

# 计算机程序设计基础（1）

## 12 指针（下）

清华大学电子工程系

杨昉

E-mail: [fangyang@tsinghua.edu.cn](mailto:fangyang@tsinghua.edu.cn)



- 指针的定义与使用

- 指针的概念、定义与引用、注意事项

- 指针作为 **函数参数**

- 数组与指针

- 指针数组

- 一维数组 与指针


- 二维数组 与指针

- 数组指针作为函数参数



# 课程回顾: 指针



| 知识点     | 使用示例   | 说明   |
|---------|--|--|
| 指针的定义   | <code>int *p;</code>   | 用以存放其他变量所占存储空间的首地址的变量叫做 <b>指针</b> ;指针 <b>定义</b> 时前面有 “ <b>*</b> ”        |
| 直接访问    | <code>int a; a = 5;</code>   | 直接通过 <b>变量名</b> 进行操作   |
| 间接访问    | <code>int a,*p=&amp;a; *p=5;</code>  | 通过指针对变量进行修改, 指针在 <b>使用</b> 时变量名前加 “ <b>*</b> ” 表示 <b>间接存取</b>            |
| 悬浮指针    | <code>int a,*p; *p=a;</code>  | 指针变量中具有 <b>确定地址值</b> 后才能被引用  |
| 指针作为形参  | <code>void f (int*p){...}</code>   | 通过 <b>地址结合</b> ,实现形参与实参的 <b>双向传递</b>                                     |
| 指向指针的指针 | <code>int **q;</code>  | <b>指向指针型数据的指针</b>  |
| 多级间接访问  | <code>int x, *p, **q;<br/>p = &amp;x; q = &amp;p; **p = 3;</code>  | 通过指针实现间接访问,称为 <b>一级间接访问</b> ,通过指向指针的指针实现 <b>二级间接访问</b> 。依此类推,C语言允许多级间接访问 |

# 课程回顾: 指针



| 知识点           | 使用示例  | 说明   |
|---------------|---|--|
| 指针数组          | <code>int *a[10];</code>  | 每个元素均为 <b>指针类型数据的数组</b>                        |
| 一维数组名         | <code>int a[5];</code> 有 <code>a=&amp;a[0]</code>   | 一维数组名实际上是对应类型的 <b>常量指针</b>                     |
| 指向一维数组元素的指针   | <code>int a[5], *p=&amp;a[1];</code> 则 <code>p[1]</code> 、 <code>*(p+1)</code> 、 <code>*(a+2)</code> 、 <code>a[2]</code> 等价 | 指向数组元素的指针加一表示 <b>指向下一个元素</b> ;数组元素可用下标法或指针法表示  |
| 二维数组          | <code>int a[2][2]={ {1,2},{3,4}};</code>  | 静态二维数组以行为主 <b>线性存储</b> 于内存                     |
| 二维数组名         | <code>int a[5][5];*(a+i)+j=&amp;a[i][j];</code><br><code>a[i][j] = *(*a+i)+j</code>   | 二维数组名也是指针, 但需分清 <b>行指针</b> 和 <b>指向数组元素</b> 的指针 |
| 一维数组作为函数参数    | <code>void f(int *a){...};</code><br><code>void f(int a[]){...};</code>   | 形参和实参分别用 <b>数组名</b> 和 <b>指针变量</b> 时的各种等价写法     |
| 数组名作实参, 指针作形参 | <code>void f(int *b){...};</code><br><code>f( (int *)a );</code>  | 使用时需将二维数组名 <b>强制类型转换为指针变量</b> 或 <b>行指针变量</b>   |
| 指针数组作为实参      | <code>void f(int **b){...};</code><br><code>b[i]=&amp;a[i][0]</code>  | 将 <b>指针数组元素</b> 分别指向对应行首地址 <sub>4</sub>        |





## 12.1、动态内存

- ❑ malloc()函数
- ❑ calloc()函数
- ❑ realloc()函数
- ❑ free()函数

## 12.2、字符串与指针

- ❑ 字符串指针
- ❑ 字符串指针作为函数参数
- ❑ strstr()函数

## 12.3、函数与指针

- ❑ 用函数指针调用函数
- ❑ 函数指针数组
- ❑ 函数指针作为函数参数
- ❑ 返回指针值的函数
- ❑ main函数的形参



## 12.1 动态内存

- ❑ malloc() 函数
- ❑ calloc() 函数
- ❑ realloc() 函数
- ❑ free() 函数



- 上一讲中，都是先定义好数组，再将数组的地址赋给指针
  - 这种在函数内部定义局部数组的方式，是在**内存栈**上分配内存空间，通常不能开太大的数组
- 想要开大数组有以下两种方法：
  - 在函数外定义**外部数组**，或者**静态(static)数组**
    - 不够灵活，数组即便不再需要也会一直占据内存空间直到程序结束
    - 这会造成不能同时定义多个外部大数组或静态大数组
  - 在**内存堆**上开辟所需的动态内存空间并将首地址赋给指针
    - 通过调用内存申请函数进行**申请**，使用完毕后再调用相应函数**释放**



## ●关于动态内存的申请与释放，主要用到下面四个函数

### □malloc()函数

- 从内存堆(heap)上申请指定字节数的内存块，并返回该内存块的首地址

### □calloc()函数

- 在内存的动态存储区分配指定大小的内存块，并自动赋初值0

### □realloc()函数

- 在保证原有数据不变的前提下修改已分配内存块的大小

### □free()函数

- 释放上面三个函数分配的内存空间

### □使用时需包含头文件stdlib.h



# malloc()函数



- malloc()函数的功能是从内存堆上申请指定字节数的内存块，并返回该内存块的首地址

□ malloc函数的原型为：

`void *malloc(申请的字节数)`

□ 调用malloc函数时，必须进行强制类型转换

`(类型*) malloc(申请的字节数)`

□ 把返回的内存首地址转换成相应指针的类型，内存申请失败时会给指针赋“**NULL**”

# malloc()函数



## ●例12-1： malloc()用法示例

- 从内存堆上动态申请一块20个char型大小的内存给p，形成一个长度为20的动态char型数组
- 申请一块10个double指针大小的内存给q，形成一个长度为10的动态double型指针数组
- 申请一块20个int型大小的内存给指向每行长度为4的行指针a，形成一个5×4的动态二维数组

```
1 #include<stdlib.h>
2 void main() {
3     char *p;
4     double **q;
5     int (*a)[4];
6     p = (char*)malloc(sizeof(char)*20);
7     q = (double**)malloc(sizeof(double*)*10);
8     a = (int(*)[4])malloc(sizeof(int)*4*5);
9 }
```

- 函数前强制转换时括号内的类型名即是指针定义格式去掉指针名
- 例如“int(\*a)[4]”强制转换类型时用“int(\*)[4]”

# malloc()函数



- 例12-2：根据输入的数据来确定数组的长度
- 前面章节曾经说过，定义数组时其长度必须是常量表达式，因此难以实现
- 但通过指针动态分配内存就可以解决这个问题

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 void main() {
4     double *p;
5     int n, i;
6     scanf("%d", &n);
7     p = (double*)malloc(sizeof(double)*n);
8     if(p==NULL) { //当内存分配失败时会给指针赋空值
9         printf("Can't get memory!\n");
10        exit(1);
11    }
12    for(i=0; i<n; i++)
13        p[i]=i;
14    for(i=0; i<n; i++)
15        printf("%3.0f\n", p[i]);
16    free(p); //释放动态分配的内存
17 }
```

8

0

1

2

3

4

5

6

7

请按任意键继续……

# malloc()函数



## ●例12-3：动态二维数组

□利用指针数组实现行数固定、列数不定的二维数组

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 void main() {
4     double *p[4];
5     int k, m, i, j;
6     scanf("%d", &m);
7     for(k=0; k<4; k++) {
8         p[k] = (double*)malloc(sizeof(double)*m);
9         if(p[k]==NULL) {
10             printf("Can't get memory!\n"); exit(1);
11         }
12     }
```

```
13     for(i=0; i<4; i++)
14         for(j=0; j<m; j++)
15             p[i][j]=i*j;
16     for(i=0; i<4; i++) {
17         for(j=0; j<m; j++)
18             printf("%3.0f", p[i][j]);
19         printf("\n");
20     }
21     for(k=3; k>=0; k--)
22         free(p[k]); //内存释放顺序与申请顺序相反
23 }
```

```
3
0 0 0
0 1 2
0 2 4
0 3 6
请按任意键继续.....
```

向函数中传递这种方式生成的二维数组时，应当将指针数组“类型名 \*[]”或者指向指针的指针“类型名 \*\*”作为形参，指针数组名作为实参

# malloc()函数



□还可以利用行指针实现列数固定、行数不定的二维数组

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 void main() {
4     double (*p)[4];
5     int m, i, j;
6     scanf("%d", &m);
7     p = (double(*)[4])malloc(sizeof(double)*m*4);
8     if(p==NULL) {
9         printf("Can't get memory!\n"); exit(1);
10    }
11    for(i=0; i<m; i++)
12        for(j=0; j<4; j++)
13            p[i][j]=4*i+j;
14    for(i=0; i<m; i++) {
15        for(j=0; j<4; j++)
16            printf("%3.0f", p[i][j]);
17        printf("\n");
18    }
19    free(p);
20 }
```

向函数中传递这种方式生成的二维数组时，由于其在内存中的存储方式也是顺序存储，和静态二维数组是一致的，因此传递方式与上一讲介绍的静态二维数组相同

```
3
0 1 2 3
4 5 6 7
8 9 10 11
请按任意键继续……
```



# malloc()函数



□还可以通过指向指针的指针实现行数与列数都不定的二维数组

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 void main() {
4     double **p;
5     int m, n, i, j, k;
6     scanf("%d %d", &m, &n);
7     //为p开辟存放m个指向行的指针的内存空间
8     p = (double**)malloc(sizeof(double*)*m);
9     //为指向每一行的指针开辟存放n个变量的空间
10    for(k=0; k<m; k++)
11        p[k] = (double*)malloc(sizeof(double)*n);
12    //为了简明去掉了检测申请内存失败的代码
```

```
13    for(i=0; i<m; i++)
14        for(j=0; j<n; j++)
15            p[i][j]=n*i+j;
16    for(i=0; i<m; i++) {
17        for(j=0; j<n; j++)
18            printf("%3.0f", p[i][j]);
19        printf("\n");
20    } //释放内存的顺序与申请时相反
21    for(k=m-1; k>=0; k--)
22        free(p[k]);
23    free(p);
24 }
```

向函数中传递这种方式生成的动态二维数组的方式与传递用指针数组生成的二维数组的方式类似

3 4

0 1 2 3

4 5 6 7

8 9 10 11

请按任意键继续……

# calloc()函数



## ●函数用法为

**(类型 \*) calloc(元素个数n, 类型占据字节数size)**

- 功能：在内存的动态存储区中分配n个长度为size的连续空间
- 函数返回一个指向起始地址的指针；如果申请不成功则返回**NULL**
- 与malloc区别在于，calloc自动初始化分配的内存空间为零，而malloc不进行初始化，里面是随机的数据
- 同时calloc可申请的动态数组也更大

# realloc()函数



## ●函数用法为：

**指针名 = (数据类型\*)realloc(指针名, 新的内存长度)**

- 功能：在原先申请内存块的基础上，再重新申请一块更长的（或更短）的内存块，以实现内存块的动态增长
- 执行时，先判断原内存块后是否存在足够的连续空间，如果有，则扩大原先的内存块，并将首地址返回
- 反之，则开辟一块新的内存空间并将原先数据复制到此后释放原来的内存块(隐含自动释放，不需要free函数)，之后返回首地址，如果申请失败则返回**NULL**

# realloc()函数



## ●例12-4: realloc() 函数用法案例

□为简洁起见，略去内存申请失败检测模块

```
malloc 010E8EA0
realloc 010E9758
 0 1 2 3 4 5 6 7 8 9
请按任意键继续.....
```

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 void main() {
4     int i;
5     int *pn=(int*)malloc(5*sizeof(int));
6     printf("malloc %p\n", pn);
7     for (i=0; i<5; i++)
8         pn[i]=i;
9     pn=(int*)realloc(pn, 10*sizeof(int));
10    printf("realloc %p\n", pn);
11    for (i=5; i<10; i++)
12        pn[i]=i;
13    for (i=0; i<10; i++)
14        printf("%3d", pn[i]);
15    printf("\n");
16    free(pn);
17 }
```

# 内存堆与栈



- 内存栈由编译器自动分配和释放

- 局部变量等存放在内存栈中

- 内存栈的容量有限

- 手动分配的内存存放于内存堆中

- 能够申请更大内存

- 同时也需要手动释放

- Debug模式下，对于未初始化的栈与堆会进行不同处理

- “烫烫烫”与“屯屯屯”



c++吧 关注 16w

我++! 程序输出烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫  
烫烫烫烫烫烫烫烫



洛天依 11 楼主

1楼 2014-8-7

是不是电脑过热警告啊，我吓得直接拔电源了





## ●对于未初始化的**内存栈**

### □编译器会默认写入0xcc

- 以字符串格式输出后会显示“烫烫烫...”
- 同学们可能已遇到过这类问题

```
1 #include <stdio.h>
2 void main() {
3     char a[10];
4     printf("%s", a);
5 }
```

烫烫烫烫烫烫烫迎? 请按任意键继续...

## ●对于未初始化的**内存堆**

### □编译器会默认写入0xcd

- 以字符串格式输出后会显示“屯屯屯...”

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main() {
4     char*a = (char*)malloc(sizeof(char)*10);
5     printf("%s", a);
6     free(a); //手动释放
7 }
```

屯屯屯屯屯 e请按任意键继续...



## 12.2 字符串与指针

- 字符串指针
- 字符串指针作为函数参数
- strstr()函数

## ●在C语言中,表示一个字符串有以下两种形式

用字符数组存放一个字符串

```
1 #include <stdio.h>
2 void main() {
3     char s[]="How do you do!";
4     printf("%s\n", s);
5 }
```

How do you do!  
请按任意键继续……

用字符指针指向一个字符串

```
1 #include <stdio.h>
2 void main() {
3     char *s="How do you do!";
4     printf("%s\n", s);
5 }
```

How do you do!  
请按任意键继续……

# 字符数组与字符指针的联系



- **字符数组**由元素组成，每个元素中存放一个**字符**；而**字符指针变量**中存放的是**地址**，也能作函数参数

□对数组赋初值：

```
char s[]="How do you do!";
```

□对字符指针变量的初始化：

```
char *s="How do you do!";
```

- 只能对字符数组中的各个元素赋值，而不能用赋值语句对整个字符数组赋值

□下面代码是**错误**的，因为数组名是**常量指针**

```
char s[20]; s="How do you do!";
```



□而对于**字符指针**则可以用赋值语句对整个字符串进行赋值

# 字符数组与字符指针



- 字符数组名虽然代表地址，数组名是常量，值不能改变：

```
char s[20]; s=s+4;
```



- 但字符数组的元素可以进行修改：

```
char s[20]="How do you do!"; s[0]='W';
```



- 与字符数组相反，字符指针本身的值是可以修改：

```
char *s="How do you do!"; s=s+4;
```



- 但字符指针所指向的字符串是常量，不能进行修改：

```
char *s="How do you do!"; s[0]='W';
```





# 字符数组与字符指针



- 可以用下标形式引用指针变量所指向的字符串中的字符

□例如

```
char *s="How do you do!";  
printf("%c ", s[4]);  
printf("%c\n", *(s+7));
```

输出为 d y

- 每一个字符串自动拥有一个首地址指针

□例如

```
printf("%s\n", "How do you do!" + 4);
```

输出为 do you do!

□例如

```
printf("%s\n", "How do you do!"[4]);
```

输出为 d

# 字符数组与字符指针



## ●可以用**sizeof**操作符求字符串所占内存的大小

□例如: `printf("%d\n", sizeof("a"));`

求字符串 “a” 所占内存大小,输出为2

□例如: `printf("%d\n", sizeof("abcd"));` 输出为5

□例如: `printf("%d\n", sizeof("ab\0cd"));` 输出为6

□如果把上面的**sizeof**操作改为使用字符串函数**strlen**求字符串长度:

`printf("%d\n", strlen("ab\0cd"));` 输出变为2

# 字符数组与字符指针



- 可以通过输入字符串的方式为字符数组输入字符元素，但不能通过输入函数让字符指针变量指向一个字符串
- 例12-5：通过控制台输入字符串给变量赋值

□给字符数组赋值，代码如下

```
1 #include <stdio.h>
2 void main() {
3     char str1[20], str2[20];
4     scanf("%s", str1);
5     scanf("%s", str2);
6     printf("str1=%s\n", str1);
7     printf("str2=%s\n", str2);
8 }
```

```
asdfghjkl
1234567890
str1=asdfghjkl
str2=1234567890
请按任意键继续.....
```

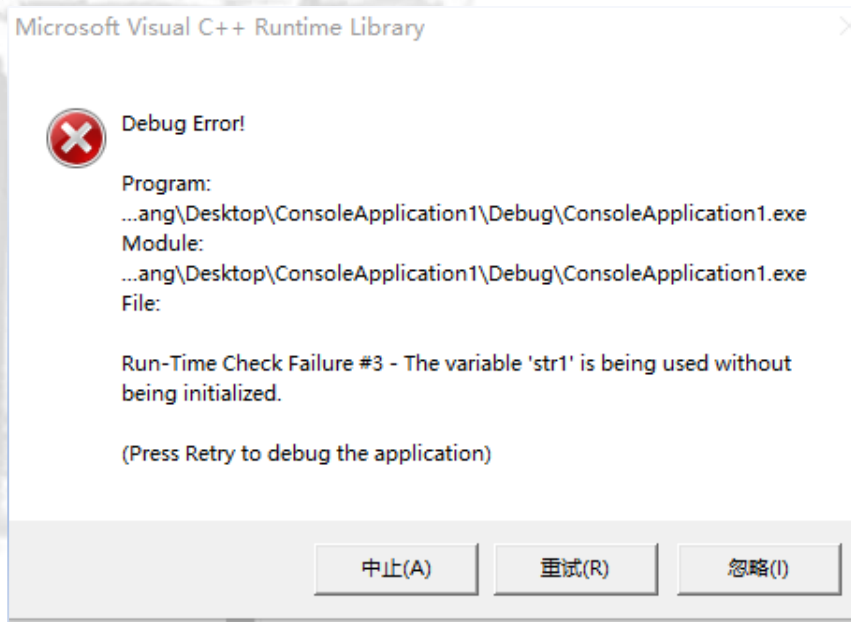
# 字符数组与字符指针



## □通过键盘输入给字符指针赋值

- 编译时会出现警告：warning C4700: 使用了未初始化的局部变量 “str1”
- 这两个字符指针变量没有初始化（赋初值）就使用了
- 由于指针变量中没有正确指向合理的内存区地址，则从键盘输入的字符串无存储空间可存放，会导致致命错误

```
1 #include <stdio.h>
2 void main() {
3     char *str1, *str2;
4     scanf("%s", str1);
5     scanf("%s", str2);
6     printf("str1=%s\n", str1);
7     printf("str2=%s\n", str2);
8 }
```



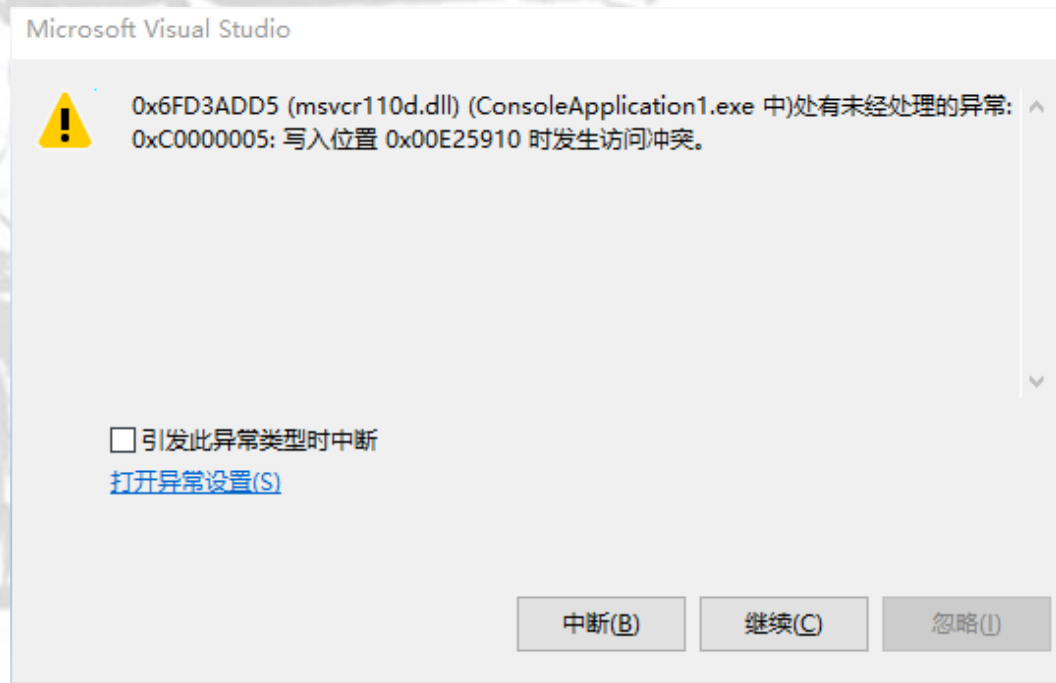
# 字符数组与字符指针



## □对两个指针赋初值再进行操作

- 运行出现致命错误：两个字符指针变量中都指向了存放字符串的内存区
- 但由于字符指针所指的字符串都是存放在常量区的常量字符串
- 常量字符串只能读而不能改写，因此scanf试图向常量区写操作导致致命错误

```
1 #include <stdio.h>
2 void main() {
3     char *str1="ccc1", *str2="ccc2";
4     scanf("%s", str1);
5     scanf("%s", str2);
6     printf("str1=%s\n", str1);
7     printf("str2=%s\n", str2);
8 }
```





# 字符数组与字符指针



## □把从键盘输入改为直接赋值，程序能够正确运行

- 虽然在定义字符指针变量的同时给这两个指针均赋了初值地址（即字符串“cccc”的首地址），并且其中字符串的长度为4（实际占5个字节，还有1个字符串结束符）
- 但在执行赋值语句时，编译系统分别为两个新的字符串常量分配了内存空间，存放在常量区，并分别将这两个字符串的首地址赋给字符指针变量str1与str2
- 这两个指针变量获得了新的地址值，分别指向新的字符串"asdfghjkl"与"1234567890"

```
1 #include <stdio.h>
2 void main() {
3     char *str1="cccc", *str2="cccc";
4     str1="asdfghjkl";
5     str2="1234567890";
6     printf("str1=%s\n", str1);
7     printf("str2=%s\n", str2);
8 }
```

```
str1=asdfghjkl
str2=1234567890
请按任意键继续……
```

# 字符数组与字符指针



- 综上所述，**字符指针只能存放字符串的首地址**

- 对于赋值语句中的字符串，编译系统会给予分配存储空间

- 对于通过键盘输入的字符串，系统是不分配存储空间的

- 字符数组之所以能通过输入字符串的方式为字符数组元素输入字符，是因为在定义字符数组时已经为数组分配了存储空间

- 可以用指针变量所指向的字符串表示**程序中的任何字符串**

- 如printf函数中的格式字符串

# 字符数组与字符指针



## ●例12-6：以下两段代码效果是等价的

```
1 #include <stdio.h>
2 void main() {
3     int a=12345;
4     double b=1234.5678;
5     char *format;
6     format = "a=%10d, b=%12.6f\n";
7     printf(format, a, b);
8 }
```



```
1 #include <stdio.h>
2 void main() {
3     int a=12345;
4     double b=1234.5678;
5     printf("a=%10d, b=%12.6f\n", a, b);
6 }
```

# 字符数组与字符指针



## □用字符数组也能实现上述功能

a= 12345, b= 1234.567800  
请按任意键继续……

```
1 #include <stdio.h>
2 void main() {
3     int a=12345;
4     double b=1234.5678;
5     char format[ ]="a=%10d, b=%12.6f\n";
6     printf(format, a, b);
7 }
```



```
1 #include <stdio.h>
2 void main() {
3     int a=12345;
4     double b=1234.5678;
5     printf("a=%10d, b=%12.6f\n", a, b);
6 }
```

- 练习11-1: `char *p; printf(p, p="p=%s");`的输出结果  
等价于: `printf("p=%s", "p=%s");` 输出结果是: `p=p=%s`

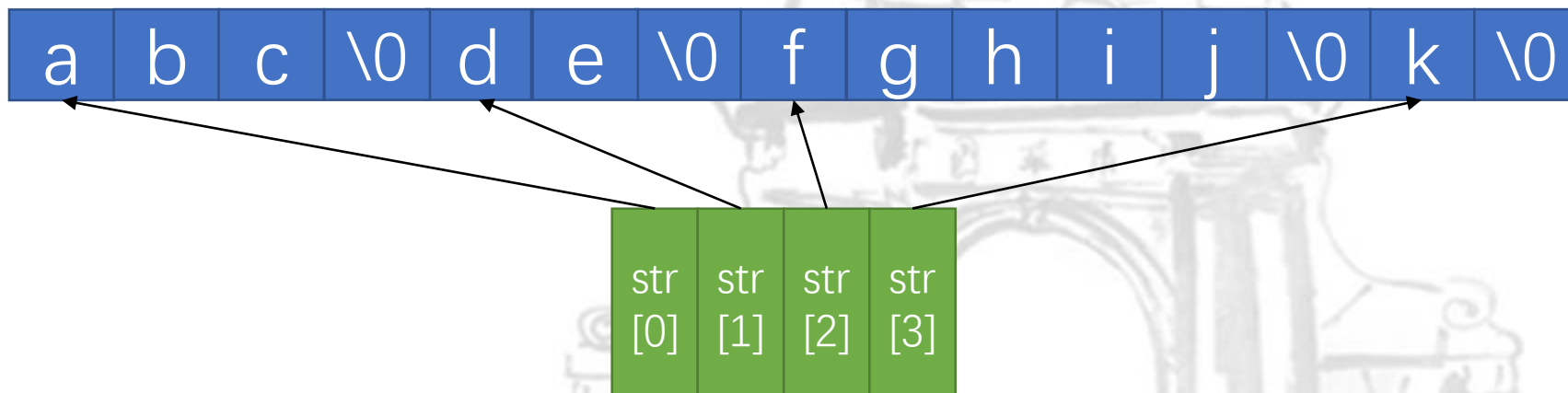
# 紧凑型字符串数组



- 定义如下的字符型指针数组，构成紧凑型字符串数组：

```
char *str[]={"abc", "de", "fghij", "k"};
```

/\* str是一个字符型指针数组, 长度为4, 由字符串个数决定的 \*/



□str[0], str[1], str[2], str[3]指针可以交换，但不能相互复制

□str[0]=str[1]



strcpy(str[1],str[2])



□字符型指针所指向的字符串都是常量，不能修改

## ●而二维字符数组又有所不同

```
char str[][6]={"abc", "de", "fghij", "k"};
```

|   |    |    |    |   |    |
|---|----|----|----|---|----|
| a | b  | c  | \0 |   |    |
| d | e  | \0 |    |   |    |
| f | g  | h  | i  | j | \0 |
| k | \0 |    |    |   |    |

❑str[0],str[1],str[2],str[3]字符串可以互相复制，但不能交换

❑strcpy(str[1],str[2]);



str[1]=str[2];



❑str[1]、str[2]都是常量



# 紧凑型字符串数组



## ●例12-7：利用紧凑型字符串数组实现根据年月日判断星期几

```
1 #include <stdio.h>
2 void main() {
3     char *Weeks[ ]={"Sunday", "Monday", "Tuesday",
4     "Wendesday", "Thursday", "Friday", "Saturday"};
5     int year, month, day, weekday, i;
6     int months[12]={0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30};
7     scanf("%d%d%d", &year, &month, &day);
8     if (year%4==0 && year%100!=0 || year%400==0)
9         months[2]++;
10    for (i=2; i<12; i++)
11        months[i] += months[i-1];
12    year--;
13    weekday=year+year/4-year/100+year/400+months[month-1]+day;
14    weekday %= 7;
15    printf("%s\n", Weeks[weekday]);
16 }
```

2021 12 03

Friday

请按任意键继续……

普通年份除以7余1，闰年除以7余2。且依据现行历法倒推，公元元年1月1日是礼拜一

# 字符指针作为函数参数



- 将一个字符串从一个函数传递到另一个函数，可以将字符数组名或字符指针变量作为函数参数

## □ 实参与形参都用字符数组名

```
void main() {  
    char str[10];  
    ...  
    f(str, 10);  
    ...  
}
```

```
void f(char str[], int n) {  
    ...  
    ...  
    ...  
    ...  
}
```

## □ 实参用字符数组名, 形参用字符指针变量

```
void main() {  
    char str[10];  
    ...  
    f(str, 10);  
    ...  
}
```

```
void f(char *s, int n) {  
    ...  
    ...  
    ...  
    ...  
}
```

# 字符指针作为函数参数



## □ 实参与形参都用字符指针变量

```
void main() {  
    char str[10], *p=str;  
    ...  
    f(p, 10);  
    ...  
}
```

```
void f(char *s, int n) {  
    ...  
    ...  
    ...  
}
```

同样，这四种表达方式是等价的

## □ 实参用字符指针变量,形参用字符数组名

```
void main() {  
    char str[10], *p=str;  
    ...  
    f(p, 10);  
    ...  
}
```

```
void f(char str[], int n) {  
    ...  
    ...  
    ...  
}
```

# 字符指针作为函数参数



## ●例12-8：实现字符串赋值及计算字符串长度的函数

```
1 #include <stdio.h>
2 int str_copy(char *str1, char *str2) {
3     int k=0;
4     while (str1[k] != '\0') {
5         str2[k]=str1[k];
6         k=k+1;
7     }
8     str2[k]='\0';
9     return k;
10 }
```

```
11 void main() {
12     char str1[20], str2[20];
13     int k;
14     printf("input str1: ");
15     scanf("%s", str1);
16     printf("str1=%s\n", str1);
17     k=str_copy(str1, str2);
18     printf("str2=%s\n", str2);
19     printf("k=%d\n", k);
20 }
```

```
input str1: qwertyuiop
str1=qwertyuiop
str2=qwertyuiop
k=10
请按任意键继续……
```

# 字符指针作为函数参数



□ 前述的str\_copy()函数还可以改写为

```
1 int str_copy(char *str1, char *str2) {  
2     int k=0;  
3     while ( str2[k] = str1[k] )  
4         k=k+1;  
5     return k;  
6 }
```

```
1 int str_copy(char *str1, char *str2) {  
2     char *k=str1;  
3     while ( *str2++ = *str1++ );  
4     return str1 - k - 1;  
5 }
```

```
1 int str_copy(char *str1, char *str2) {  
2     int k=0;  
3     while ( *str2++ = *str1++ )  
4         k=k+1;  
5     return k;  
6 }
```

当运行到字符串最后的结束符' \0' 时, 会出现str2[k]=' \0' 的情况, 这个表达式本身的值为0, 因此执行完毕后会退出循环, 另外两个代码同理

# 字符指针作为函数参数



□在上述程序中,函数str\_copy( )中的两个形参均定义为字符指针变量,也可以定义为字符数组

- ✓ 需要说明的是,在主函数中定义的两个字符数组的长度均为20,因此,被复制的字符串长度最大为19
- ✓ 在函数str\_copy( )中,是不限制字符串长度的,它是用字符串结束符来判断是否结束复制工作

```
1 int str_copy(char str1[], char str2[]) {  
2     int k=0;  
3     while(str1[k]!='\0') {  
4         str2[k]=str1[k];  
5         k=k+1;  
6     }  
7     str2[k]='\0';  
8     return k;  
9 }
```



# strstr()函数



- strstr()函数的原型为:

**char \*strstr(char \*str1, char\* str2)**

□功能：查找字符串str2是否在字符串str1中出现过，是则返回第一次出现的位置，否则返回**NULL**

- 例12-9： strstr()用法示例

```
1 #include <stdio.h>
2 #include <string.h>
3 void main() {
4     char *s1="abcdeabcdeabcdeabcde";
5     char *s2 ="dea", *s3;
6     s3=strstr(s1, s2);
7     printf("%s\n", s3);
8 }
```

deabcdeabcdeabcde  
请按任意键继续……



## 12.3、函数与指针

- 用函数指针调用函数
- 函数指针数组
- 函数指针作为函数参数
- 返回指针值的函数
- main函数的形参

# 用函数指针调用函数



- 一般来说,程序中的每一个函数经编译链接后,其目标代码在计算机内存中是连续存放的
  - 该代码的首地址就是函数执行时的入口地址
  - 在第一章绪论中曾经介绍过,现代计算机体系架构中的程序指令都是存储在计算机内存中
- 在C语言中,函数名本身就代表该函数的入口地址
  - 所谓指向函数的指针,就是指向函数的入口地址

# 用函数指针调用函数



- 指向函数的指针变量其定义形式如下:

**类型标识符 (\*指针变量名)(所指函数的参数说明列表);**

□即使参数类型说明列表为空,最后的()不能省略

- 定义了函数指针变量后,就可以通过它间接调用它所指向的函数

□同其他类型的指针一样,首先必须将一个函数名 (代表该函数的入口地址,即函数指针) 赋给函数指针变量

□然后才能通过函数指针间接调用该函数

# 用函数指针调用函数



- 例12-10：从键盘输入一个大于1的正整数n,当n为偶数时, 计算 $1+1/2+1/4+\dots+1/n$  当n为奇数时计算 $1+1/3+1/5+\dots+1/n$

n为偶数时计算值的函数even()：

```
1 double even(int n) {  
2     int k; double sum;  
3     sum=1.0;  
4     for (k=2; k<=n; k=k+2)  
5         sum=sum+1.0/k;  
6     return sum;  
7 }
```

n为奇数时计算值的函数odd()：

```
1 double odd(int n) {  
2     int k; double sum;  
3     sum=1.0;  
4     for (k=3; k<=n; k=k+2)  
5         sum=sum+1.0/k;  
6     return sum;  
7 }
```

# 用函数指针调用函数



## 主函数常规方式调用函数

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double even(int), odd(int);
5     printf("input n;");
6     scanf("%d", &n);
7     if (n>1) {
8         if (n%2==0) /*n为偶数*/
9             printf("even=%9.6f\n", even(n));
10        else /*n为奇数*/
11            printf("odd=%9.6f\n", odd(n));
12    }
13    else printf("ERR!\n");
14 }
```

## 主函数通过函数指针调用函数

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double even(int), odd(int), (*p)(int);
5     printf("input n;");
6     scanf("%d", &n);
7     if (n>1) {
8         if (n%2==0) p=even;
9         /*n为偶数, 指针变量p指向函数even()*/
10        else p=odd;
11        /*n为奇数, 指针变量p指向函数odd()*/
12        printf("sum=%9.6f\n", (*p)(n));
13    }
14    else printf("ERR!\n");
15 }
```

二者的最终效果是等价的！



# 用函数指针调用函数



- 在给函数指针变量赋值时,只需给出函数名,不必给出参数
- 可以通过指向函数的指针变量来调用函数,其调用形式为

**(\*函数指针变量名)(实参表)**

- 也可以用下面的调用形式

**函数指针变量名(实参表)**

- 比如将前面代码中的(\*p)(n)改写为p(n)
- 对函数指针变量运算是没有意义的
  - 若p为函数指针变量,则p=p+1是没有意义的

# 函数指针数组



- 同理，还可以定义函数指针数组

- 数组中的每一个元素都是一个函数指针，用于指向不同的函数

- 定义方式：

`int (*p[5]) ();`

- 还可以同样定义二维函数指针数组：

`int (*proc [5][5]) ();`

- 通过使用函数指针数组调用不同的函数，实现程序的简化

- 使函数指针数组每个元素分别指向不同的函数

- 需要时依次调用，减少代码冗余



## ●例12-11：用函数指针数组改写例12-10的主函数

□函数指针数组元素p[0]指向even，p[1]指向odd

```
1  #include <stdio.h>
2  void main() {
3      int n, k;
4      double even(int), odd(int);
5      double (*p[2])(int)={even, odd};
6      printf("input n;");
7      scanf("%d",&n);
8      if (n>1)
9          printf("sum=%9.6f\n", p[n%2](n));
10         /* 或(*p[n%2])(n) */
11     else
12         printf("ERR!\n");
13 }
```

# 函数指针变量作为函数参数



- 函数指针作为某函数的参数时

- 可以实现将函数指针所指向的函数入口地址传递给该函数

- 当函数指针指向不同函数的入口地址时，在该函数中就可以调用不同的函数

- 且不需要对该函数体作任何修改

- 例12-12：用不动点迭代法求下列方程的根

- 1)  $x - 1 - \arctan(x) = 0$

- 2)  $x - 0.5\cos(x) = 0$

- 3)  $x^2 + x - 6 = 0$

# 函数指针变量作为函数参数



□root()函数是对任一函数进行迭代求根的函数

```
1 #include <stdio.h>
2 #include <math.h>
3 int root(double *x, int m, double eps, double (*f)()) {
4     double x0;
5     do {
6         x0=*x;
7         *x= f(x0);/* 或 *x= (*f)(x0); */
8         m=m-1;
9     } while ((m!=0)&&(fabs(*x-x0)>=eps));
10    if (m==0)
11        return(0);
12    return(1);
13 }
```

# 函数指针变量作为函数参数



## □主函数和其他部分程序如下：

```
14 /* f(x)=1+arctan(x) */
15 double f1(double x)
16 {return(1.0+atan(x));}
17 /* f(x)=0.5cos(x) */
18 double f2(double x)
19 {return(0.5*cos(x));}
20 /* f(x)=(6+3x-x^2)/4 */
21 double f3(double x)
22 {return((6.0+3*x-x*x)/4);}
```

```
23 void main() {
24     double x, (*p)(double);
25     x=1.0; p=f1;
26     if (root(&x, 50, 0.00001, p))
27         printf("x1=%f\n", x);
28     x=1.0; p=f2;
29     if (root(&x, 50, 0.00001, p))
30         printf("x2=%f\n", x);
31     x=1.0;
32     if (root(&x, 50, 0.00001, f3))
33         printf("x3=%f\n", x);
34 }
```

```
x1=2.132267
x2=0.450183
x3=2.000000
请按任意键继续.....
```



# 返回指针值的函数



- 在C语言中,一个函数不仅可以返回整型、字符型、实型等数据,也可以返回指针类型的数据

□即C语言中还允许定义返回指针值的函数,其形式如下:

```
int *fun() {  
    int *p;  
    ...  
    ...  
    return p;  
}
```

□定义的函数fun( )将返回一个指向整型量的指针值

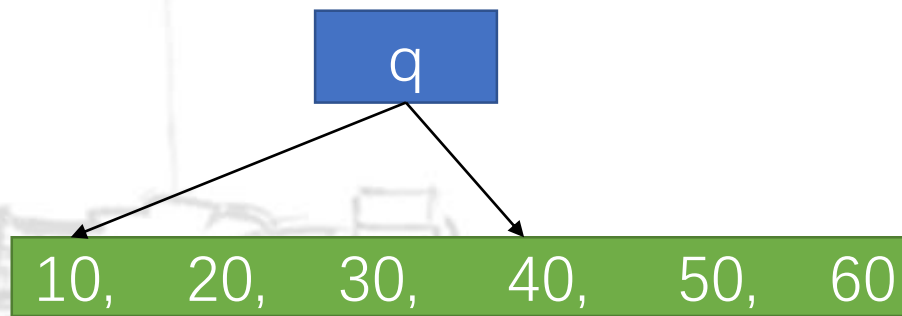
□在C语言中,可以定义返回任何类型指针的函数

# 返回指针值的函数



## ●例12-13：返回指针函数举例

```
1 #include <stdio.h>
2 int *f(int *p) {
3     p += 3;
4     return p;
5 }
6 void main() {
7     int a[]={10, 20, 30, 40, 50, 60}, *q=a;
8     q = f(q);
9     printf("%d\n", *q);
10 }
```



40  
请按任意键继续·····

调用f后, q的值改变了, 指针函数通过函数名返回指针值

## ●练习12-2：返回函数指针

```
1 #include <stdio.h>
2 int f1() {
3     printf("f1!\n"); return 1;
4 }
5 int (*f(int n)) () {
6     int (*q)()=f1; return q;
7 }
8 void main() {
9     int (*p)();
10    p = f(1);
11    p(); //表示执行p指向的函数
12    printf("OK!\n");
13 }
```

```
f1!
OK!
请按任意键继续.....
```

指针函数通过函数名返回  
“**指向函数名的指针**”  
或者说：f函数返回一个  
**指向函数的指针**

# main函数的形参



- 如果一个C程序的主函数需要外界为之传递参数时，就需要通过main函数形参来传递参数，其一般形式如下：

```
void main(int argc, char *argv[]) {  
  
    ...  
  
}
```

□argv是一个字符型指针数组，用以指向不同的字符串

□argc的值是argv指向的字符串个数+1

□在控制台命令行中运行程序时，如果主函数没有参数，则直接用文件名加回车即可运行，否则需输入以下命令行：

**文件名 字符串1 字符串2 …… <回车>**



- 例12-14：编写一个命令程序，其命令符为10-16，用以输出控制台命令行中除命令符外以空格分隔的所有字符串
  - 第四行中k从1开始，命令行中输入的命令为：文件名 字符串1 字符串2 ……，这些字符串的值分别赋给argv[0], argv[1], argv[2]
  - 即argv[0]中存储的字符是.exe文件的名称，这是不需要输出的

```
1 #include <stdio.h>
2 void main(int argc, char *argv[]) {
3     int k;
4     for (k=1; k<=argc-1; k++)
5         printf("%s\n", argv[k]);
6 }
```

# main函数的形参



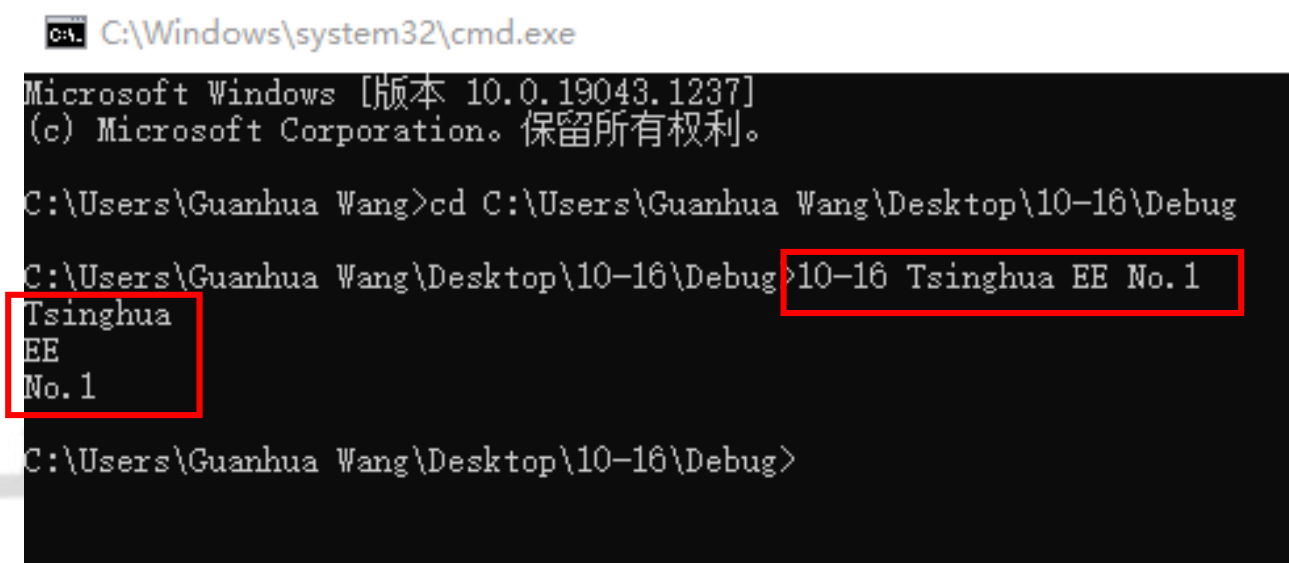
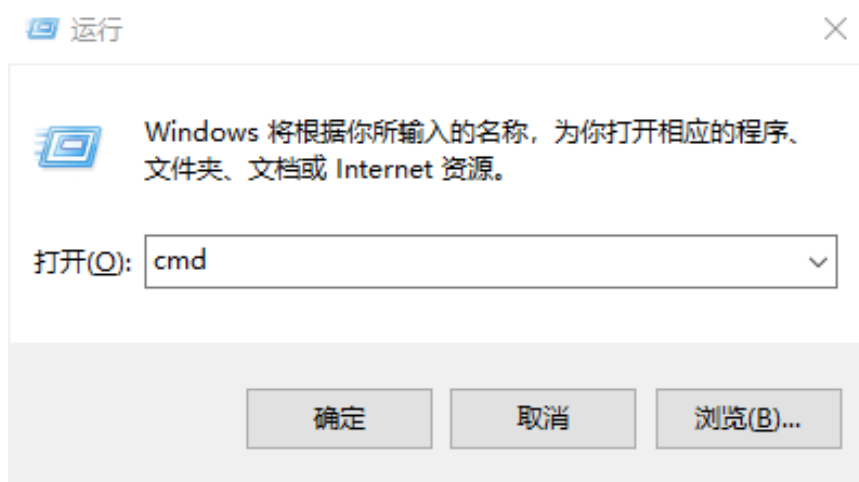
## ●运行结果：

□首先，按下键盘“win” + “R” 或右击开始，选择“运行”，  
在出现的窗口中输入“cmd”，调出控制台

□之后将路径cd到可执行文件的目录下

10-16是可执行文件的名称

□输入字符串：“10-16 Tsinghua EE No.1”，按下回车





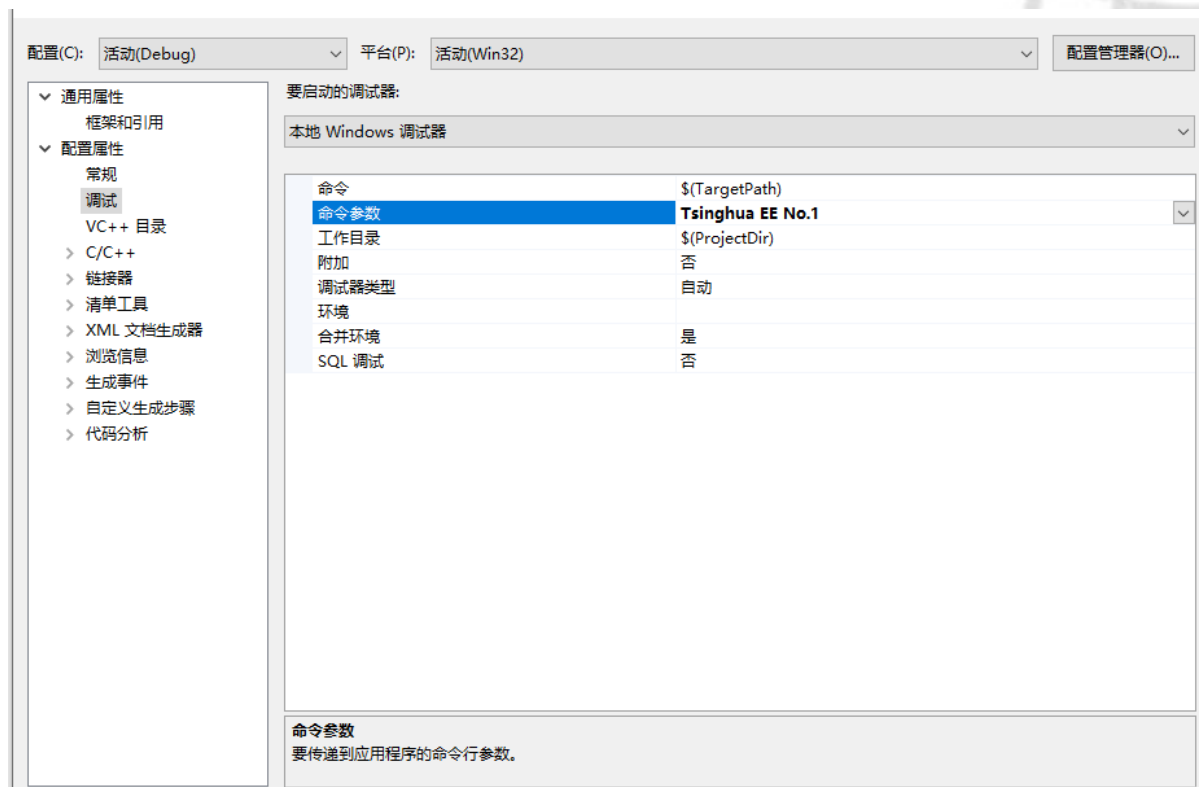
# main函数的形参



## ●还可以直接在VS中设置命令行参数

□项目-><文件名>属性->配置属性->调试->命令参数

□具体参见课本P270



Tsinghua  
EE  
No.1  
请按任意键继续……



- 练习11-3：阅读以下程序，程序生成的可执行文件名为test.exe，使用命令行test Tsinghua No.1运行该文件

□请输出结果，并分析原因

```
1 #include <stdio.h>
2 void main(int argc, char *argv[]) {
3     printf("%d\n", argc);
4     printf("%c-%s\n", *(argv[1]+1), argv[2]);
5 }
```

```
3
s-No.1
请按任意键继续...
```



## ●例12-15：利用变步长梯形法实现积分的数值计算

□ 设有定积分  $s = \int_a^b f(x)dx$ ，算法描述如下：

□ (1) 将积分  $[a,b]$  划分为  $n$  等分，每个区间长度  $h=(b-a)/n$

□ (2) 每个积分区间用梯形近似公式计算该区间积分值，最后将各个区间的值加和作为最终积分计算结果，则有

$$T_n = \frac{h}{2} \sum_{k=0}^{n-1} (f(x_k) + f(x_{k+1})), \text{ 其中 } x_k = a + k * h$$



□ (3) 在 $T_n$ 的基础上计算 $T_{2n}$

$$\begin{aligned} T_{2n} &= \frac{h}{4} \sum_{k=0}^{n-1} ((f(x_k) + f(x_{k+0.5})) + (f(x_{k+0.5}) + f(x_{k+1}))) \\ &= \frac{h}{4} \sum_{k=0}^{n-1} (f(x_k) + f(x_{k+1})) + \frac{h}{2} \sum_{k=0}^{n-1} f(x_{k+0.5}) \\ &= \frac{1}{2} T_n + \frac{h}{2} \sum_{k=0}^{n-1} f(x_{k+0.5}) \end{aligned}$$

□ (4) 若对于规定的误差 $\epsilon$ ,  $|T_{2n} - T_n| < \epsilon$ , 则返回 $T_{2n}$ 作为最终的结果, 否则令 $n=2n$ , 返回第 (1) 步



- 在计算时，设定初始区间划分个数 $n=1$
- 在计算前将误差 $p$ 的初值设为 $eps+1$ ，使其满足进行循环的条件

```
1  #include <math.h>
2  double fpts(double a, double b,
3  double eps, double (*f)(double)) {
4      int n, k;
5      double fa, fb, h, t1, p, s, x, t;
6      fa=(*f)(a); fb=(*f)(b);
7      n=1; h=b-a;
8      t1=h*(fa+fb)/2.0;
9      p=eps+1.0;
10     while (p>=eps) {
11         s=0.0;
12         for (k=0;k<=n-1;k++) {
13             x=a+(k+0.5)*h;
14             s=s+(*f)(x);
15         }
16         t=(t1+h*s)/2.0; /*计算T2n*/
17         p=fabs(t1-t);   /*计算精度*/
18         t1=t; n=n+n;
19         h=h/2.0;
20     }
21     return t;
22 }
```

# 程序举例



□在前面函数的基础上，计算下面三个积分，精度要求 $1e-5$

$$1) \int_0^1 e^{-x^2} dx \quad 2) \int_{-1}^1 \frac{1}{1+25x^2} dx \quad 3) \int_0^1 \frac{\ln(1+x)}{1+x^2} dx$$

```
23 double f1(double x)
24 { return(exp(-x*x)); }
25 double f2(double x)
26 { return(1.0/(1+25*x*x)); }
27 double f3(double x)
28 { return(log(1+x)/(1+x*x)); }
```

```
s1=7.468232e-001
s2=5.493573e-001
s3=2.721969e-001
请按任意键继续.....
```

```
29 void main() {
30     double f1(double), f2(double), f3(double),
31           (*p)(double);
32     p=f1;
33     printf("s1=%e\n", ffts(0.0, 1.0, 0.00001, p));
34     p=f2;
35     printf("s2=%e\n", ffts(-1.0, 1.0, 0.00001, p));
36     p=f3;
37     printf("s3=%e\n", ffts(0.0, 1.0, 0.00001, p));
38 }
```





## □也可以用函数指针数组实现

```
29 void main() {  
30     double f1(double), f2(double), f3(double),  
31         (*p[3]) ()={f1, f2, f3};  
32     printf("s1=%e\n", ffts(0.0, 1.0, 0.00001, p[0]));  
33     printf("s2=%e\n", ffts(-1.0, 1.0, 0.00001, p[1]));  
34     printf("s3=%e\n", ffts(0.0, 1.0, 0.00001, p[2]));  
35 }
```

## □或者直接使用函数名

```
29 void main() {  
30     double f1(double), f2(double), f3(double);  
31     printf("s1=%e\n", ffts(0.0, 1.0, 0.00001, f1));  
32     printf("s2=%e\n", ffts(-1.0, 1.0, 0.00001, f2));  
33     printf("s3=%e\n", ffts(0.0, 1.0, 0.00001, f3));  
34 }
```



- 动态内存的申请
  - 动态数组的生成
- 字符数组与字符指针
  - 字符指针与字符数组的异同点：初始化与赋值
- 函数与指针
  - 用函数指针变量调用函数
  - 函数指针数组：实现程序的简化
  - main()函数的形参

# 本节作业



## ●第十二次作业:

□P273- P274习题5、8、10，要求写出解答过程，电子版  
or 纸质版完成后拍照上传到word文档上传到网络学堂

□P275习题13，要求上机实验，将源代码和运行结果截图  
粘贴到word文档中提交到网络学堂

# 附加作业



## ●三维动态数组

- 对于输入的自然数 $a$ ,  $b$ ,  $c$ , 请建立三维动态数组 $D$ , 使其维度为 $a*b*c$ ; 给数组 $D$ 赋值, 其中第 $a\_1$ 行、 $b\_1$ 列、 $c\_1$ 层位置的元素为 $(c\_1 - 1)*a*b + (a\_1 - 1)*b + b\_1$ ; 将数组按层依次输出

## ●函数指针数组

- 二分法求零点是一种有效的方法, 请查阅资料了解其基本原理
- 给定函数 $f_1(x)=\log_{10}(x)$ ,  $f_2(x)=1/x-1.5$ ,  $f_3(x)=e^x-1/x$ , 定义域 $0<x<10$
- 请用C语言实现二分法求零点函数, 要求将函数指针作为参数 ( $f_1, f_2, f_3$ ), 输出三个函数的零点

## ●main函数形参

- 使用`main`函数的参数, 实现一个整数计算器, 程序可以接受三个参数, 第一个参数“-a”选项执行加法, “-s”选项执行减法, “-m”选项执行乘法, “-d”选项执行除法, 后面两个参数为操作数
- 注意: 请实现程序检查输入边界, 当输入参数量超过上述要求时报错

THANKS