, "Dogs", 4 );
    strncpy( string + 9, "mean", 4 );
    printf ( "After:  %s\n", string );
}
```

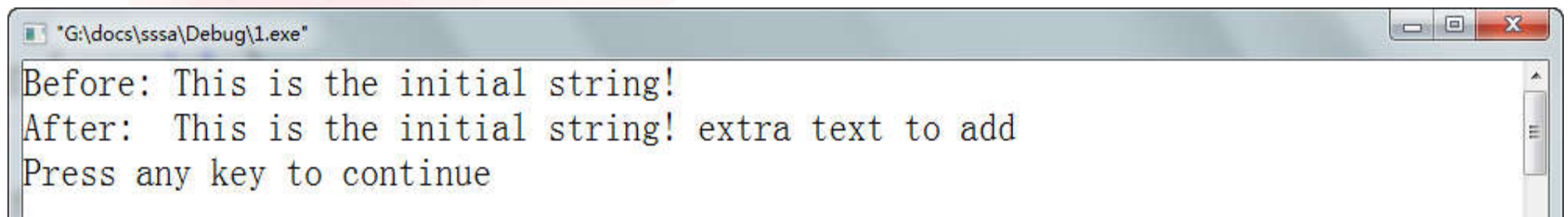


```
"G:\docs\sssa\Debug\1.exe"
Before: Cats are nice usually
After: Dogs are mean usually
Press any key to continue
```

## 例4: **strncat**

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

void main( void ) {
    char string[80] = "This is the initial string!";
    char suffix[] = " extra text to add to the string...";
    /* Combine strings with no more than
       |19 characters of suffix: */
    printf( "Before: %s\n", string );
    strncat( string, suffix, 19 );
    printf( "After:  %s\n", string );
}
```

A screenshot of a Windows command prompt window titled "G:\docs\sssa\Debug\1.exe". The window displays the output of the program: "Before: This is the initial string!" followed by "After: This is the initial string! extra text to add" on the next line. Below the output, it says "Press any key to continue". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
"G:\docs\sssa\Debug\1.exe"
Before: This is the initial string!
After: This is the initial string! extra text to add
Press any key to continue
```

## 例5: strcmp strnicmp

```
char string1[] = "The quick brown dog jumps over the lazy fox";
char string2[] = "The QUICK brown fox jumps over the lazy dog";

void main( void )
{
    char tmp[20];
    int result;
    printf( "Compare strings:\n\t\t%s\n\t\t%s\n\n", string1, string2 );
    printf( "Function:\tstrcmp (first 10 characters only)\n" );
    result = strcmp( string1, string2, 10 );
    if( result > 0 )
        strcpy( tmp, "greater than" );
    else if( result < 0 )
        strcpy( tmp, "less than" );
    else
        strcpy( tmp, "equal to" );
    printf( "Result:\t\tString 1 is %s string 2\n\n", tmp );
    printf( "Function:\tstrnicmp _strnicmp (first 10 characters only)\n" );
    result = _strnicmp( string1, string2, 10 );
    if( result > 0 )
        strcpy( tmp, "greater than" );
    else if( result < 0 )
        strcpy( tmp, "less than" );
    else
        strcpy( tmp, "equal to" );
    printf( "Result:\t\tString 1 is %s string 2\n\n", tmp );
}
```

```
C:\WINDOWS\system32\cmd.exe

Compare strings:
        The quick brown dog jumps over the lazy fox
        The QUICK brown fox jumps over the lazy dog

Function:      strcmp (first 10 characters only)
Result:        String 1 is greater than string 2

Function:      strnicmp _strnicmp (first 10 characters only)
Result:        String 1 is equal to string 2
```





## 5.字符串操作2——较常用

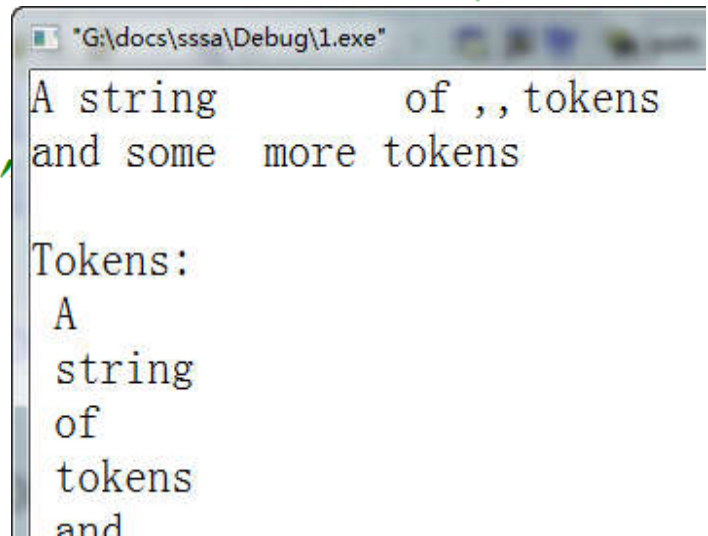
作用	函数	备注
字符串 <b>找</b> 子字符串	<b>strstr( const char *string, const char *strCharSet )</b>	找到 <b>strCharSet</b> 在 <b>string</b> 中的位置，返回为指针
字符串中 <b>找</b> 指定字符	<b>strchr(const char *string, int c ), strrchr</b>	找到 <b>c</b> 在 <b>string</b> 中的位置，返回为指针。 <b>r , right</b> ，从右边找
字符串中 <b>找</b> 指定字符集中的字符	<b>strcspn( const char *s1, const char *s2 );</b>	返回 <b>s1</b> 中第一个 <b>在</b> <b>s2</b> 中的字符的位置
字符串中 <b>找</b> 不在指定字符集中的字符	<b>strspn (const char *s1,const char * s2)</b>	返回 <b>s1</b> 中第一个 <b>不在</b> <b>s2</b> 中的字符位置
字符串分解	<b>strtok</b>	



## 例6: strtok

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

void main( void ) {
    char string[] = "A string\t of ,, tokens\nand some \n\nmore tokens";
    char seps[] = " , \t\n";
    char *token;
    printf( "%s\n\nTokens:\n", string );
    /* Establish string and get the first token: */
    token = strtok( string, seps );
    while( token != NULL ) {
        /* While there are tokens in
        printf( " %s\n", token );
        /* Get next token: */
        token = strtok( NULL, seps );
    }
}
```



"G:\docs\sssa\Debug\1.exe"

A string of ,, tokens  
and some more tokens

Tokens:  
A  
string  
of  
tokens  
and



## 13.2 例子：成语接龙

- 示例：  
一马当先  
先发制人  
人山人海
- 成语数据库 如何存储？
- 完成该任务，需要编写哪些函数？



# 数据如何存储？

- 成语接龙.txt  
空格分开

思路一：存放成语的数组，每格元素均为一个成语的字符数组，遍历查找成语的首个汉字；

思路二：所有成语，读入到一个大的字符数组中，对其进行查找、判断；



## 总的思路：步骤分解

- 第1步：提示用户输入并读入成语
- 第2步：判读用户输入成语是否正确
  - 看成语是否在成语库中
  - 如果不在库中，打印信息，程序终止
- 第3步：找到成语结尾汉字
- 第4步：在文本文件中找到以该汉字开头的成语
  - 如果找不到，打印信息，程序终止
- 第5步：输出成语，并返回第1步



- 第1步：提示用户输入并读入成语
  - `void InputIdiom(char *input, int len);`
- 第2步：判读用户输入成语是否正确
  - 如果不对，打印信息，程序终止
  - `int IsBeginingWith(char *input, char *beginningWord) ;`
- 第3步：找到成语结尾汉字
  - `void GetEndingWord(char *input, char *endingWord, int len);`
- 第4步：找到以该汉字开头的成语
  - 如果找不到，打印信息，程序终止
  - `int findIdiom_BeginWithWord(char *chArray, char *searchingWord, char *idiom, int len);`





## Task 1.1

- 读入文本文件“成语接龙.txt”中的所有字符
- `int readTxt(char *fileName, char *chArray, int len)`
- 方法：
  - 打开文件
  - 逐个字符读入到字符数组，直到碰到**EOF**
  - 关闭文件



//1 读入给定文本文件中的所有字符，并存入字符数组中

```
int readTxt(char *fileName, char *chArray, int len)
{
    FILE *fp = fopen(fileName, "rt");
    assert(fp);
    int index;
    for (index = 0; index < len-1; index++) {
        chArray[index] = getc(fp);
        if (chArray[index] == EOF) break;
    }
    chArray[index] = '\0';
    fclose(fp);
    return index;
}
```

```
void main(void) {  
    char chArray[LEN_FILE];  
    int count = readTxt("成语接龙.txt",chArray,LEN_FILE);  
    printf("input %d chars\n",count);  
}
```

- 运行程序



## Task 1.2

- 找到以给定汉字（如"竹"）开头的成语
  - 返回值表示是否找到满足条件的成语
  - 若找到，返回**1**；否则，返回**0**
- **int findIdiom\_BeginWithWord(char \*chArray, char \*theWord, char \*idiom, int len);**
- 理解题意，并根据题意写出测试



```
void test_findIdiom_BeginWithWord() {  
    char *pArray = " 胸有成竹 + 竹报平安 + 安富尊荣 + 荣华富贵 ";  
    char idiom[LEN_IDIOM];  
    int nResult = findIdiom_BeginWithWord(pArray,"贵",  
        idiom,LEN_IDIOM);  
    assert(nResult==0);  
  
    nResult = findIdiom_BeginWithWord(pArray,"安",  
        idiom,LEN_IDIOM);  
    assert(nResult==1);  
    printf("find %s\n",idiom);  
}
```

- 1 运行程序
- 2 需要编写程序 **findIdiom\_BeginWithWord** 使测试通过



## 编写思路

- 对" 胸有成竹 + 竹报平安 + 安富尊荣 + 荣华富贵 "
- 用**findIdiom**找到含“安”的成语
- 若没找到，程序结束
- 否则，看这个成语是否以“安”开头
  - 如果是，则结束
  - 如果不是，则继续找下一个





//找到以给定汉字（如“竹”）开头的成语

//返回值表示是否找到满足条件的成语：若找到，返回1；否则，返回0.

```
int findIdiom_BeginWithWord(char *chArray, char *searchingWord, char *idiom, int len)
{
```

```
    while ( chArray = findIdiom(chArray, searchingWord, idiom, len) )
```

```
        if ( strncmp(idiom, searchingWord, //chArray指针变量的值逐步后移
```

```
            strlen(searchingWord)) == 0 ) // 找到
```

```
            return 1;
```

```
    return 0;
```

```
}
```

//找到第一个包含给定汉字（如“竹”）的成语

//返回值为指向chArray中被找到的成语的下一个字符的指针

// 如未找满足要求的成语到则返回NULL

```
char* findIdiom(char *chArray, char *theWord, char *idiom, int len)
```

```
{
```

```
    int i, j;
```

```
    char *pos = strstr(chArray, theWord);
```

```
    if (pos==NULL) return NULL;
```

```
    int n = pos - chArray;
```

//寻找成语的开头. 注意：成语须以' ' 开头

```
    for(i=n-1; i>=0; i--)
```

```
        if ( chArray[i] == ' ' ) break;
```

// 将找到的成语存入数组idiom. 注意：成语须以' ' 或 '\n' 结尾

```
    for(j=0; j<len-1; j++)
```

```
    {
```

```
        if ( chArray[i+1+j]==' ' || chArray[i+1+j]=='\n' ) break;
```

```
        idiom[j] = chArray[i+1+j]; //复制了查到的成语，存放在idiom
```

```
    }
```

// 结尾加上空字符

```
    idiom[j] = '\0';
```

```
    return chArray+i+j;
```

```
}
```



```
//找到第一个包含给定汉字（如“竹”）的成语
//返回值为指向chArray中被找到的成语的下一个字符的指针
// 如未找满足要求的成语到则返回NULL
char* findIdiom(char *chArray, char *theWord, char *idiom, int len)
{
    int i, j;
    char *pos = strstr(chArray, theWord);
    if (pos==NULL) return NULL;
    int n = pos - chArray;
    //寻找成语的开头. 注意：成语须以' '开头
    for(i=n-1;i>=0;i--)
        if ( chArray[i] == ' ' ) break;
    // 将找到的成语存入数组idiom. 注意：成语须以' '或'\n' 结尾

    for(j=0;j<len-1;j++)
    {
        if ( chArray[i+1+j]==' ' || chArray[i+1+j]=='\n' ) break;
        idiom[j] = chArray[i+1+j]; //复制了查到的成语，存放在idiom
    }
    // 结尾加上空字符
    idiom[j] = '\0';
    return chArray+i+j;
}
```



## 13.3 强调：程序调试

- 调试核心：定位错误
- 编译错误：
  - 错误位置由编译器直接给出
- 运行错误：
  - 运用排除法，不断分析、试探，定位错误



# 排除运行错误主要方法

- 测试驱动
  - 自动定位错误
- 程序日志
  - 根据日志，通过分析，帮助定位错误
- 跟踪调试
  - 断点设置、单步跟踪等，跟踪程序执行过程，来定位错误



谢谢大家！

