

# C语言与程序设计

## The C Programming Language



---

### 第8章 指针

毛伏兵

华中科技大学计算机学院

# 8.1 指针的概念与指针的使用

## 8.1.1 指针的概念

- 变量占有一定数目(根据类型)的连续存储单元;

`short x, a[5];`

- 变量的连续存储单元首地址称为**变量的地址**。

`&x,     a <= => &a[0]`

- 思考：用什么类型的变量来保存地址数据？**

地址		变量名
0xFEBC		x
0xFEBD		
0xFEC0		a[0]
0xFEC1		a[1]
0xFEC2		a[2]
0xFEC3		a[3]
0xFEC4		a[4]

## 8.1.2 指针变量的声明

数据类型 \*标识符;



所指 (即所指向) 变量的数据类型

```
short x=1, y=2, a[10], *p;
```

```
p=&x;
```

```
p=a;
```

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

思考: 为什么使用指针?

# 思考：为什么使用指针？

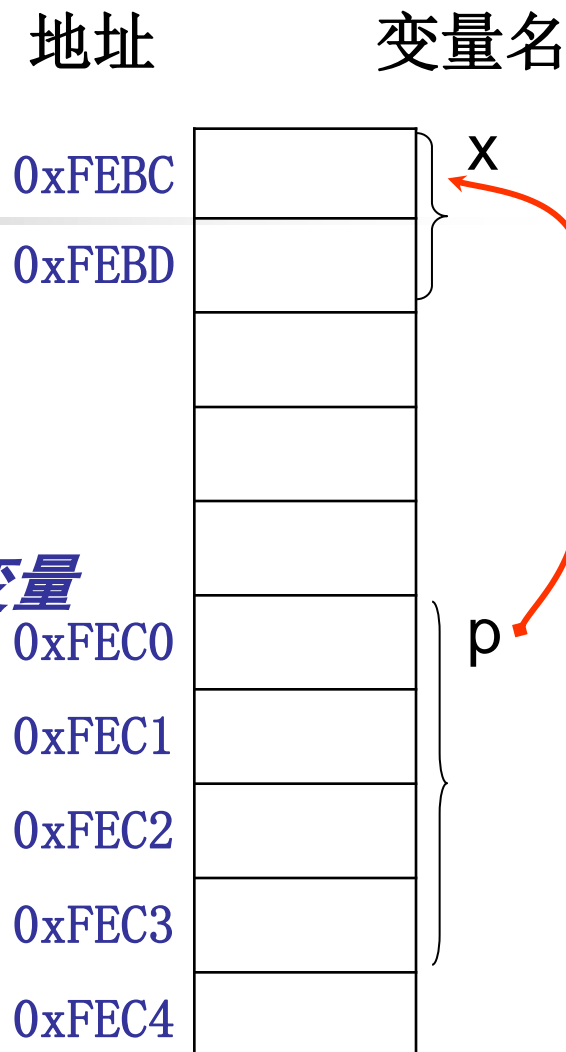
- 直接：通过变量名存取（或访问）变量

`x=0x1234 ; printf( "%x" , x);`

- 间接：通过变量的地址（即指针）存取变量

`p=&x ;`  
`printf( "%x" , *p);`

先访问 p，得到x的地址，  
再通过该地址找到它所指向的单元中的值。

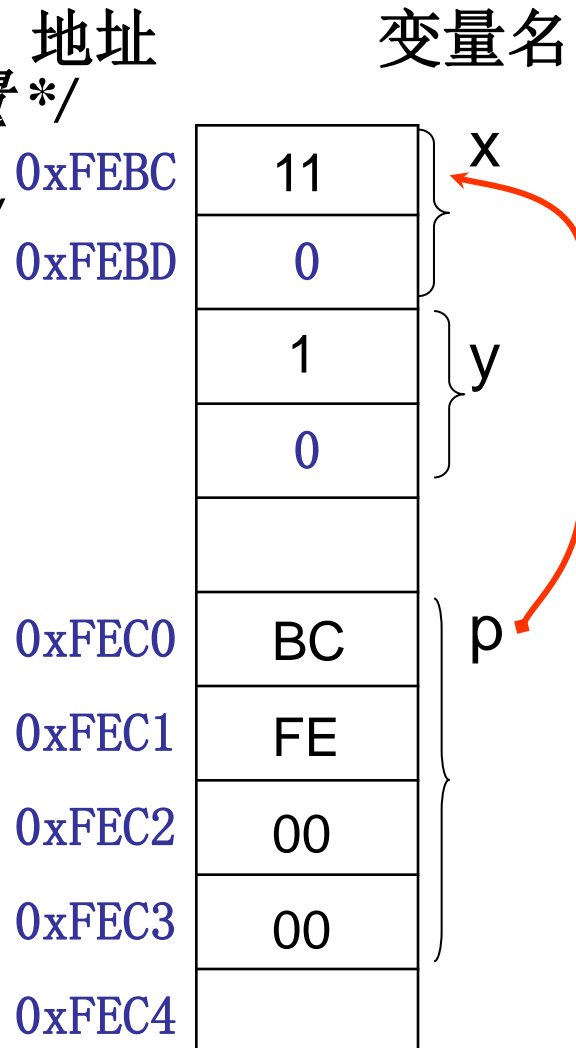


# 指出下面程序段的输出结果

```
short x=1, y=2, *p; /* p是short型指针变量 */  
p=&x; /* 将x的地址给p, 即指针p指向x */  
y=*p; /* *p==>x */  
*p += 10; /* x+=10 */  
printf("x=%hd,y=%hd",x,y);
```

输出?

x=11,y=1



## 思考：为什么指针有类型？

```
short x=0x1020, *p1=&x;
```

```
char *p2=&x;
```

```
int a,b;
```

```
a=*p1;
```

```
b=*p2;
```

```
printf(“%d,%d”,a,b) ;
```

4128,32

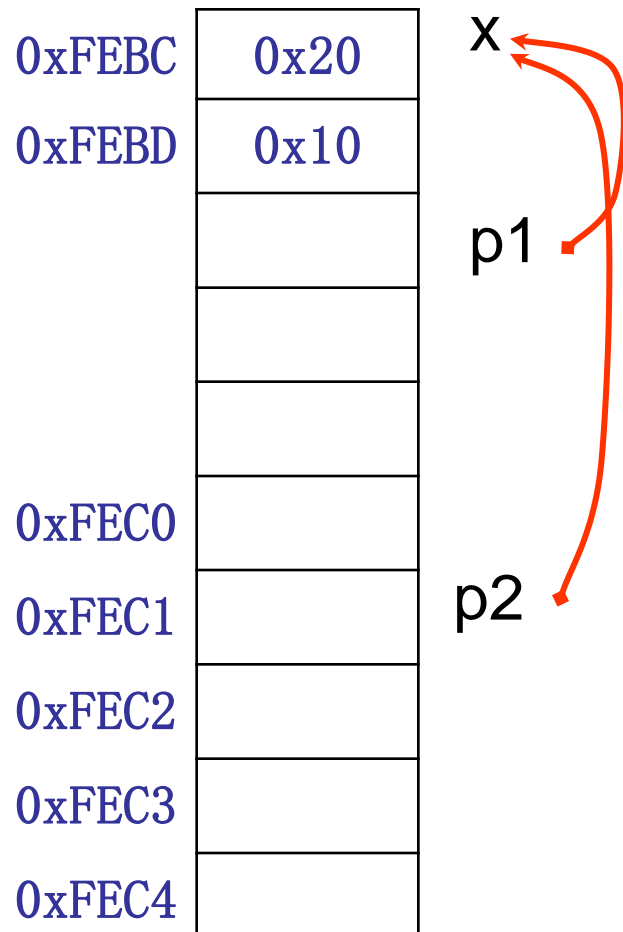
```
b=*(p2+1);
```

```
printf(“%d”,b) ;
```

16

地址

变量名





## 8.1.3 指针的使用

---

1) 建立指针变量与被指变量间的指向关系。

取地址运算符 **&**

2) 通过指针变量来间接访问和操作指针所指的变量。

间访运算符 **\***

# 1、取地址运算符-单目 &

## &操作数

返回其后操作数的内存地址，操作数必须是一个左值表达式。例如

*int b;    char s[20];*

以下表达式哪些正确？ 指出正确表达式的类型。

*&b* 、 *&s[2]*、 *&s*

*int \** , *char \**,    *×*

如果右操作数的类型是T，

则表达式&操作数的类型是 *T \**





## 2、指针的赋值

---

```
int x, *p;
```

```
float a[5], *pf;
```

```
p=&x;    /* 取整型变量x的地址赋给整型指针变量p,  
          指针p指向变量x    */
```

```
pf=a;    /* 等价于pf=&a[0];  
          指针pf指向a数组的第一个元素a[0]。  
          数组名 a 的数据类型是float *    */
```



# 悬挂指针

指出下面程序段的错误:

```
int x,*p; /* 说明p是一个整型指针变量, 其值不确定. */  
x=25;  
*p=x;    /* 使用悬挂指针, 错! */  
scanf("%d",p); /* 使用悬挂指针, 错! */
```

- 指针的声明只是创建了指针变量, 获得了指针变量的存储, 但并没有给出指针变量指向那个具体的变量, 此时指针的值是不确定的随机值, 指针处于“无所指”的状态。称为悬挂指针（野指针）。
- 不能使用悬挂指针

### 3、间访运算符---单目\*

**\*操作数** /\* 返回 操作数所指地址处的变量值。\*/

设有声明：

```
char ch=' a' , *pc=&ch;
```

```
*pc=' b' ;
```

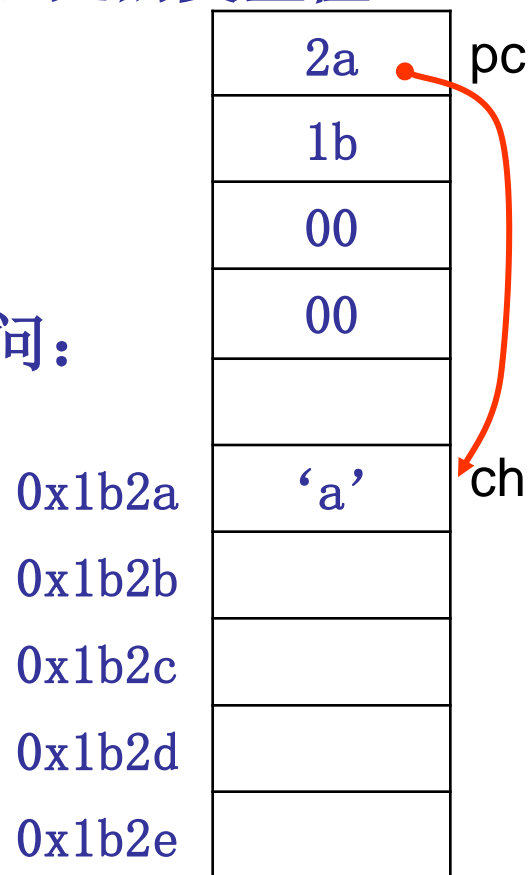
并且假设字符变量ch的地址是0x1b2a，试问：

**ch, pc, 以及\*pc的值是什么？**

**ch:** 'b' (直接访问)

**pc:** 0x1b2a

**\*pc:** 'b' (间接访问)



声明指针的目的是希望通过指针实现对内存中变量的快速访问。



## 8.2 指针运算

---

指针允许进行算术运算、赋值运算和关系运算。通过指针运算可以快速定位到指定的内存单元，提高程序的执行效率。

## 8.2.1 指针的算术运算

(+, -, ++, --, +=, -=)

### 1、指针的移动

指针的移动指指针在原有位置的基础上，通过加一个整数实现指针的**前移**（地址增大的方向）或者通过减一个整数实现指针的**后移**。

$p \pm k$   
指针 ↑      ↑ 整型

$p$  前移/后移  $k$ 个元素

已知声明如下，指出下面表达式的值。

(各题的表达式相互无关)

```
int x[5]={1, 3, 5, 7, 9}, *px=&x[1];
```

(1) ++\*px                      4

(2) \*++px                      5

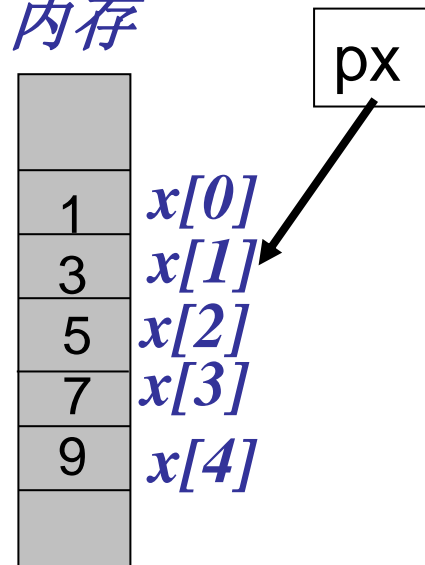
(3) \* (--px)                    1

(4) \*(px--)                    3

(5) \*px++                      3

(6) px+=2, \*px                7

内存



# 所有运算符的优先级和结合性规则

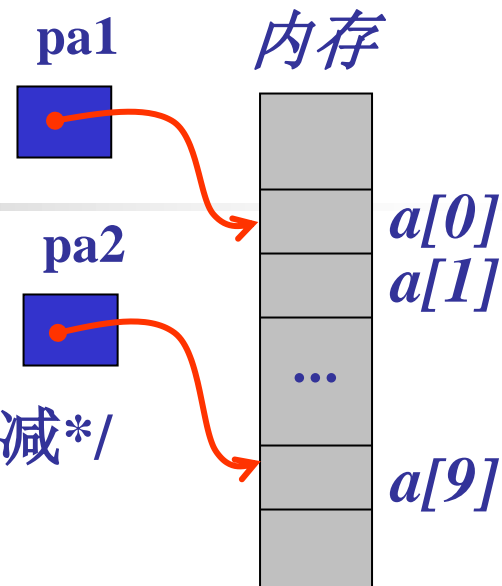
优先级 (从高到低)	运算符	运算符名称	结合性
1	( ) [ ] > . ~	圆括号 下标 间接引用结构体成员	左结合
单目运算符	!	逻辑非	
	~	按位取反	
	++、--	自增、自减	
2	+、-	取正、取负	右结合
	(数据类型) &、* sizeof	强制类型转换 取地址、间接引用 数据长度	
3	*/、/、%	乘、除、求余数	左结合
4	+、-	加、减	左结合
5	<<、>>	左移、右移	左结合
6	<、> >=、<=	大于、小于 大于等于、小于等于	左结合
7	=、!=	等于、不等于	左结合
8	&	按位与	左结合
9	^	按位异或	左结合
10		按位或	左结合
11	&&	逻辑与	左结合
12		逻辑或	左结合
13	? :	条件	右结合
14	=、+=、-=、*=、/=、%=、>>=、 <<=、&=、^=、 =	赋值	右结合
15	,	逗号	左结合

## 2、两个指针相减

两个指针可以相减

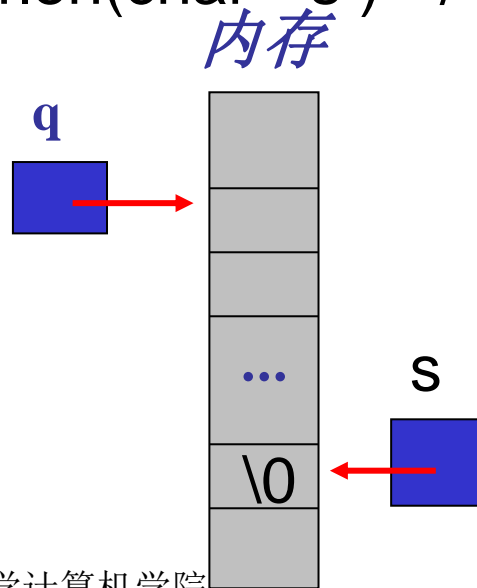
$pa2 - pa1$  /\*等于所指元素的下标相减\*/

指向同一数组元素的指针



int strlen(char s[ ]) /\* 等价于 int strlen(char \*s) \*/

```
{ char *q=s;
  while(*s) s++;
  return (s-q);
}
```







(1) 同类型的指针可以直接赋值。如：

```
int a[3]={1, 2, 3}, x, *p, *q;
```

```
p=a; q=&x;
```

(2) 不同类型的指针必须使用类型强制转换再赋值。

```
long x;
```

```
char *p;
```

```
p = (char *) &x;
```



## 9.2.3 指针的关系运算

---

<, <=, >, >=, ==, !=

只限于同类型指针，  
不同类型指针之间的关系运算被视为非法操作。

## 8.3 指针作为函数的

### 8.3.1 形参指针对实参变量的

例 形参的修改无法影响实参变量的

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

```
void swap(int x,int y)
```

```
{ int t;
```

```
    t=x;x=y;y=t;
```

```
}
```

```
int main(void)
```

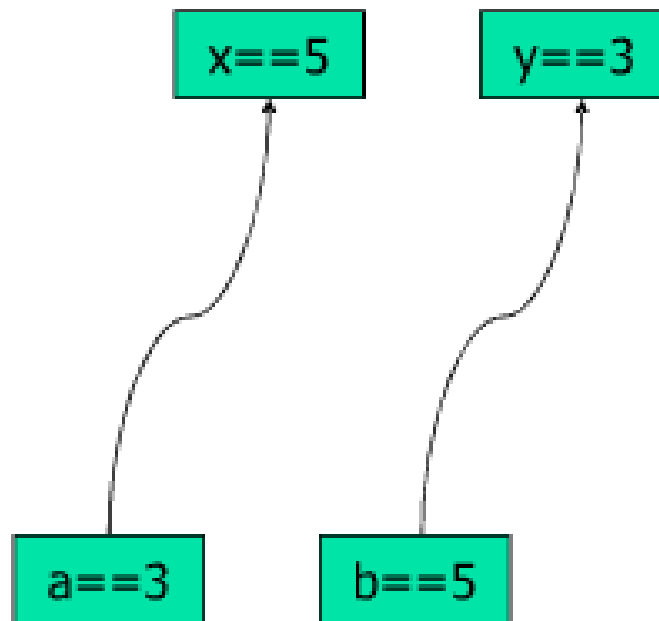
```
{ int a=3,b=5;
```

```
    swap(a,b);
```

```
    printf(“a=%d,b=%d\n”,a,b);
```

```
    return 0;
```

```
}
```



例 以指针作为函数的参数实现变量值的交换。

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

```
void swap(int *px, int *py)
```

```
{ int t;
```

```
    t=*px;*px=*py;*py=t;
```

```
}
```

```
void main(void)
```

```
{ int a=10, b=20, c=30;
```

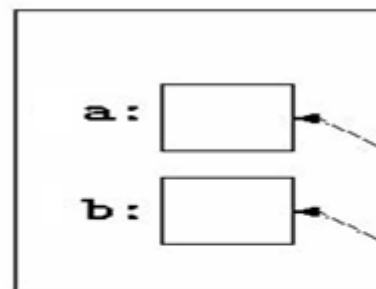
```
    swap(&a, &b);
```

```
    swap(&b, &c);
```

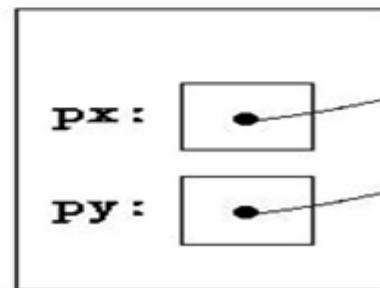
```
    printf( "a=%d, b=%d, c=%d\n ", a, b, c );
```

```
}
```

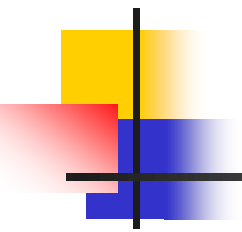
in caller:



in swap:



- 
- 
- 改变主调函数中变量的值
  - 使函数送回多个值



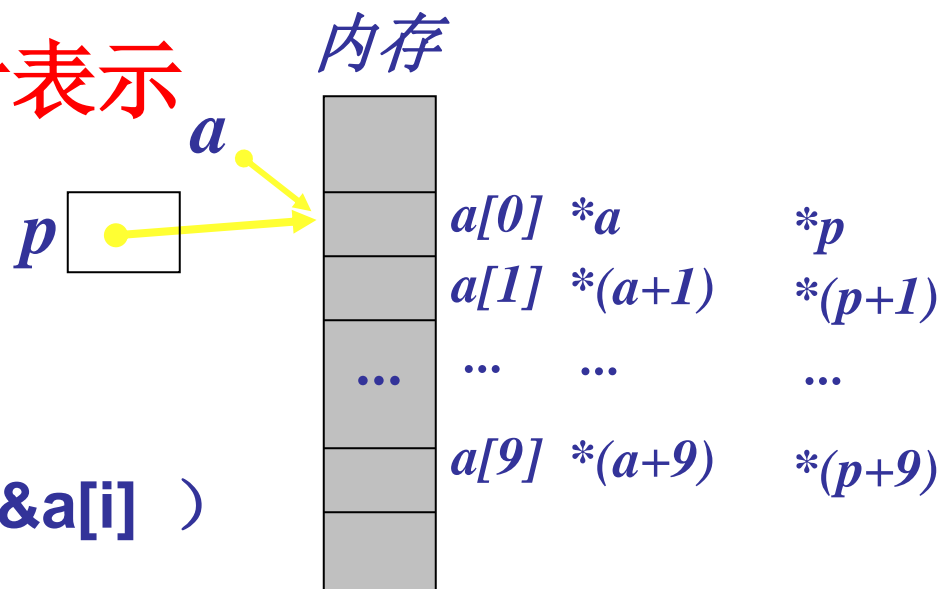
```
// implicit returned values:  
void sum(int x,int y,int *result)  
{  
    *result=x+y;  
}  
  
// the caller  
int s;  
sum(3,4,&s);
```

## 8.4 数组的指针表示

数组元素既可以用下标表示，也可以用指针表示。

### 8.4.1 一维数组的指针表示

```
int a[10], *p=a;
```



### 数组元素的表示

下标法  $a[i]$  (地址为  $\&a[i]$ )

指针法 { 数组名  $*(a+i)$  (地址为  $a+i$ )

指针变量  $*(p+i)$  (地址为  $p+i$ )

已知: #define N

int a[N],\*p, i;

p=a;

- 输入数组的全部元素

(1) for(i=0;i< N;i++)

scanf("%d", \_\_\_\_\_) ;

(2) for( ;p<a+ N;p++)

scanf("%d", \_\_\_\_\_ p ) ;

&a[i]

a+i

p+i

p++

a++







已知: `int a[10],*p, i;`

`p=a;`

• 输出数组的全部元素

`for(i=0;i<10;i++)  
printf(“%d”, _____ );`

`for( ; p<a+10;p++)  
printf(“%d”, _____*p );`

`a[i]  
*(a+i)  
*(p+i)  
*p++`

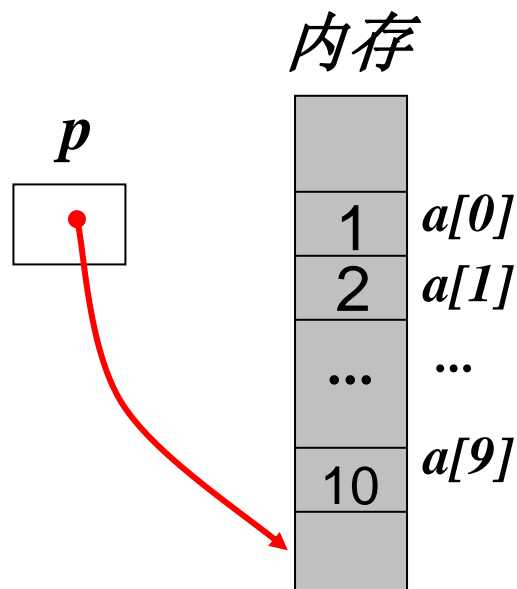
• 输出数组*a*的全部元素( 正确否? )

```
int a[10],*p , i;
```

```
for( p=a,i=1; i<=10; i++){  
    *p=i;  
    p++;  
}
```

```
while(p<a+10)
```

```
    printf(“%d”, *p++ );
```



## 8.4.2 一维数组参数的指针表示

```
void sort ( int a[ ], int n)
{
    .....
}
```

↑  
int \*a

形参 { 不指定长度的数组  
指针

实参 { 数组名  
指向数组元素的指针变量

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

```
#define N 10
```

```
void BubbleSort ( int a[ ],int n) // 形参为不指定长度的数组
```

```
{  
    int i, j, t;  
    for (i=1; i<n; i++) /* 共进行n-1轮"冒泡" */  
        for (j=0; j<n-i ; j++) /* 对两两相邻的元素进行比较 */  
            if (a[j]>a[j+1] ) { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }  
}
```

```
int main ( )
```

```
{ int x[N], i;  
    printf (" please input %d numbers: \n ", N);  
    for (i=0; i<N; i++) scanf(" %d ", &x[i]);  
    BubbleSort (x , N ) ; // 实参为数组名  
    printf (" the sorted numbers: \n ");  
    for (i=0; i<N; i++) printf("%d ", x[i]);  
    return 0;
```

```
}
```

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

```
#define N 10
```

```
void BubbleSort ( int *a,int n) // 形参为指针
```

```
{  
    int i, j, t;  
    for (i=1; i<n; i++) /* 共进行n-1轮"冒泡" */  
        for (j=0; j<n-i ; j++) /* 对两两相邻的元素进行比较 */  
            if (a[j]>a[j+1] ) { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }  
}
```

```
int main ( )
```

```
{ int x[N], i;  
    printf (" please input %d numbers: \n ", N);  
    for (i=0; i<N; i++) scanf(" %d ", &x[i]);  
    BubbleSort (x , N ) ; // 实参为数组名  
    printf (" the sorted numbers: \n ");  
    for (i=0; i<N; i++) printf("%d ", x[i]);  
    return 0;
```

```
}
```

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

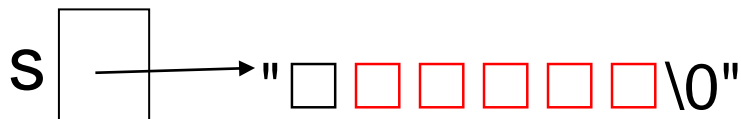
```
#define N 10
```

```
void BubbleSort ( int *a,int n) // 形参为指针
```

```
{  
    int i, j, t;  
    for (i=1; i<n; i++) /* 共进行n-1轮"冒泡" */  
        for (j=0; j<n-i ; j++) /* 对两两相邻的元素进行比较 */  
            if ( *(a+j)>*(a+j+1) ) {  
                t=*(a+j); *(a+j)=*(a+j+1); *(a+j+1)=t;  
            }  
}  
int main ( )  
{ int x[N], i,*p=a;  
    printf (" please input %d numbers: \n ", N);  
    for (i=0; i<N; i++) scanf(" %d ", &x[i]);  
    BubbleSort (p , N ) ; // 实参为指针变量  
    ..... ;  
}
```

# 库函数strlen(s)递归实现

- 字符串看成 “一个字符后面再跟一个字符串”

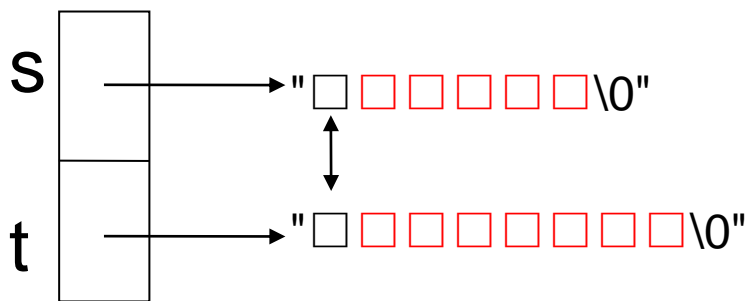


```
int strlen(char s[ ])
{
    if(s[0]=='\0')
        return 0;
    else
        return(1+strlen(s+1));
}
```

```
int strlen(char *s)
{
    if(*s=='\0')
        return 0;
    else
        return(1+strlen(++s));
}
```

## 【例12.2】 用递归实现标准库函数strcmp(s, t)

- 字符串看成“一个字符后面再跟一个字符串”



```
int strcmp_rec(char *s, char *t)
{
    if(*s != *t || *s == '\0')
        return(*s - *t);
    else
        return(strcmp(++s, ++t));
}
```

递归结束条件:

- (1) 两个串首字符不等
- 或
- (2) 两个串是空串



# 例 验证密码

```
int main(void)
{
    char pw[]="1234",s[20];
    int count=3;
    do {
        printf("Input password\n") ;
        gets(s);
        count--;
    } while(strcmp_rec(pw,s)&&count));
    if(strcmp_rec(pw,s))
        return 1;

    ....
    return 0;
}
```

## 例8.15 计算两个向量的数量积（向量的数乘）

■ **向量：** n个实数组成的有序数组

$$\mathbf{a} = (a_1, a_2, \dots, a_n)$$

$$\mathbf{b} = (b_1, b_2, \dots, b_n)$$

■ **向量的数量积**等于它们对应分量的乘积的和  $\sum_{i=1}^n a_i b_i$

■ 将计算两个向量的数量积定义成函数scalar

**形式参数：** 向量a, double \*a

向量b, double \*b

向量的维数n, int n

**返回值：** 数量积

## 例8.15 计算两个向量的数量积（向量的数乘）

$$\sum_{i=1}^n a_i b_i$$

```
double scalar(double *a,double *b,int n)
```

```
{
```

```
    double s,i;
```

```
    for(s=0,i=0;i<n;i++) {
```

```
        s += *a * *b;
```

```
        a++;
```

```
        b++;
```

```
    }
```

```
    return s;
```

```
}
```

} s += ( (\*a++) \* (\*b++) );

### 例8.15 计算两个向量的数量积（向量的数乘）。

```
#include <stdio.h>
#define SIZE 3
double scalar(double *a,double *b,int n);
void main(void)
{
    double x[SIZE],y[SIZE],*px,*py;
    for(px=x;px<x+SIZE;px++) /* x+SIZE是元素x[SIZE]的地址 */
        scanf("%lf",px);
    getchar();
    for(py=y;py<&y[SIZE];) /* &y[SIZE]就是y+SIZE */
        scanf("%lf",py++);
    px=x;py=y;
    printf("the scalar is %f\n", scalar(px,py,SIZE));
}
```

### 8.4.3 用指向基本数组元素的指针表示多维数组

```
#define M 3
#define N 2
int a[M][N]={ {1, 3},
               {4, 6},
               {7, 9}
             };

```

```
int *p;
```

数组a的元素的数据类型为int，则指向数组a基本元素的指针为int\*。

- (1) 使该指针指向一个数组元素;
- (2) 用该指针表示数组元素;
- (3) 对该指针还可以施行运算，使它指向数组中的其它元素。

思考：如何用指针p逐行输出数组a的所有元素？

```
for(p=a[0]; p<a[0]+M*N;p++){
    if (!( (p-a[0])%N )) printf("\n");
    printf("%5d",*p);
}

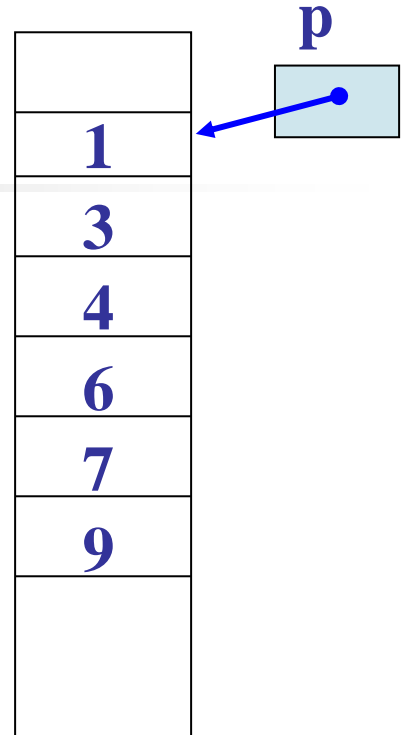
```

```
int a[M][N]={ {1, 3},
               {4, 6},
               {7, 9}
             };
```

```
int *p;
```

如何用指针p逐行输出数组a的所有元素？

```
p=a[0]; /* 等价于 p=&a[0][0] */
for(i=0; i<M;i++) {
    printf("\n");
    for(j=0; j<N;j++)
        printf("%5d", *(p+i*N+j) );
}
```



## 8.5 指针数组

### 8.5.1 指针数组的声明及使用

#### 1. 指针数组的声明

**cv T \* cv 标识符[常量表达式]={初值表};**

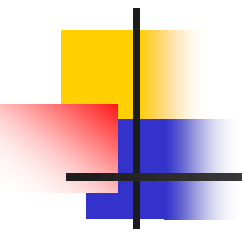
数据类型

指针数组名

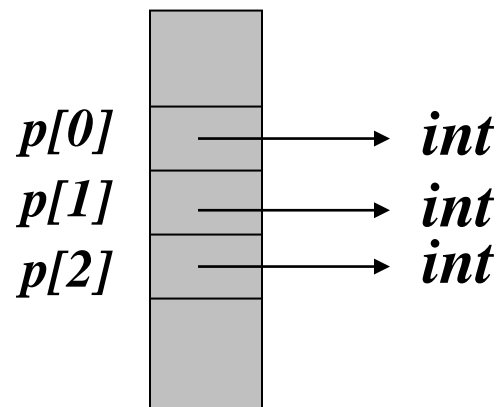
可选项：可以是**const**等

指针数组的大小

可选项，用于初始化指针数组。



```
int *p[3]; /* p是一个有 3 个元素的整型指针数组  
           即每个元素是指向整型变量的指针 */
```

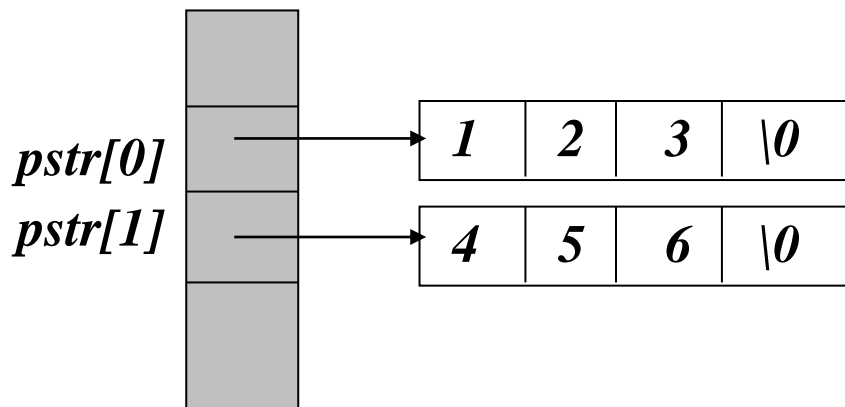






```
char *pstr[2]={ “123” , “456” };
```

*/\* pstr 是一个有 2 个元素的字符指针数组  
pstr[0]指向字符串 “123”  
pstr[1]指字符串 “456” \*/*





```
const int x=1,y=2;  /* x, y 值不可更改 */
```

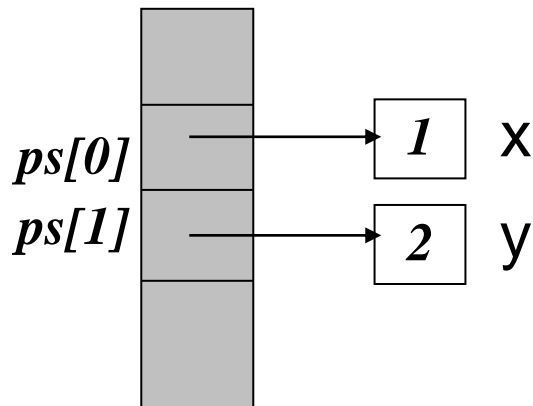
```
const int *ps[2];
```

/\* ps是由2个指向整型常量的指针组成的指针数组;  
简称为指向常量的整型指针数组。\*/

```
ps[0]=&x; ps[1]=&y; /* ps[0]指向x, ps[1]指向y */
```

```
*ps[0]=4; /* 错误,指针数组ps的元素所指的对象不能修改 */
```

```
ps[0]=&y; /* 正确,ps的元素本身可以修改,使之指向其他对象*/
```





---

```
const char * const p[2]={ “abc”, “123”};
```

/\* **p**是一个指向常量字符串的字符型指针常量数组。**p**的元素本身不能修改，**p**的元素所指对象也不能修改。\*/



## 2. 指针数组的使用

---

指针数组的应用:

描述二维数组 (数值型二维数组, 字符串数组)

尤其是每行元素个数不相同的二维数组  
(如三角矩阵, 有不同长度的字符串组成的字符串数组)

# 例1 输入N本书名。

如何存储读入的N本书名？

(1) 二维数组

(2) 指针数组

```
#define N 3
int main( )
{
    int i;
    char *s[N];
    for(i=0;i<N;i++)
        fgets(s[i], 20, stdin);
    return 0;
}
```

如何  
解决

错误，使用了悬挂指针



```
#define N 3
```

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

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

```
int main( )
```

```
{
```

```
    int i;
```

```
    char *s[N], t[50];
```

```
    for(i=0;i<N;i++) {
```

```
        gets(t);
```

```
        s[i] = (char *)malloc(strlen(t)+1);
```

```
        strcpy(s[i],t);
```

```
    }
```

```
    .....
```

```
}
```

在该头文件中给出函数原型:

```
void *malloc(unsigned  
size);
```

分配**size**字节的存储区。  
返回所分配单元的起始地址。  
如不成功, 返回**NULL(空指针)**



# 动态存储分配函数 **malloc**

- 动态存储分配函数是C的标准函数，函数的原型声明在头文件**<stdlib.h>**中给出。

**void \* malloc(unsigned size);**

**功能：** 分配**size**字节的存储区。

**返回值：** 所分配单元的起始地址。如不成功，返回**NULL**

# 无值型指针与空指针

类型为**void \***的指针称为无值型指针或**void**指针。不能对**void**指针执行间接访问操作，即对**void**指针施行“**\***”操作属于非法操作。

指针值为**0**的指针称为空指针，**0**在**C**中往往用符号常量**NULL**表示并被称为空值。

```
scanf ( " %d" ,&n) ;
```

```
/* 建立大小为n的int型数组 */
```

```
p=(int *)malloc( n*sizeof(int));
```

```
if(p==NULL)  exit(-1);
```



## 例 用字符指针数组构造程序的菜单系统 menu.c

/\* 菜单函数，函数返回值是整形，代表所选的菜单项 \*/

```
int MenuSelect(void )
```

```
{  char *menu[ ]={
```

```
    "1:Enter record" ,
```

```
    "2:Search record on name" ,
```

```
    "3>Delete a record"
```

```
    "4:Add a record" ,
```

```
    "5:Quit"      };
```

```
int i;
```

```
system("cls"); /* 清屏 */
```

```
do {
```

```
    puts(menu[i]);
```

```
    printf("\n Enter your choice(1-5):");
```

```
    scanf("%d",&i); /* 输入选择项 */
```

```
} while( i<1 || i>5 ); /* 选择项不在1-5之间重输 */
```

```
return c; /* 返回选择项，调用函数根据该数调用相应的函数 */
```

```
}
```

## 例 判断字符串是否关键字。

```
/* 若s是关键字, 函数返回1; 否则,返回0 */
int iskey(char *s)
{
    char *keyword[]={ /*keyword中的每个元素都指向特定的关键字字符串*/
        "auto", "_Bool", "break", "case", "char", "_Complex",
        "const", "continue", "default", "restrict", "do", "double",
        "else", "enum", "extern", "float", "for", "goto",
        "if", "_Imaginary", "inline", "int", "long", "register",
        "return", "short", "signed", "sizeof", "static", "struct",
        "switch", "typedef", "union", "unsigned", "void", "volatile",
        "while", "" };
    int i;
    for(i=0;*keyword[i]!='\0';i++) /*将标识符s与每个关键字比较, 是返回1*/
        if(!strcmp(s, keyword[i])) return 1;
    return 0; /*否返回0*/
}
```

## 例 判断字符串是否合法的用户标识符。

```
/* t是标识符, 函数返回1; 否则, 返回0 */
int isid(char * t)
{
    char *s=t;
    if( !isalpha(*t) && *t!= '_' ) /* 首字符不是字母或下划线, 返回0 */
        return 0;
    for(t++;*t!='\0';t++) /* 后续字符不是字母、数字、或下划线, 返回0 */
        if(!(isdigit(*t)||isalpha(*t)||*t=='_'))
            return 0;
    if(iskey(s)) /* s是关键字, 返回0 */
        return 0;
    return 1;
}
```



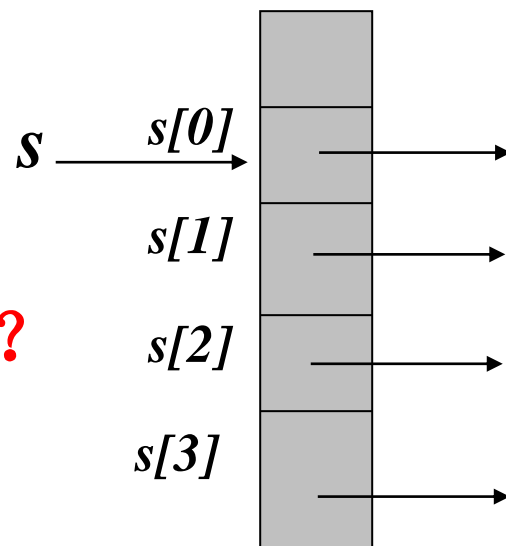
```
char *s[4];
```

问：s[0][0]，s[0]，s类型分别是什么？

s[0][0] 《==》 \*s[0]      类型 char

s[0]，指针数组的元素，类型 char \*

s，指针数组名，指向s[0]，类型：char \*\*（字符型的二级指针）



二级指针的应用：作函数的形参

## 8.5.2 多重指针

```
int x, *p=&x;
```

x有地址 (&x)，那 p有地址吗？

&p 的类型？

&p —> p □ —> x □

欲保存p的地址，应该定义一个什么类型的变量？

```
int x, *p=&x, **pp=&p; /* pp是int的二级指针 */
```

pp □ —> p □ —> x □

```
**pp=5; /* x=5 */
```

```
*p=8; /* x=8 */
```

## 定义函数strsort对字符串按字典序升序排序

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

```
void strsort ( char *s[ ] , int size )
```

```
{
```

```
    char *temp;
```

```
    int i, j ;
```

```
    for(i=0; i<size-1; i++)
```

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

```
            if ( strcmp(s[j],s[j+1] ) >0 ) {
```

```
                temp=s[j];
```

*/\* 移动指针变量 \*/*

```
                s[j]=s[j+1];
```

```
                s[j+1]=temp;
```

```
            }
```

```
}
```



char \*\*s

```
#define N 3
```

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

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

```
void main( )
```

```
{
```

```
    int i;
```

```
    char *s[N], t[10];
```

```
    for(i=0;i<N;i++) {
```

```
        gets(t);
```

```
        s[i] = (char *)malloc(strlen(t)+1);
```

```
        strcpy(s[i],t);
```

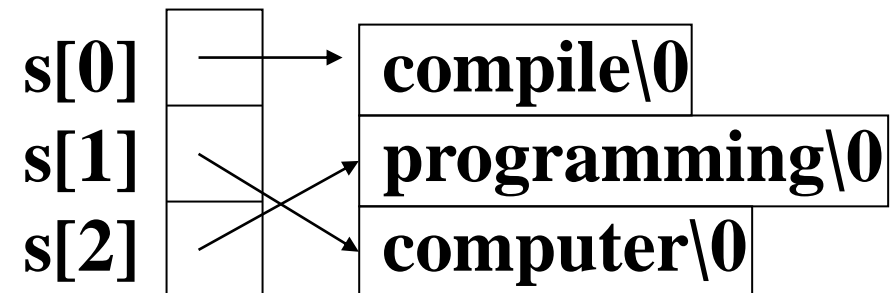
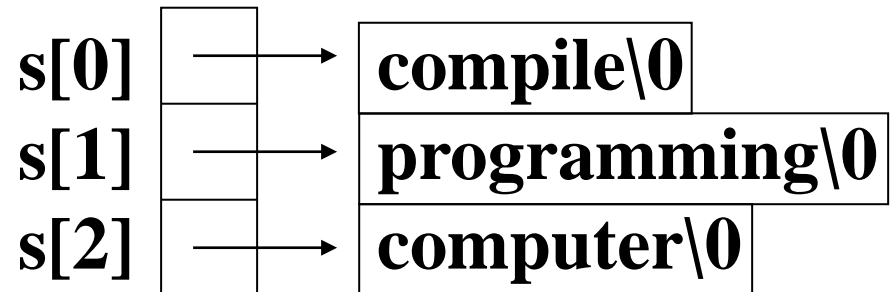
```
    }
```

```
    strsort(s,N);
```

```
    for(i=0;i<N;i++) puts(s[i]);
```

```
}
```

**例** 输入N本书名，排序后输出。



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

```
int main(void)
```

```
{
```

```
    int n,sum,i;
```

```
    scanf("%d",&n);
```

```
    for(sum=0,i=1;i<=n;i++)
```

```
        sum+=i;
```

```
    printf("1+2+...+%d=%d\n",n,sum);
```

```
    getchar();
```

```
    return 0;
```

```
}
```

## 户输入

所需的数据何时输入

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

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

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

```
{
```

```
    int n,sum,i;
```

```
    if(argc!=2) {
```

```
        printf("Command line error!\n");
```

```
        return -1;
```

```
    }
```

```
    n=atoi(argv[1]);
```

```
    for(sum=0,i=1;i<=n;i++)
```

```
        sum+=i;
```

```
    printf("1+2+...+%d=%d\n",n,sum);
```

```
    return 0;
```

```
}
```





## 8.6 带参数的main函数

### 8.6.1 命令行参数

- ▶ 在命令行中给定的参数就是命令行参数。

`sum_arg 4`

(以上`sum_arg`参数是必须的，否则不能运行)

- ▶ 命令行的参数由谁来接收

运行程序时操作系统将命令行参数传给main函数的形式参数

## 带参数的main函数的一般形式

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

Diagram illustrating the parameters of the `main` function:

- An arrow points from the number `2` to the parameter `argc`, indicating the number of arguments.
- An arrow points from the array `argv` to the example array `= { "sum", "10" }`, indicating that `argv` is an array of character pointers.

实参由命令行提供。

**argc: argument count,**

代表命令行中参数的个数（包括文件名）

**argv: argument value,** 为字符指针数组，

**argv[i]**指向命令行的第 **i+1** 个参数（字符串）的首字符

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

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

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

```
{
```

```
    int n,sum,i;
```

```
    if(argc!=2) {
```

```
        printf("Command line error!\n");
```

```
        return -1;
```

```
    }
```

```
    n=atoi(argv[1]);
```

```
    for(sum=0,i=1;i<=n;i++)
```

```
        sum+=i;
```

```
    printf("1+2+...+%d=%d\n",n,sum);
```

```
    return 0;
```

```
}
```

C:\> **sum** 11

argv[0]

argv[1]

命令行中参数的个数（包括文件名）

长度为**argc**的字符指针数组  
每个参数是一个字符串，  
有**argc**个字符串



C:\> sum 11

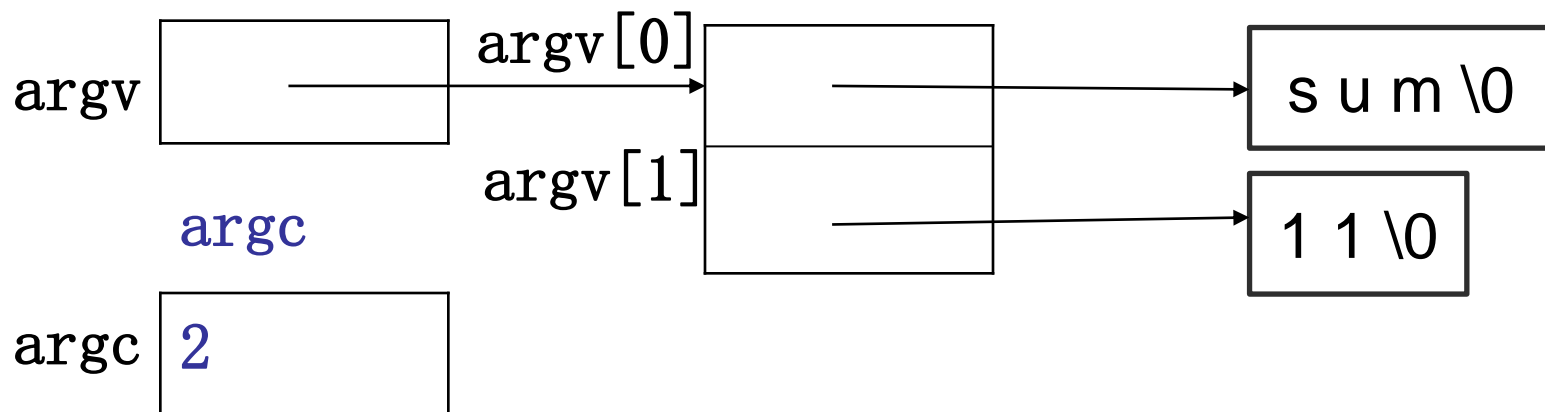
```
int main(int argc, char *argv[])
```

```
{
```

```
.....
```

```
}
```

char \*\*argv





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

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

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

```
{
```

```
    int n,sum,i;
```

```
    if(argc!=2) {
```

```
        printf("Command line error!\n");
```

```
        return 1;
```

```
    }
```

```
    n=atoi(*++argv);
```

```
    for(sum=0,i=1;i<=n;i++)
```

```
        sum+=i;
```

```
    printf("1+2+...+%d=%d\n",n,sum);
```

```
    return 0;
```

```
}
```

# 在集成开发环境下调试程序时命令行所带参数如何输入？(只输入文件名后的参数)

1) 在C::B下:

选择菜单“**project/set programs' arguments...**”，  
在“**Program arguments**”文本框中输入main函数的参数。

2) Dev: **Execute/Parameters...**

3) 在VC下:

工程（**project**）->设置（**setting**）->调试（**debug**）  
-> 程序变量（**program arguments**）



## 8.7 指针函数

在C语言中，函数返回的只能是值。这个值可以是一般的数值，也可以是某种类型的指针值。如果函数的返回值是指针类型的值，该函数称为指针函数。

类型 \*函数名（形参表）；

如：char \*strcpy(char \*t, const char \*s)；

函数strcpy是一个字符指针函数。即：该函数的返回值是字符指针。

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

```
char *strcpy( char *t, const char *s )
```

```
{
```

```
    char *p=t ;
```

```
    while(*t++ = *s++)
```

```
        ;
```

```
    return(p) ; /* 返回第1个串的首地址 */
```

```
}
```

```
int main( )
```

```
{
```

```
    char st1[40]=" abcd" , st2[ ]=" hijklmn" ;
```

```
    printf( "%s" , strcpy( st1,st2));
```

```
    return 0;
```

```
}
```



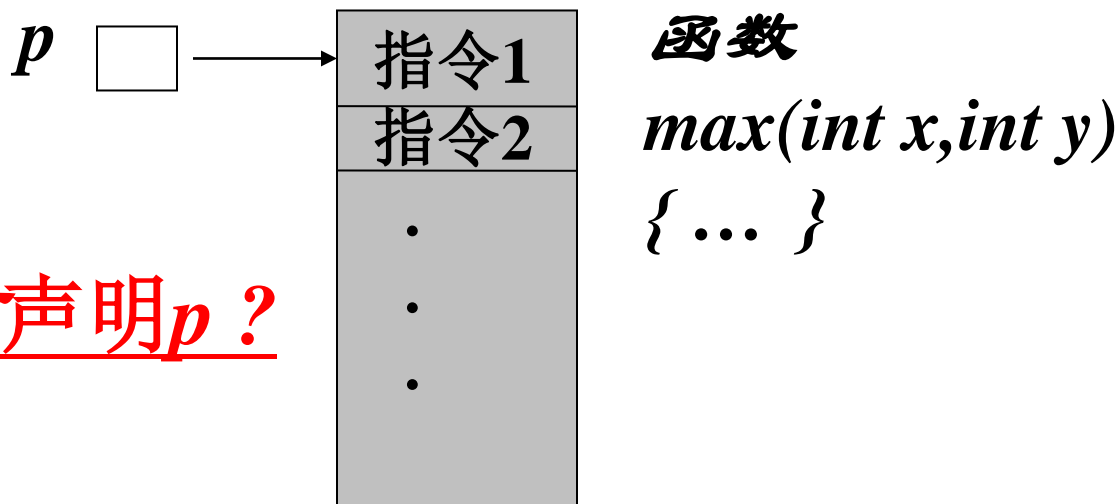


---

如果函数返回的指针指向一个数组的首元素，就间接解决了函数返回多值的问题。

## 8.8 函数的指针

每个函数都占用一段内存单元，有一个起始地址。



如何声明 $p$ ?

## 8.8.1 函数指针的声明

类型 (\*标识符) (形参表);

↑                      ↑                      ↑

所指函数的返回类型      函数指针名                      所指函数的形参的类型与个数

*int (\*p) ( int, int);*

*/\* p 是指向有两个 int参数的int函数的指针 \*/*

## 函数指针的应用举例

```
#include "stdio.h"
```

```
void f1(int x)
```

```
{    printf("x=%d\n",x); }
```

```
void f2(int x,int y)
```

```
{    printf("x=%d\ty=%d\n",x,y);}
```

```
int main(void)
```

```
{    void (*pf1)(int x);
```

```
    void (*pf2)(int x,int y);
```

```
    pf1=f1;
```

```
    pf2=f2;
```

```
    pf1(5);    /* 等价于(*pf1)(5);*/
```

```
    pf2(10,20); /* 等价于(*pf2)(10, 20);*/
```

```
    return 0;
```

```
}
```

## 函数指针的使用:

(1) 通过初始化或赋值使其指向特定的函数;

函数指针名=函数名;

(2) 通过函数指针来调用它所指的函数



# 定义一个通用的整数排序函数

- 既能实现升序排序，也能实现降序排序

函数名称: **sort**

函数参数:

**v**--待排序数组的首地址

**n**--数组中待排序元素数量

**comp**--指向函数的指针，用于确定排序的规则

函数返回值: 无

*/\*对指针v 指向的n个整数按comp规则排序 \*/*

```
void sort ( int *v, int n, int (*comp)(int, int) )
```

```
{
```

```
    int i, j ;
```

```
    for(i=1; i<n; i++)          /*冒泡法*/
```

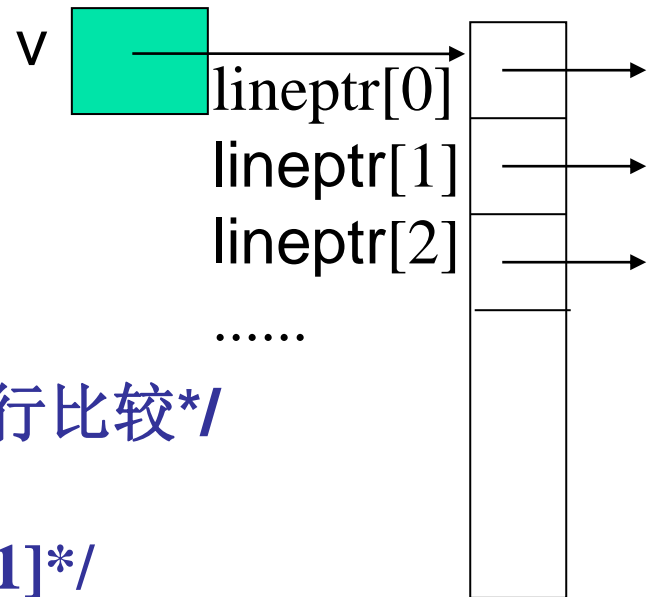
```
        for(j=0; j<n-i; j++)
```

```
            /*对v[j]和v[j+1]按照comp的规则进行比较*/
```

```
            if ( /*comp*/(v[j],v[j+1])>0 )
```

```
                swap(v, j , j+1) ; /*交换v[j]和v[j+1]*/
```

```
}
```



在main中



# 规则：按升序

```
int asc(int x, int y)
{
    if(x>y) return 1;
    else return 0;
}
```

// caller

```
int a[6]={4,6,3,9,7,2};
```

```
sort(a,6,asc);
```

// 思考：如果要降序排呢？如何定义描述规则的函数

自学教材例8-30

# 进阶：定义更通用的排序函数

- 能够对int、char、double、字符串、struct类型的数据排序。
- 既能实现升序排序，也能实现降序排序
- 函数参数
  - void \*v, 待排序数组首地址
  - int n, 数组中待排序元素数量
  - int size, 各元素的占用空间大小（字节）
  - int (\*fcmp)(const void \*,const void \*), 指向函数的指针，用于确定排序的规则
- **stdlib.h**中的标准库函数**qsort**----**万能数组排序函数**
  - void qsort(void \*base, int nelem, int width,  
int (\*fcmp)(const void \*,const void \*));
- P269（快速排序），p296例13.6（通用的排序函数定义）
- **思考**：如何调用qsort对字符串数组排序。