

# C语言与程序设计

## The C Programming Language



---

### 第8章 指针

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;

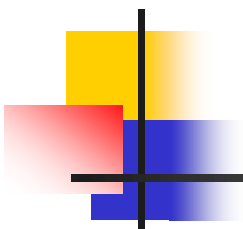
`short x, a[5];`

- 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]



➤ **指针**: 变量的地址

**&x**: 指针常量

**p**: 指针变量

➤ **指针变量**: 存放地址数据的变量。  
指针变量也是一种变量，也要占用一定的内存单元。

**指针的特殊之处在于它存放的是另一个变量所占存储单元的起始地址。**

地址	变量名	
0xFEBC	1	x
0xFEBD	0	
	2	y
	0	
		p
0xFEC0		
0xFEC1		
0xFEC2		
0xFEC3		
0xFEC4		

## 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?

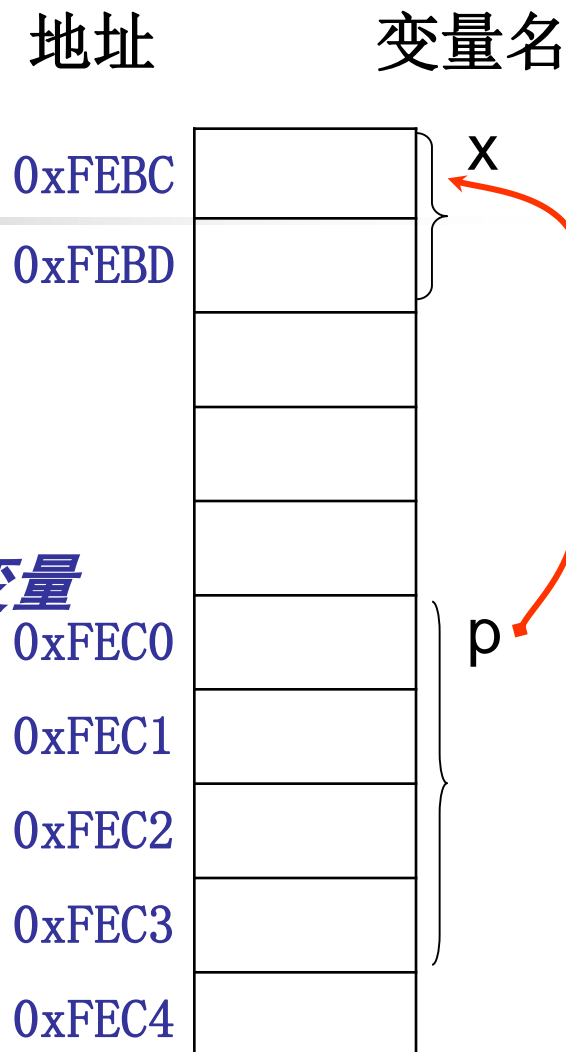
- 直接: 通过变量名存取 (或访问) 变量

`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。

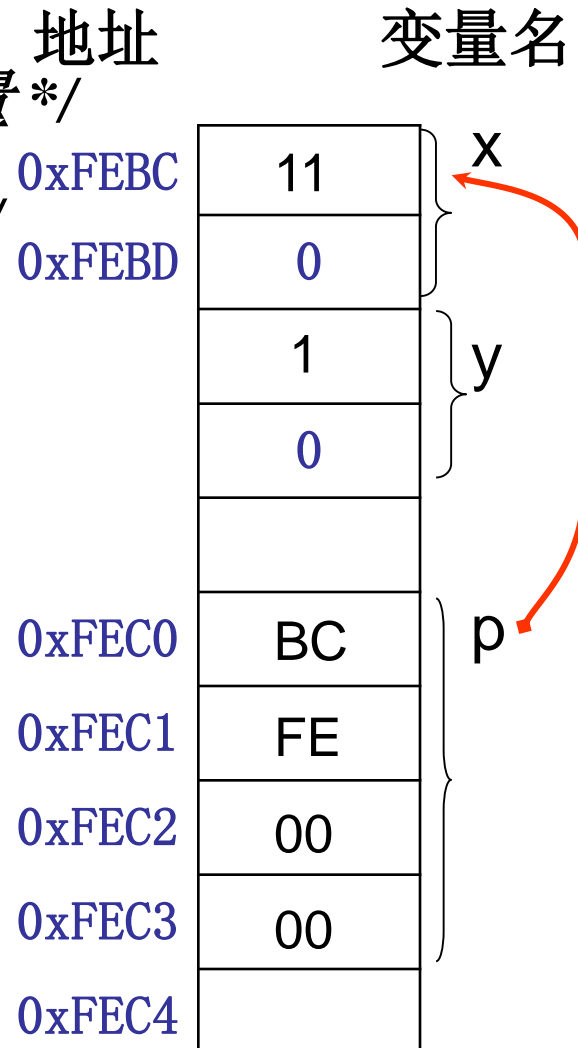


# 指出下面程序段的输出结果

```
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) ;
```

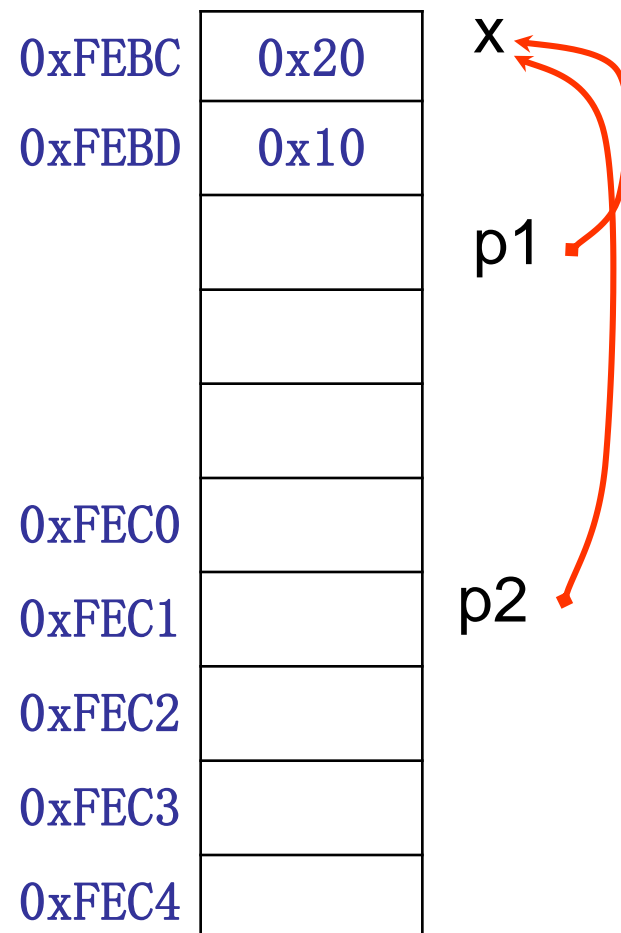
4128,32

```
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' ;
```

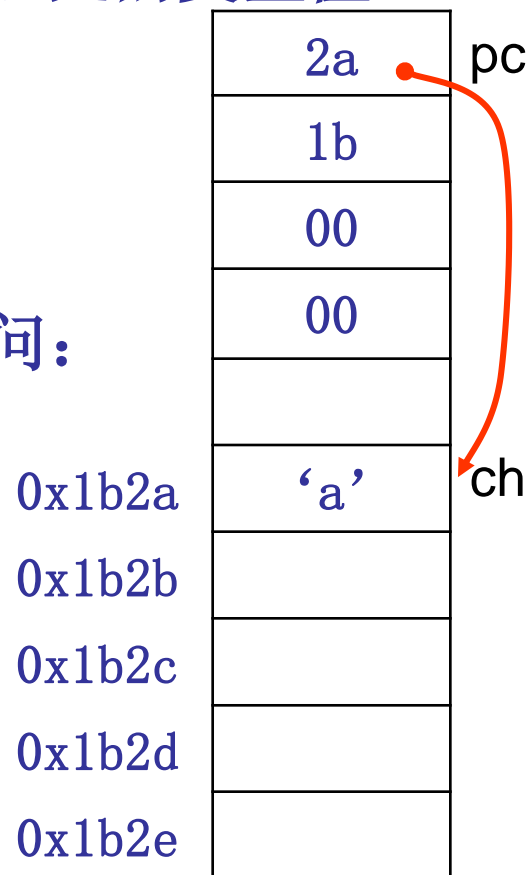
并且假设字符变量ch的地址是0x1b2a，试问：

**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

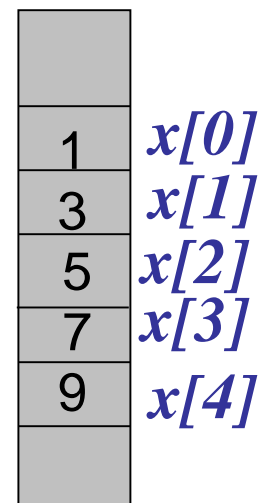
(3) \* (--px)                    1

(4) \*(px--)                    3

(5) \*px++                      3

(6) px+=2, \*px                7

内存



*x[0]*

*x[1]*

*x[2]*

*x[3]*

*x[4]*

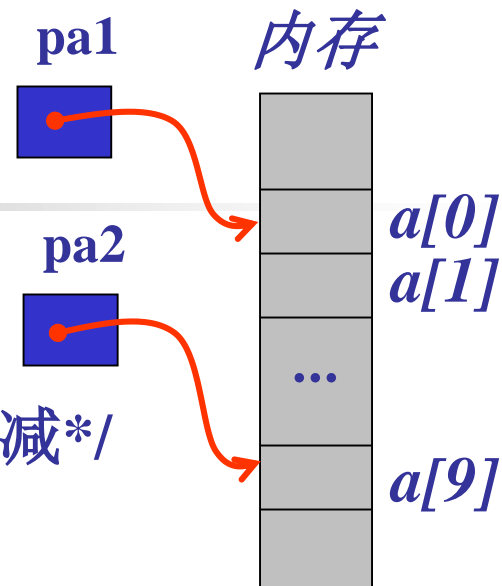
px

## 2、两个指针相减

两个指针可以相减

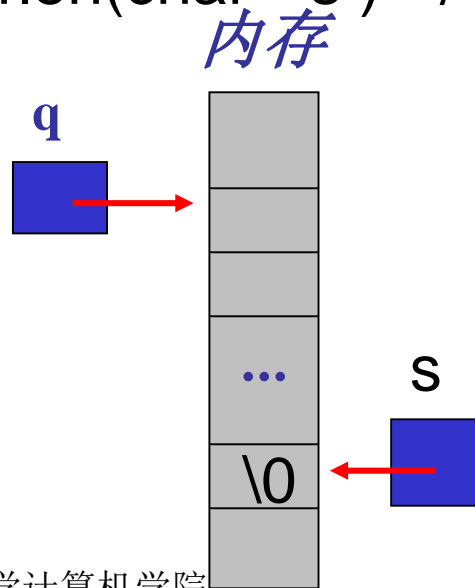
$pa2 - pa1$  /\*等于所指元素的下标相减\*/

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);
}
```

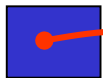




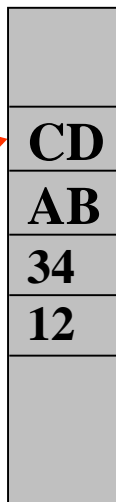
## 8.2.2 指针的赋值运算

例8.11 一个长整型数占4个字节，其中每个字节又分成高4位和低4位。试从长整型数的低字节开始，依次取出每个字节的高4位和低4位并以十六进制字符的形式进行显示。

char \*p



内存



$x=0x1234ABCD$

取低4位:  $(*p)\&0x0f$

取高4位:  $(*p>>4)\&0x0f$

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

```
int main(void)
```

```
{
```

```
    long x=0x1234ABCD,k;
```

```
    char *p=(char *)&x;    /* 类型强制，x被视为4字节字符类 */
```

```
    char up_half,low_half; /* up_half存高4位，low_half存低4位 */
```

```
    for(k=0;k<4;k++) {
```

```
        low_half=(*p)&0x0f; /* 取低4位 */
```

```
        if(low_half<10)
```

```
            low_half+='0'; /* 低4位二进制转换成字符 '0'-'9' */
```

```
        else
```

```
            low_half=(low_half-10)+'A'; /* 低4位二进制转换成字符 'A'-'F' */
```

```
        up_half=(*p>>4)&0x0f; /* 取高4位 */
```

```
        if(up_half<10)
```

```
            up_half|='0'; /* 高4位二进制转换成字符 '0'-'9' */
```

```
        else
```

```
            up_half=(up_half-10)+'A'; /* 高4位二进制转换成字符 'A'-'F' */
```

```
        printf("%c  %c\n",up_half,low_half);
```

```
        p++; /* 指向整型数x下一个字节 */
```

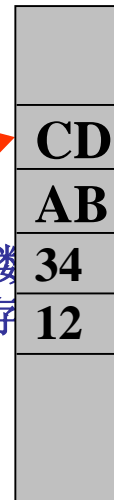
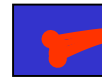
```
    }
```

```
    return 0;
```

```
}
```

char \*p

内存



$x=0x1234ABCD$

p所指 \*/

\*/



(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)
```

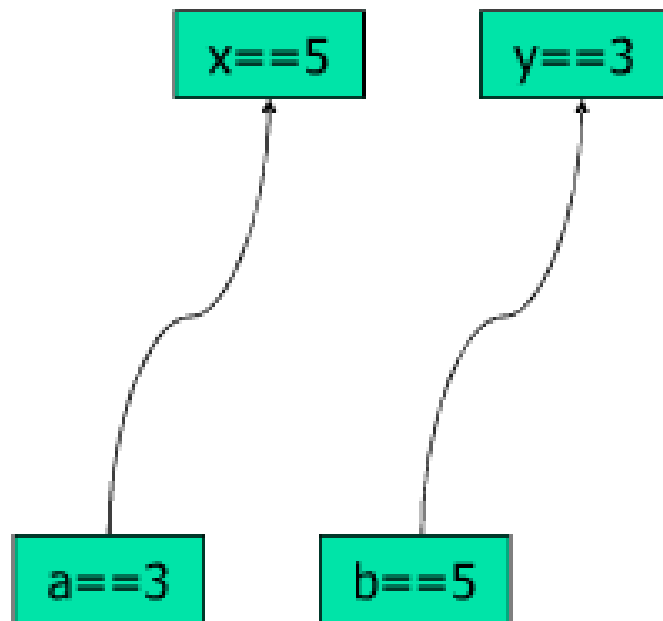
```
{ int a=3,b=5;
```

```
    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;
```

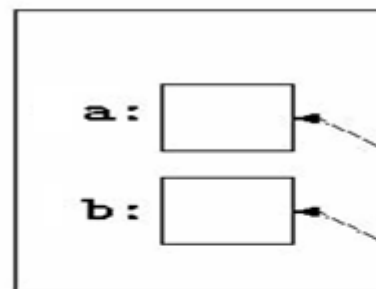
```
    swap(&a, &b);
```

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

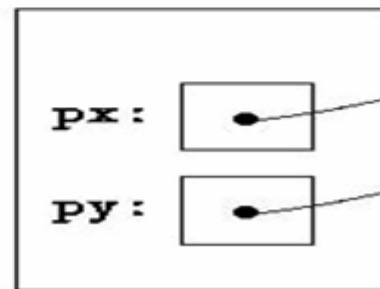
```
    printf( "a=%d, b=%d, c=%d\n ", a, b, c );
```

```
}
```

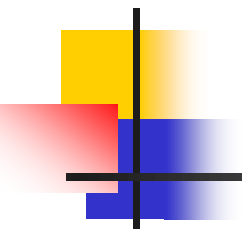
in caller:



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);
```

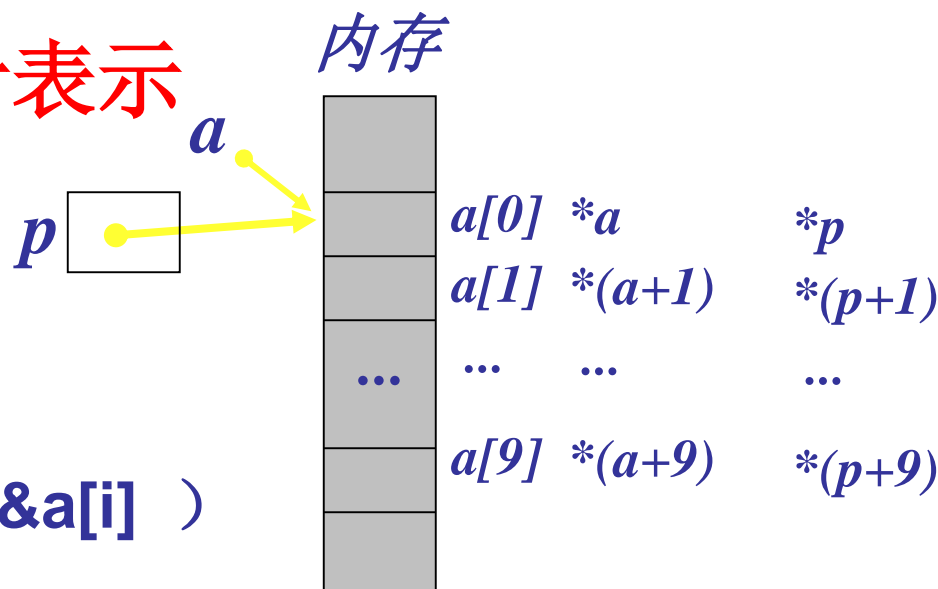


## 8.4 数组的指针表示

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++`

## 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>
```

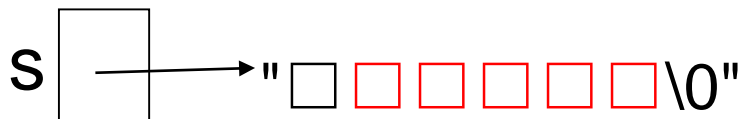
```
#define N 10
```

```
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”



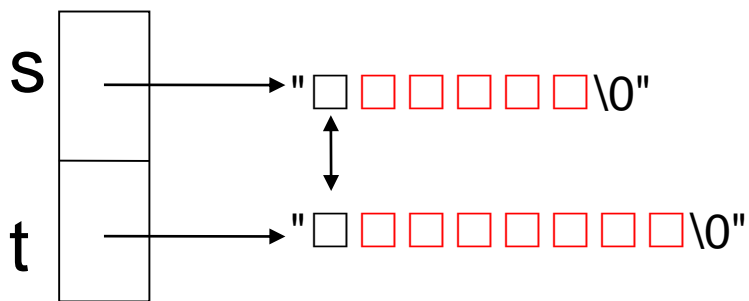
```
int strlen(char s[ ])
{
    if(s[0]=='\0')
        return 0;
    else
        return(1+strlen(s+1));
}
```

```
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_rec(++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;
}
```

### 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);
```

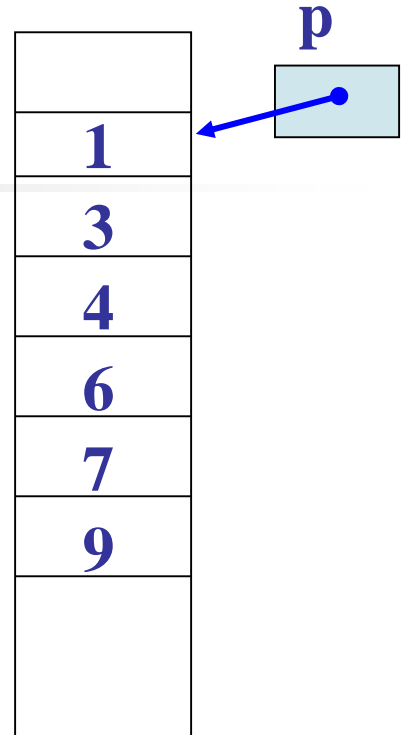
13:24 }

```
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) );  
}
```





## 8.4.4 高精度计算--- 超大整数的加法运算

**高精度运算**，是指参与运算的数(加数，减数，因子...) 范围大大超出了标准数据类型（整型，实型）能表示的范围的运算。例如。求两个**100**位的数的和，计算**100!**，都要用到高精度算法。



# 高精度计算主要解决的问题

---

高精度计算主要解决以下三个问题：

一、大数据（加数、结果）存储

二、运算过程

三、大数据的输入和输出



# 存储

---

用数组存储，每个数组元素存储1位（可以优化！），有多少位就需要多少个数组元素。个位在 $x[0]$ ，十位在 $x[1]$ , ...

**用数组表示数的优点：**每一位都是数的形式，可以直接加减；运算时非常方便。



# 运算过程

模拟列竖式计算两数相加（如**45+96**）。注意：

（1）运算顺序：从低位向高位运算；先低位后`高位；

（2）运算规则：相同位的两个数相加再加上进位，成为该位的和；这个和去掉向高位的进位就成为该位的值；如上例：

**5+6+0=11**，向前进一位**1**，本位的值是**1**；可借助%、/运算完成这一步；

**4+9+1=14**，向前进一位**1**，本位的值是**4**

（3）最后一位的进位：如果完成两个数的相加后，进位位值不为**0**，则应添加一位；

（4）如两个加数位数不一样多，则按位数多的一个进行计算；





# addBigNum

- **函数名称:** addBigNum
- **函数功能:** 两个大整数相加
- **函数参数:**
  - 输入 x--被加数
  - y--加数
  - n--x和y位数的较大者
  - 输出 z--和数
- **函数返回值:**
  - 和数的实际位数
- ⑩ **函数原型**  
`int addBigNum(int *x,int *y,int n, int *z);`



```
int addBigNum(int *x,int *y,int n, int *z)
```

```
{    int i,carry;
```

```
    for(carry=0,i=0;i<n;i++,z++,x++,y++){
```

```
        *z = *x + *y + carry;    /* 带进位的加法运算 */
```

```
        carry = *z /10;        /* 计算新的进位 */
```

```
        *z %= 10;              /* 计算该位 */
```

```
    }
```

```
    if(carry) { /* 和的最高位 */
```

```
        *z=carry;
```

```
        i++;
```

```
    }
```

```
    return i; // 返回和数的实际位数
```

```
}
```



# 输入

---

- ✓ 用字符一位一位输入，先输入高位再输入低位
- ✓ 放于数组中，从**0**号元素开始放，即高位放于低地址
- ✓ 实际存储：低位在低地址（也可以低位在高地址）
- ✓ 反转数组元素

# getBigNum

- **函数功能**：输入一个大整数，放于数组中，数组的每个元素存放一个十进制数，低位在低地址，即个位在x[0]，十位在x[1],...

- **函数参数**：

  - 存放大整数的数组x

  - 能输入的最多位数lim

- **函数返回值**：

  - 输入整数的实际位数

- **函数原型**

  - int getBigNum(int \*x,int lim);**

```
int getBigNum(int *x,int lim)
{
    int i,t,c, *p1,*p2;
    for(i=0;i<lim;i++)      /* 数组元素清零 */
        *(x+i)=0;
    /* 输入,最高位在x[0],.... */
    for(i=0,c=getchar();i<lim && isdigit(c);i++) {
        *(x+i)=c-'0';  // 字符 -> 整数
        c=getchar();
    }
    for(p1=x,p2=x+i-1;p1<p2;p1++,p2--) /* 反转 */
        t=*p1,*p1=*p2,*p2=t;
    return i;  // 返回读入的实际位数
}
```

# 输出

从高位按运算结果的实际位数输出每一位（数组元素）

**/\* 函数功能：输出一个大整数**

**函数参数：待输出的整数x**

**整数x的实际位数n**

**函数返回值： 无 \*/**

```
void putBigNum(int *x,int n)
```

```
{ int *p;
```

```
for(p=x+n-1;p>=x;p--) { /* 从高位开始输出 */
```

```
    putchar(*p+'0'); /* 将每一位转成对应数字字符输出，  
    或者 printf ("%d", *p); */
```

```
}
```

```
}
```



# 超大整数的加法运算步骤

- 1、输入被加数 $x$
  - 2、输入加数 $y$
  - 3、 $z=x+y$
  - 4、输出 $z$
- } 输入一个大数 (getBigNum)
- 两个大数相加 (addBigNum)
- 输出一个大数 (putBigNum)

//超大整数的加法运算程序

#define max(a,b) ((a)>(b)?(a):(b))

int main(void)

```
{    int x[N],y[N],z[N+1],lenx,leny,lenz,lenmax,i;
    printf("输入被加数: ");
    lenx=getbint(x,N);    /* 输入被加数x */
    printf("输入加数: ");
    leny=getbint(y,N);    /* 输入加数y */
    lenmax=max(lenx,leny);
    for(i=0;i<=N;i++)    /* 和数清零 */
        *(z+i)=0;
    lenz=addbint(x,y,z,lenmax); /* z=x+y */
    putbint(z,lenz); /* 输出z */
    putchar('\n');
    return 0;
}
```



## 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};**

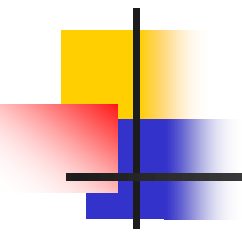
数据类型

指针数组名

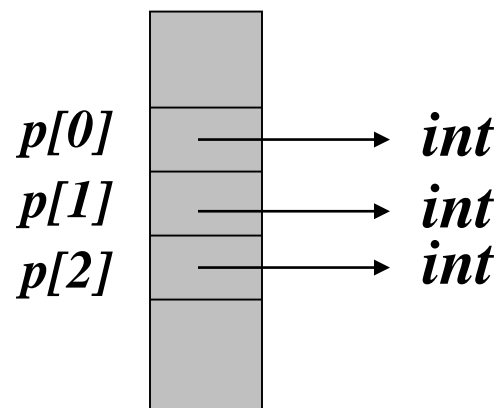
可选项：可以是**const**等

指针数组的大小

可选项，用于初始化指针数组。



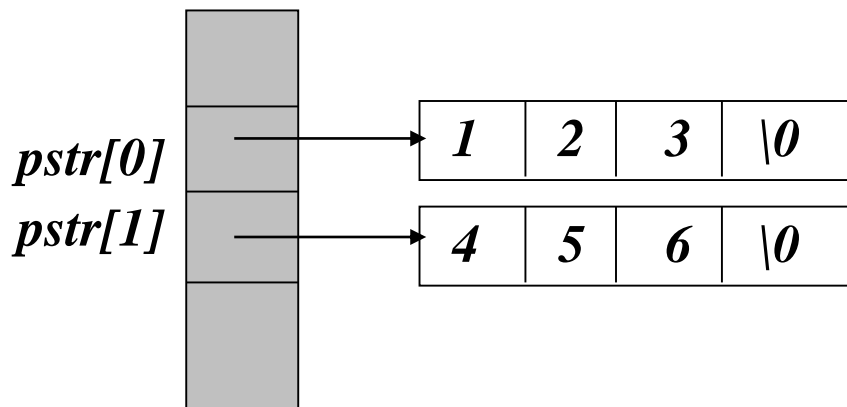
```
int *p[3]; /* p是一个有 3 个元素的整型指针数组  
           即每个元素是指向整型变量的指针 */
```





```
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 */
```

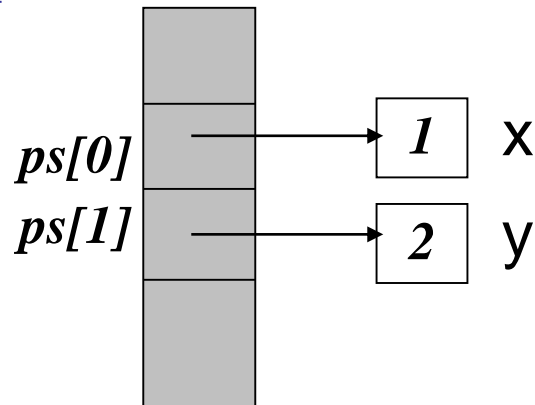
```
const int *ps[2];
```

/\* 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*/
```





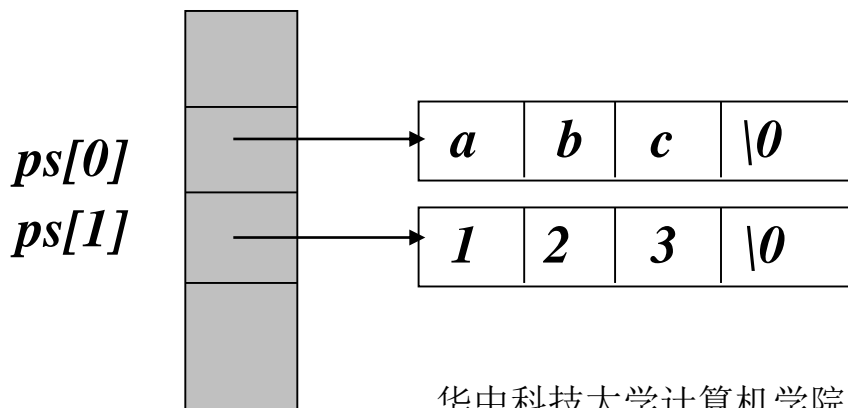
```
char * const ps[2]={ “abc”, “123”};
```

/\* **ps**是有2个元素的字符指针数组，但数组中的每个元素都是指针常量，通过初始化已经分别固定指向字符串“**abc**”和字符串“**123**”

即：指针常量数组。\*/

`ps[0] = “xyz”;` /\* 错误,指针数组ps的元素不能修改 \*/

`*ps[0] = ‘A’ ;` /\* 语法正确,指针元素所指的对象的值是可改  
但是，运行出错 \*/





---

```
const char * const ps[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 输出杨辉三角形

1				
1	1			
1	2	1		
1	3	3	1	
1	4	6	4	1

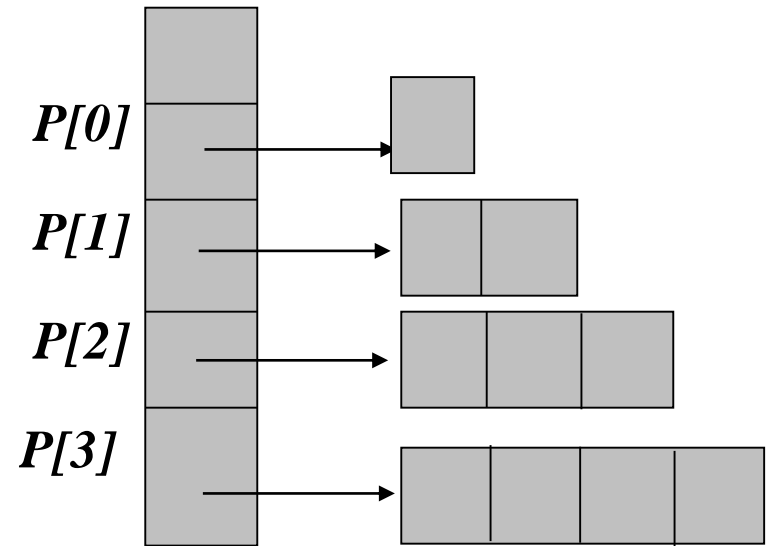
如何表示（存储）杨辉三角形？

(1) 二维数组

(2) 指针数组

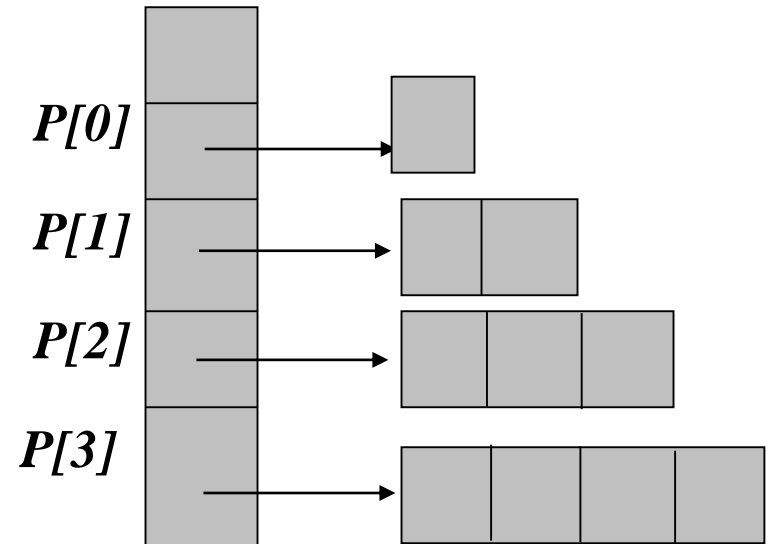
# 通过指针数组描述杨辉三角形

```
#define N 4
#define SIZE N*(N+1)/2
void main()
{
    int *p[N], a[SIZE], sum=0, i;
    for(i=0; i<N; i++) {
        p[i] = &a[sum];
        sum += i+1;
    }
    ....
}
```



# 通过指针数组描述杨辉三角形

```
#define N 4
#define SIZE N*(N+1)/2
void main()
{
    int *p[N], a[SIZE] , i;
    for(i=0;i<N;i++) {
        p[i]=&a[ i*(i+1)/2 ];
    }
    ....
}
```



```
#define N 4
```

```
#define SIZE N*(N+1)/2
```

```
void main()
```

```
{
```

```
int *p[N], a[SIZE], sum=0, i;
```

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

```
    p[i] = &a[sum]; sum += i+1;
```

```
}
```

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

```
    *p[i] = 1; *(p[i]+i) = 1;
```

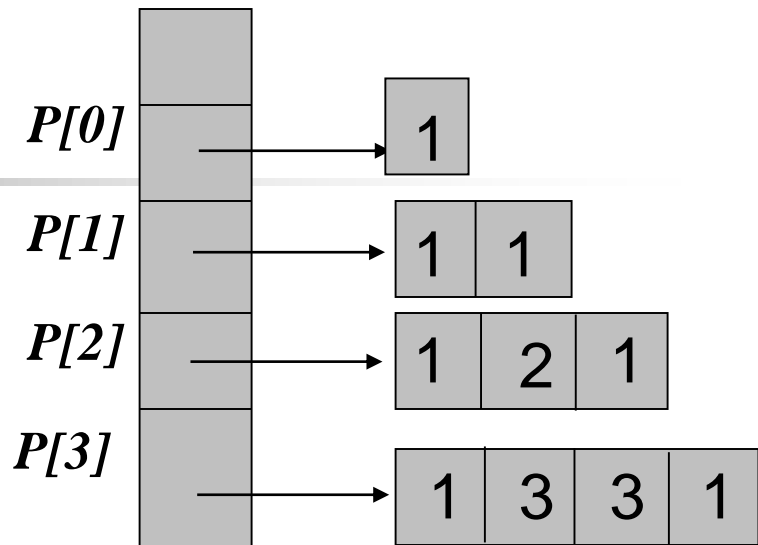
```
}
```

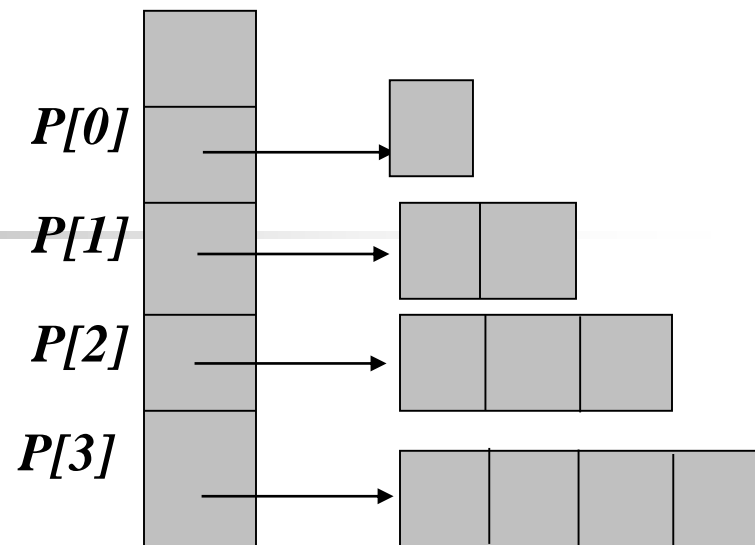
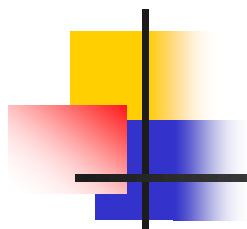
```
for(i=2; i<N; i++)
```

```
    for(k=1; k<i; k++)
```

```
        p[i][k] = p[i-1][k-1] + p[i-1][k];
```

```
}
```





但用指针方式存取数组元素比用下标速度快，  
并节省存储空间，

**int \*p[6], a[21] ;**

**6\*4+21\*4=108B**

**int x[6][6] ;**

**36\*4=144B**





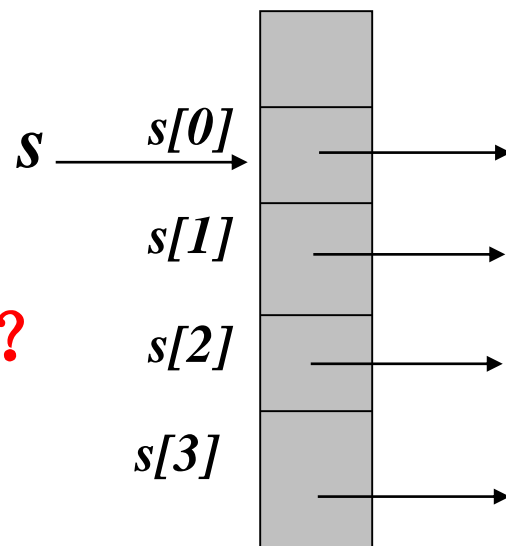
```
char *s[4];
```

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 */
```

```
#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` below to `argc`, indicating the number of arguments.
- An arrow points from the array `= { "sum", "10" }` below to `argv`, indicating the array of argument values.

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

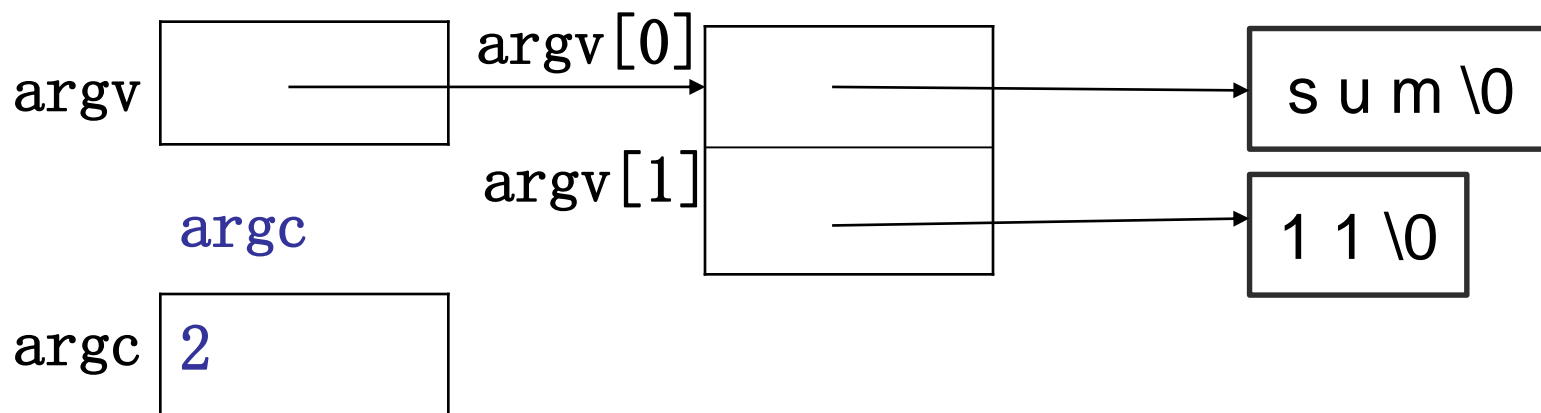
```
int main(int argc, char *argv[])
```

```
{
```

```
.....
```

```
}
```

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;
```

```
}
```

## 例 查找子串的指针函数。

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

```
char *strstr(const char *s, const char *ms);
```

```
int main(void)
```

```
{  char s1[]="abcdefghijk", s2[]="fgh", *p;
```

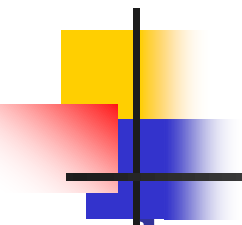
```
    p=strstr(s1, s2);
```

```
    printf("%p \t %s\n", &s1[0], s1 );
```

```
    printf("%p \t %s\n", p , p );
```

```
    return 0;
```

```
}
```



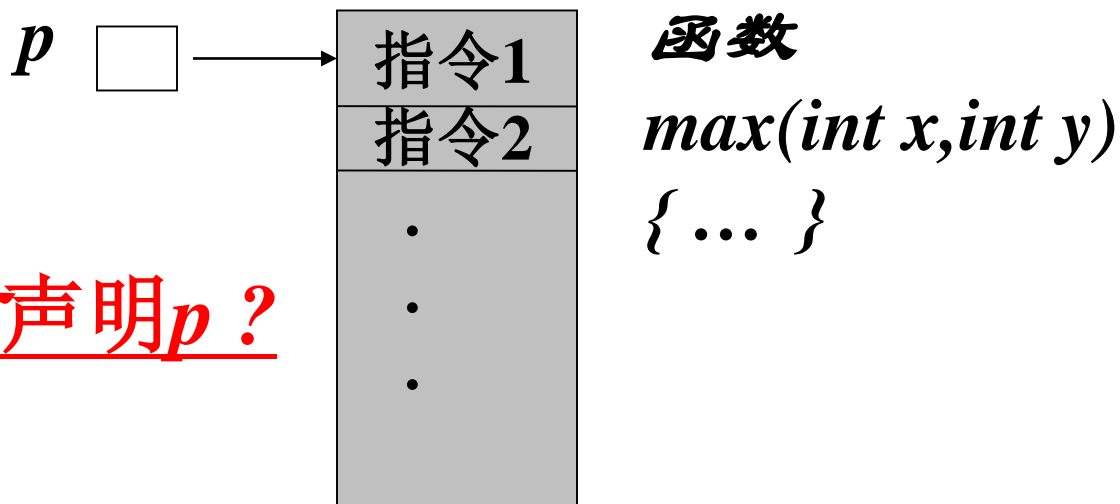
```
char *strstr ( const char *s,const char *ms)
{ char *ps=s,*pt,*pc;
  while(*ps!='\0'){
    for(pt=ms,pc=ps;*pt!='\0'&&*pt==*pc; pt++, pc++)
      ;
    if(*pt=='\0') return ps;
    ps++;
  }
  return NULL; /* NULL : 0 */
}
```



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

## 8.8.2 函数指针的应用

### 例[8.30]

读取从键盘输入的正文，再将正文以行为单位排序后输出。通过命令行参数-n决定排序方法。如果有命令行参数-n，则将输入行按照数值大小进行排序；否则将按照字典顺序排序。

输入：

C:\>ex-30

a book

this is a pen.

that is a car.

C language

^Z

则输出为：

C language

a book

that is a car.

this is a pen.

输入：

C:\>ex-30 -n

34

112

21

56

^Z

则输出为：

21

34

56

112



# 定义一个通用的排序函数

- 既能实现将输入行按照数值大小进行排序；也能实现按照字典顺序排序。
- 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--**指向输入行的指针数组

**n--**行数

**comp--**指向函数的指针，用于确定排序的规则

函数返回值：无

*/\*对指针数组v指向的n行对象按comp规则排序 \*/*

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

```
{
```

```
    int i, j ;
```

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

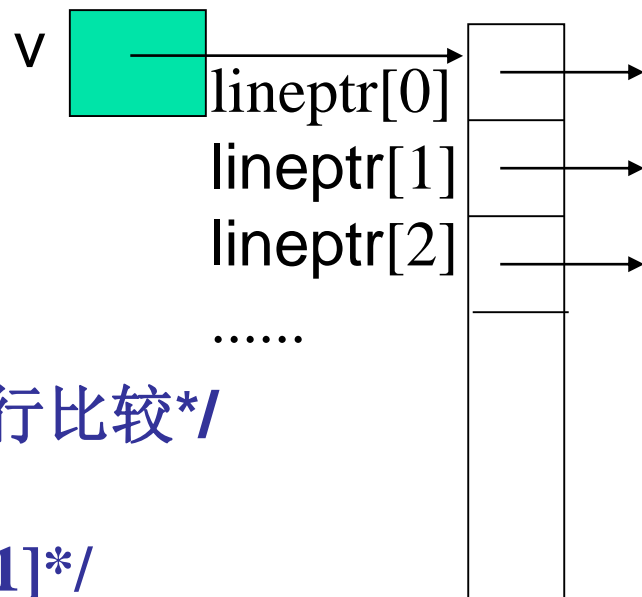
```
        for(j=0; j<n-i; j++)
```

```
            /*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：按字典序降序

---

```
int strcmp_des(char *s1, char *s2 )  
{  
    if(strcmp(s1,s2)<0)  
        return 1;  
    else return 0;  
}
```



# 规则：按数值大小升序

---

```
int numcmp_asc(char *s1, char *s2 )
{
    double v1,v2;

    v1=atof((const char *)s1);
    v2=atof((const char *)s2);
    if(v1>v2) return 1;
    else return 0;
}
```

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

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

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

```
#define MAXLINES 500 /* 正文的最大行数 */
```

```
char * lineptr[MAXLINES]; /* 指针数组, 描述正文 */
```

```
int readlines(char *[ ],int); /* 读入正文的各行 */
```

```
void writelines(char *[ ],int); /* 输出正文的各行 */
```

```
/* 对正文按行排序 */
```

```
void sort(char *[ ], int, int(*)(char *,char *));
```

```
int numcmp(char *,char *); /* 按数值大小比较两个串 */
```

```
void swap ( char *[ ], int , int) ; /* 交换 */
```

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

```
{  
    int nlines; /* 正文的实际行数 */
```

```
    int numeric=0; /* 默认命令行无-n(即正文行由任意字符串组成)  
        有-n 则numeric=1 (即正文行由数字串组成) */
```

```
    if(argc>1&&strcmp(argv[1],"-n") ==0 )    numeric=1;
```

```
    if(numeric==1)    printf("input lines, every one is a numeric:\n");
```

```
    else    printf("input lines, every one is a string:\n");
```

```
    if ( (nlines=readlines(lineptr,MAXLINES)) >0 ) {
```

```
        sort( lineptr, nlines, (numeric?numcmp:strcmp));
```

```
        writelines(lineptr,nlines);
```

```
    }
```

```
    else    printf("input too big to sort\n");
```

```
    return 0;
```

```
}
```



```
#define MAXLEN 100 /* 正文行的最大长度 */
```

```
int getline(char *, int ); /* 读入一行 */
```

```
/* readlines: 最多读入 maxlines 行正文, 指针数组 lineptr 指向  
该正文, 返回实际读入的行数 */
```

```
int readlines(char *lineptr[ ], int maxlines)
```

```
{
```

```
    int len, nlines;
```

```
    char *p, line[MAXLEN];
```

```
    nlines=0;
```

```
    while( (len=getline ( line, MAXLEN ) ) >0 )
```

```
        if ( nlines >= maxlines || (p=(char *)malloc(len+1)) == NULL )
```

```
            return(-1); /* 出错返回 */
```

```
        else {
```

```
            strcpy(p,line);
```

```
            lineptr[nlines++]=p;
```

```
        }
```

```
    return(nlines);
```

```
}
```



*/\* getline: 读入字符数最多为(lim-1)的一行到 s,  
返回该行实际字符数 \*/*

```
int getline(char s[ ], int lim)
{
    int c, i;

    i=0;
    while( --lim>0 && (c=getchar( ))!=EOF && c!='\n')
        s[i++]=c;
    s[i]='\0';    /* 添加串尾 */
    return(i);
}
```



---

*/\* 输出指针数组 *lineptr* 指向的 *nlines* 行正文 \*/*

**void writelines(char \*lineptr[ ], int nlines)**

**{**

**while( nlines - - > 0 )**

**printf(“%s\n”, \*lineptr++);**

**}**

*/\* swap: 将指针数组 v 指向的正文第 i 行和第 j 行交换 \*/*

**void swap ( void \*v[ ], int i, int j )**

**{**

**void \*temp ;**

**temp=v[i];**

**v[i]=v[j];**

**v[j]=temp;**

**}**

*/\* numcmp: 按数值大小比较串 s1 和 s2 \*/*

**int numcmp(char \*s1, char \*s2 )**

**{**

**double v1,v2;**

**v1=atof(s1);**

**v2=atof(s2);**

**return(v1<v2?-1:v1>v2 ? 1:0 );**

**}**



# 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*/
```

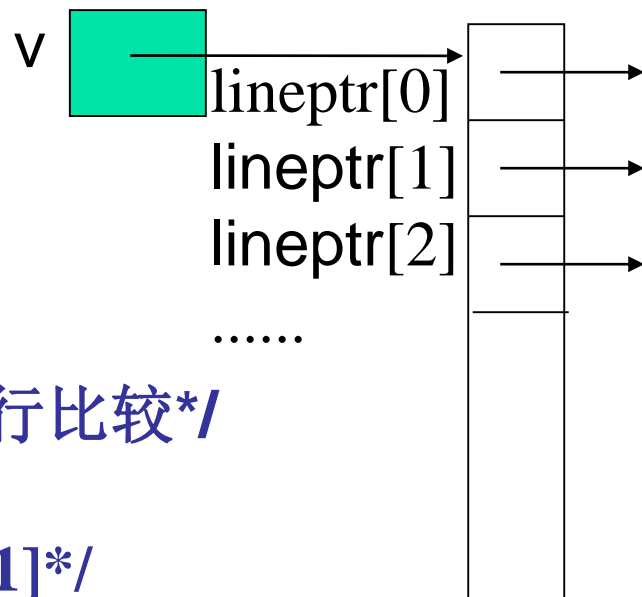
```
        for(j=0; j<n-i; j++)
```

```
            /*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。

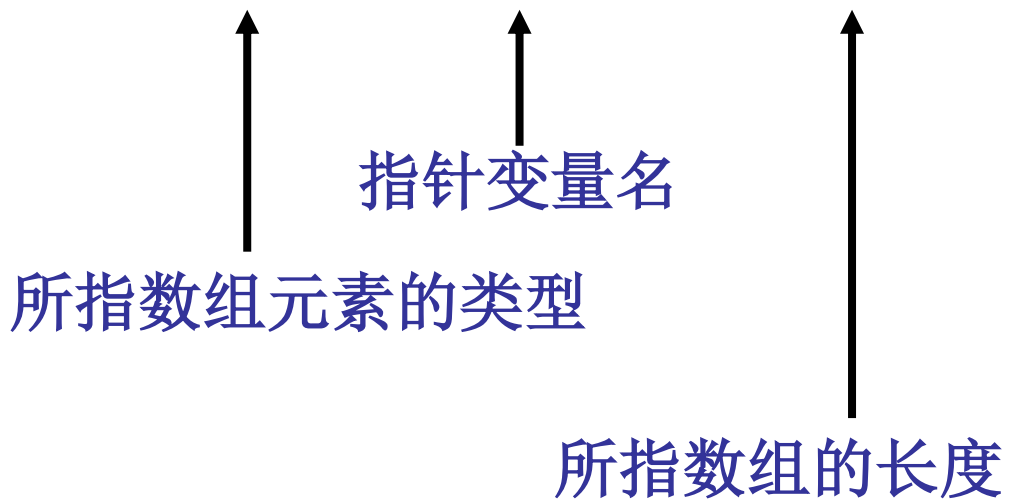


# 11.1 指向数组的指针

## 11.1.1 指向数组的指针的声明与定义

指向数组的指针又称为数组的指针。

类型名 (\*标识符) [常量];



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

```
int *p;
```

将指针**p**指向数组的首元素

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

✓

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

✓

```
p=a;
```

✗

```
p=(int *)a;
```

✓

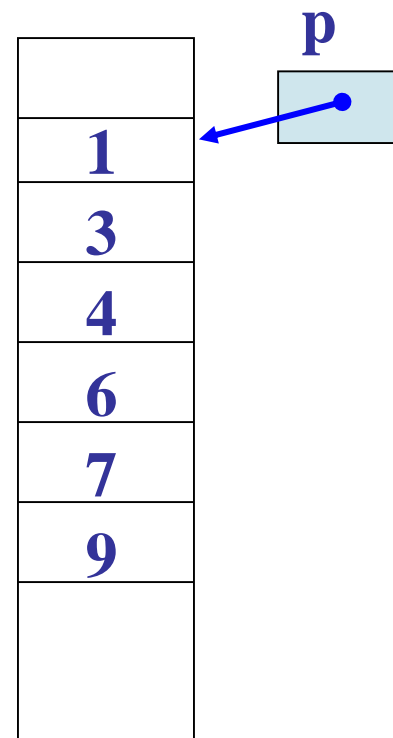
```
p= *a;
```

✓

指针常量**a**是什么类型？

—— 指向数组的指针

```
int (*)(2)
```



## 11.1.2 用数组名间访二维数组的元素

二维数组被看成以1维数组（行）为元素的一维数组；

`int u[2][3]={1, 3, 5  
              {2, 4, 6}};`

$u \rightarrow$

$u[0]$	1	3	5
$u[1]$	2	4	6

$u$ 被看成有两个1维数组（行）元素( $u[0], u[1]$ )组成的一维数组

$u[0]$  第0行首地址 即  $u[0] == \&u[0][0]$

$u[1]$  第1行首地址 即  $u[1] == \&u[1][0]$

$i$ 行 $j$ 列的元素的地址？

(1) 用指向数组元素的指针表示:  $u[i]+j$

(2) 用指向一维数组的指针表示:  $*(u+i)+j$

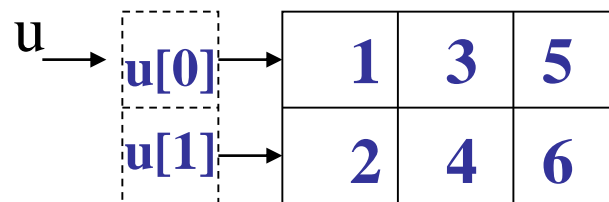
如何间访 $i$ 行 $j$ 列的元素？

(1)  $*(u[i]+j)$

(2)  $*(*(u+i)+j)$

## 二维数组元素的表示方法

```
int u[2][3]={ {1, 3, 5}  
              {2, 4, 6} };
```



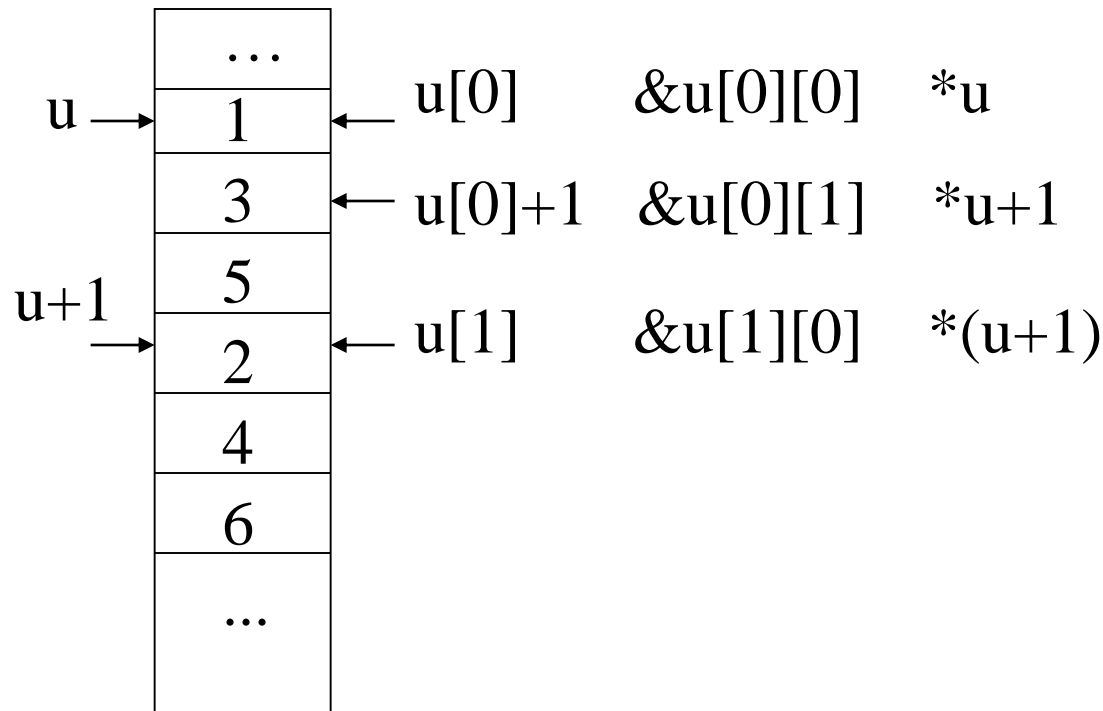
1、下标法       $u[i][j]$

2、指针法       $*(u[i]+j)$        $u[i]: \text{int} *$

$*(*(u+i)+j)$        $u: \text{int} (*)[3]$

# 用指针变量描述二维数组元素

```
int u[2][3]={  
    {1, 3, 5}  
    {2, 4, 6}  
};
```



## 1、将指针定义为指向数组元素

见8.4.3

## 2、将指针定义为指向由m个元素组成的数组的指针

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

```
int (*p)[2];
```

*p是指向数组的指针，该数组有2个int 型元素*

```
p=a;
```

<code>*(p)</code> 或 <code>(p)[0]</code>	<code>*(p+1)</code> 或 <code>(p)[1]</code>
<code>*(p+1)</code> 或 <code>(p+1)[0]</code>	<code>*(p+1+1)</code> 或 <code>(p+1)[1]</code>
<code>*(p+2)</code> 或 <code>(p+2)[0]</code>	<code>*(p+2+1)</code> 或 <code>(p+2)[1]</code>

## 例 二维数组元素的输入/输出

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

```
#define I 2
```

```
#define J 3
```

```
void main(void)
```

```
{   int u[I][J], (*p)[J]=u;
```

```
    int j;
```

```
    for(j=0;j<J;j++)    /* 用指向数组元素的指针完成第0行元素的输入 */
```

```
        scanf("%d", (u[0]+j));
```

```
    for(p++,j=0;j<J;j++) /* 用指向数组的指针完成第1行元素的输入 */
```

```
        scanf("%d", (*p+j));
```

```
    for(j=0;j<J;j++) /* 用指向数组的指针完成第0行元素的输出 */
```

```
        printf("%6d", *(&u[0][j]));
```

```
    printf("\n");
```

```
    for(j=0;j<J;j++)    /* 用指向数组元素的指针完成第1行元素的输出 */
```

```
        printf("%6d", *(&u[1][j]));
```

```
    printf("\n");
```

```
}
```

## 11.1.3 用指向数组的指针表示三维数组

类推:

三维数组被看成以二维数组（页）为元素的一维数组

一个n维数组被看成以n-1维数组为元素的一维数组。

```
int x[2][3][4];
```

```
int (*p2)[3][4]=x;
```

/\*p2是指向**3行4列的二维整型数组**的指针\*/

```
*(*(*(p2+i)+j)+k) <==> p2[i][j][k]
```



## 11.1.4 二维数组作函数参数

- 形参说明为数组

```
fun(int x[ ][4])  
{  
    ...  
}
```

- 形参说明为指针

{ 指向数组元素的指针  
 指向下一级数组的指针

```
fun(int *x)  
{  
    ...  
}
```

```
fun(int (*x)[4] )  
{  
    ...  
}
```

## 【例11.5】 日期转换问题。



---

输入一个年、月、日表示的日期，将其转换为该年的第几天。如果输入某年的第几天，将其转换为该年某月某日并输出。



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

```
#define N 13
```

```
int day_of_year(int year,int month,int day,int (*pdaytab)[N]);
```

```
void month_day(int year,int yearday,int *p,int *pmonth,  
               int *pday);
```

```
void main(void)
```

```
{
```

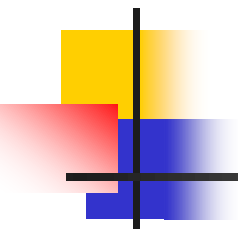
```
    int day_tab[2][N]={
```

```
        {0,31,28,31,30,31,30,31,31,30,31,30,31}, /* 非闰年 */
```

```
        {0,31,29,31,30,31,30,31,31,30,31,30,31} /* 闰年 */
```

```
    };
```

```
    int yy,mm,dd,yday;
```



```
printf("input the year,month, and day please!\n");
scanf("%d%d%d",&yy,&mm,&dd);
yday = day_of_year(yy,mm,dd,day_tab);
printf("day of the year is %d\n",yday);
printf("input the year and the days of the year please!\n");
scanf("%d%d",&yy,&yday);
month_day(yy,yday,day_tab[0],&mm,&dd);
printf("It's %d month and %d day in %d.\n",mm,dd,yy);
}
```

**/\* day\_of\_year函数：根据年year、月month、日day表示的日期转换为该年的第几天返回，pdaytab：指向一维数组的指针 \*/**

**int day\_of\_year(int year,int month,int day,int (\*pdaytab)[N])**

**{**

**int i,leap;**

**/\* leap=1(闰年)，0 (非闰年) \*/**

**leap=year%4==0 && year%100 !=0 || year%400==0;**

**for(i=1;i<month;i++)**

**day+=\*(\*(pdaytab+leap)+i);**

**return day;**

**}**

形参为指向数组元素的指针，  
实参应为数组名或指向下一级  
数组的指针变量

**pdaytab[leap][i]：该年第 i 月的天数**

**/\* month\_day函数：根据某年year的第几天yearday，将其转换为该年的某月某日，形参p：指向数组元素的指针 \*/**  
**/\* pmonth：指向转换的某月， pday：指向转换的某日 \*/**

```
void month_day(int year,int yearday,int *p,int *pmonth,  
int *pday)
```

```
{
```

```
int i,leap;
```

```
leap=year%4==0 &&
```

```
for(i=1;yearday>*(p+leap*N+i);i++)
```

```
yearday-=*(p+leap*N+i);
```

```
*pmonth=i;
```

```
*pday=yearday;
```

```
}
```

形参为指向下一级数组的指针，  
实参应为数组元素的地址，  
或为指向数组元素的指针变量。

p[leap\*N+i]：该年第 i 月的天数

# 11.2 用typedef定义类型表达式

## 11.2.1 类型表达式

C中的表达式可以分成两类。

- (1) **值表达式**，由运算符和操作数组成，可被CPU处理和计算
- (2) **类型表达式**，由类型说明符和数据类型名组成，类型说明符有：()、[]、\*

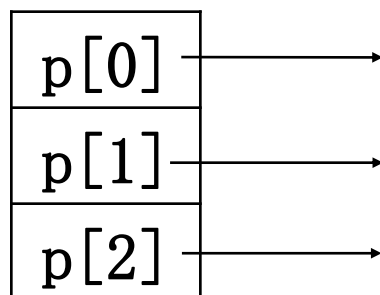
**int (\*) [5]**

## 11.2.2 用typedef定义类型表达式

类型表达式可以出现在两个地方：

### 1) 声明语句中

```
int *p[3];
```





## 2) typedef定义中

**typedef**是关键字，为一个类型表达式定义一个别名。

**typedef** 类型区分符 说明符 ;



基本类型 结构 联合

也可以是由 **typedef** 定义的类型名

(1) **typedef unsigned int size\_t;**

**size\_t**定义为**unsigned int**类型

**size\_t x, y; /\* unsigned int x, y; \*/**

(2) **typedef char \*string ;**

*/\* 将 string 定义为 char 型指针 \*/*

**string p, s[10];**  **char \*p, \*s[10];**

*/\* p 是字符指针,*

*s 是含有10个元素的字符指针数组 \*/*

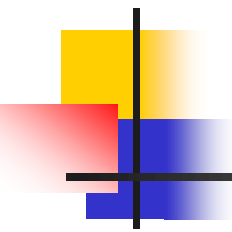
而 **#define string char \***

*/\* string 是宏名 , 简单的串替换 \*/*

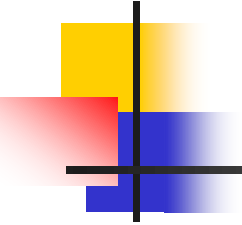
**string p, s[10];**  **char \*p, s[10];**

*/\* p 是字符指针,*

*s 是含有10个元素的字符数组 \*/*



```
(3)  typedef struct {  
        char num[12];    /* 学号成员，字符数组类型 */  
        char name[9];    /* 姓名成员，字符数组类型 */  
    } student;  
  
    student a,b;
```



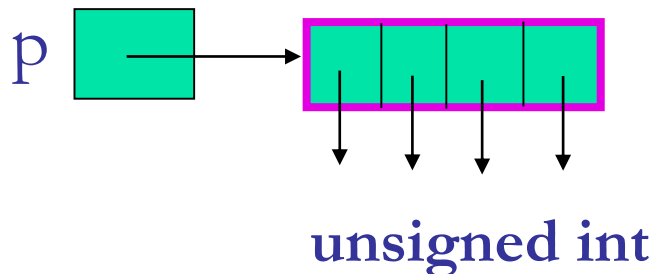
---

(4) `typedef char * (*p_to_fun)(char *,char *);`  
p\_to\_fun 定义为 `char *(*)(char *,char *)`

`p_to_fun` fptr;

## 11.3 复杂说明的解释

*unsigned \*(\*p)[4]*



*p* 是指向数组的指针，  
该数组有4个无符号整型指针元素。  
或

*p* 是有4个无符号整型指针元素的数组的指针。



# 指针：保存变量地址的变量

---

## ■ \* [ ] ( )

1. `int p;`
2. `int p[10];`
3. `Int p( );`
4. `int *p;`
5. `int *p[ 10];`
6. `int *p( );`
7. `int (*p)( );`



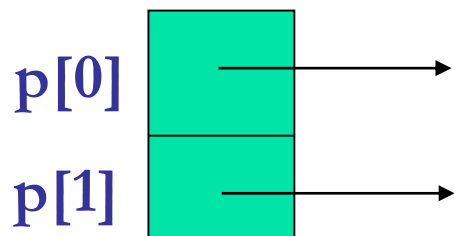
# 复杂指针的声明

- `int (*( *p( )) [10]) ( )`
- **P:** 函数，函数的返回类型是一个指针，该指针指向一个有**10**个元素的数组，数组中的每一个元素都是一个指向函数的指针，函数的返回类型为整型
- `int (*( *p [10]) ( )) [10]`
- **P:** 数组，数组中的元素为指针，指针指向一个返回类型为指针的函数，该指针指向一个有**10**个元素的整型数组



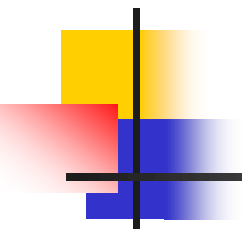
```
char *(*p[2])(char *, int);
```

*p*是含有2个指针元素的数组，每个指针指向有一个字符指针参数和一个整型参数，返回值为字符型指针的函数。



函数：*char* \* (***char*** \*, ***int***)

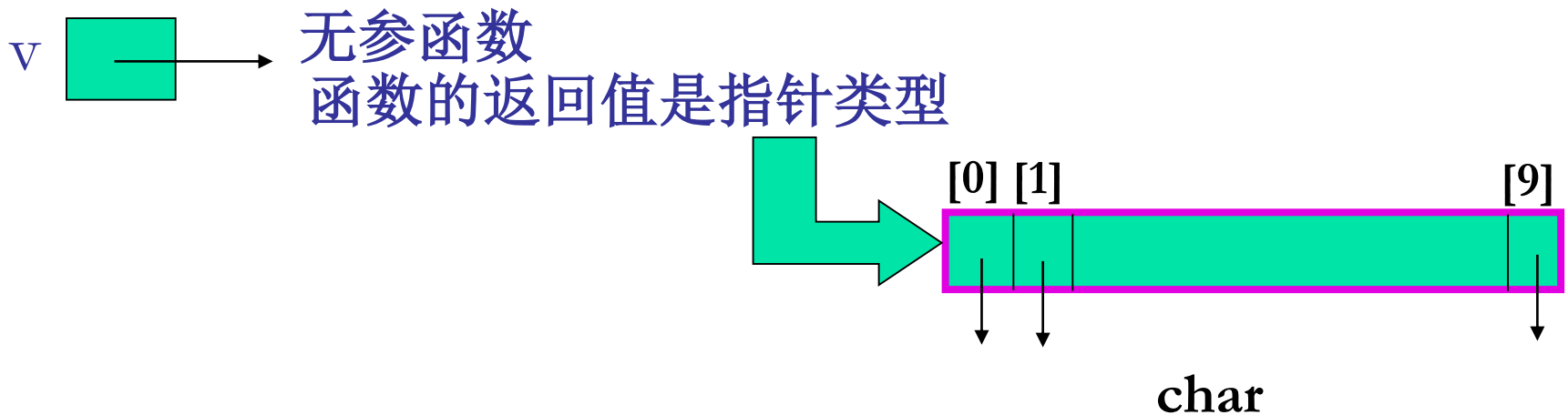




```
char * (*(*v)(void))[10];
```

(6) (5) (3) (1) (2) (4)

v 是函数的指针，该函数没有参数，  
返回值是指向有10个元素的字符指针数组的指针。





```
int (*f(char *(*)(int)))[10];
```

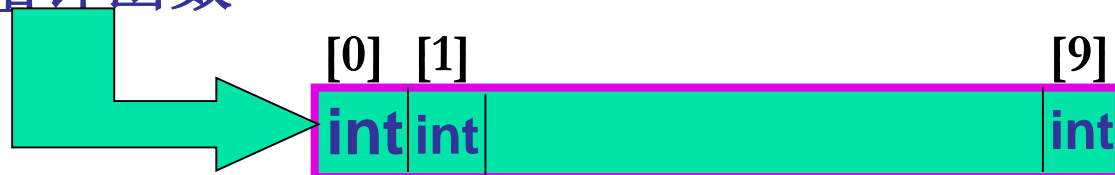
*f*是一个指针函数，

*f*函数的形参为一个指向函数的指针，所指函数有一个整型形参且返回值类型为char \*；

*f*函数的返回值是指向有10个整型元素的数组的指针。

*f* 的形参： char \*(\*)(int)

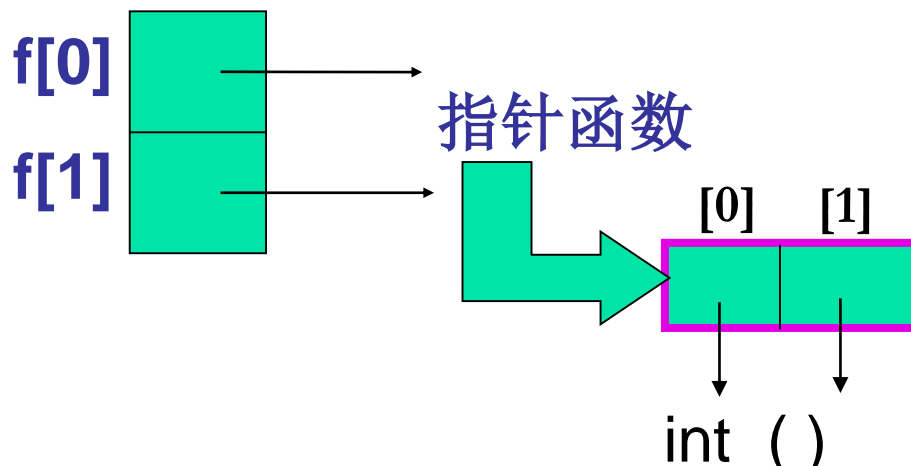
*f*是一个指针函数





```
int ( *(*(*f[2])())[2])();
```

f是一个有两个元素的函数指针数组；所指函数的返回值是指向有两个元素的指针数组的指针，指针数组的每个指针元素指向一个无参整型函数。（无参整型函数指函数没有参数且返回值为整型值）。





- 数组指针

`int (*p)[10]`

- 指针数组

`int *p[10]`

- 指针函数

`int *p( )`

- 函数指针

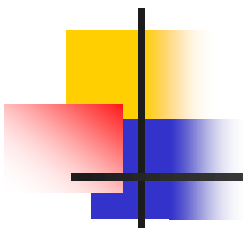
`int (*p)( )`

- 函数指针数组

`int (*p[10])( )`

- 函数指针数组的指针

`int (*( *p)[10])( )`



	声明	说明
数组指针	<code>int (*p)[10]</code>	P: 指针，指向一个大小为10的整型数组。
指针数组	<code>int *p[10]</code>	P: 数组，数组的每个元素为指向整型数据的指针
指针函数	<code>int *p( )</code>	P: 函数，函数的返回值是指向整型的指针
函数指针	<code>int (*p)( )</code>	P: 指针，指针指向一个函数，指向的这个函数没有参数，返回类型为int型
函数指针数组	<code>int (*p[10])( )</code>	P: 数组，数组的元素为指针，指针指向返回类型为int的函数
函数指针数组的指针	<code>int (*( *p)[10])( )</code>	P: 指针，指向具有10个元素的数组，数组的每个元素都是指针，指向返回类型为int的函数