第10章 指针

- 地址和指针的概念
- 变量的指针和指向变量的指针变量
- 数组与指针
- 字符串与指针
- 指向函数的指针
- 返回指针值的函数
- 指针数组和指向指针的指针
- 有关指针的数组类型和指针运算的小结

★本章学习目标：

- ❖认识到用地址作为一种数据类型的重要性。
- ❖理解指针包括地址和类型两种属性。
- ❖掌握指针运算符&和*。
- ❖能够通过地址引用调用在被调函数与主调函数之间共享数据。
- ❖理解指针和数组的关系。
- ❖理解指向函数的指针的用法。

★C程序设计中使用指针可以:

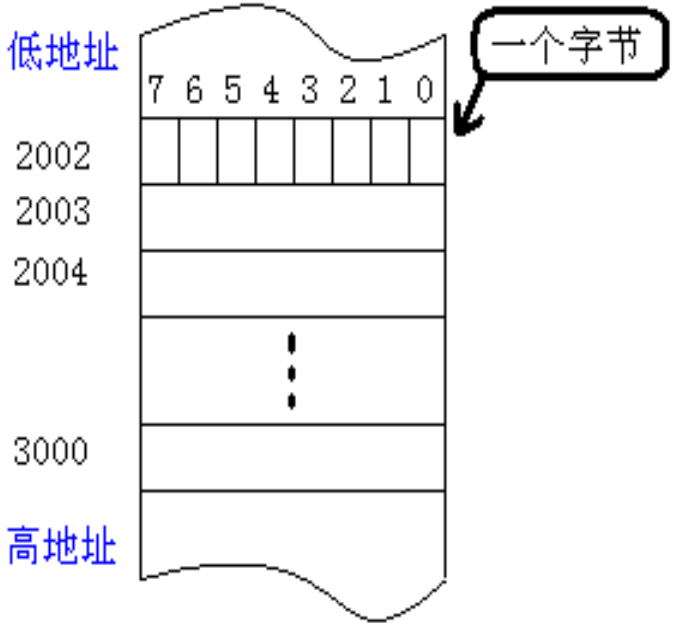
- ❖使程序简洁、紧凑、高效
- ❖有效地表示复杂的数据结构
- ❖动态分配内存，直接访问内存地址
- ❖得到多于一个的函数返回值

§ 10.1 地址和指针的概念

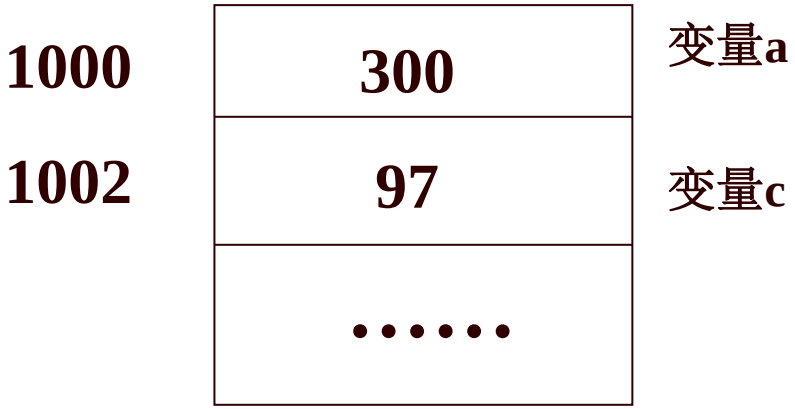
指针是**变量的地址**，变量的地址就是**内存地址**。

1、内存地址

在计算机中，把内存区划分为一个一个的存储单元，每个单元为一个字节（8位），每一个字节都有一个唯一的编号，这个编号就是唯一的内存地址。



§10.1 地址和指针的概念



注意：1、程序中定义的每个数据在编译后都占有各自的内存区。

2、数据所占有的存储单元个数是由其类型决定的。如：整型2字节，实型4字节

3、首地址：第1个单元的地址，

4、内存单元的地址与单元的数据区别：

5、系统通过变量名对内存单元访问

2、变量地址

定义一个变量，系统会根据变量的类型分配内存单元。

内存单元的首地址，就是变量的地址。

变量的值：即变量所在内存单元的内容

3、指针和指针变量

（1）指针就是地址

–变量在内存单元的首地址称为该变量的“指针”。

•理解 -通过变量的指针可以找到变量的内存单元。

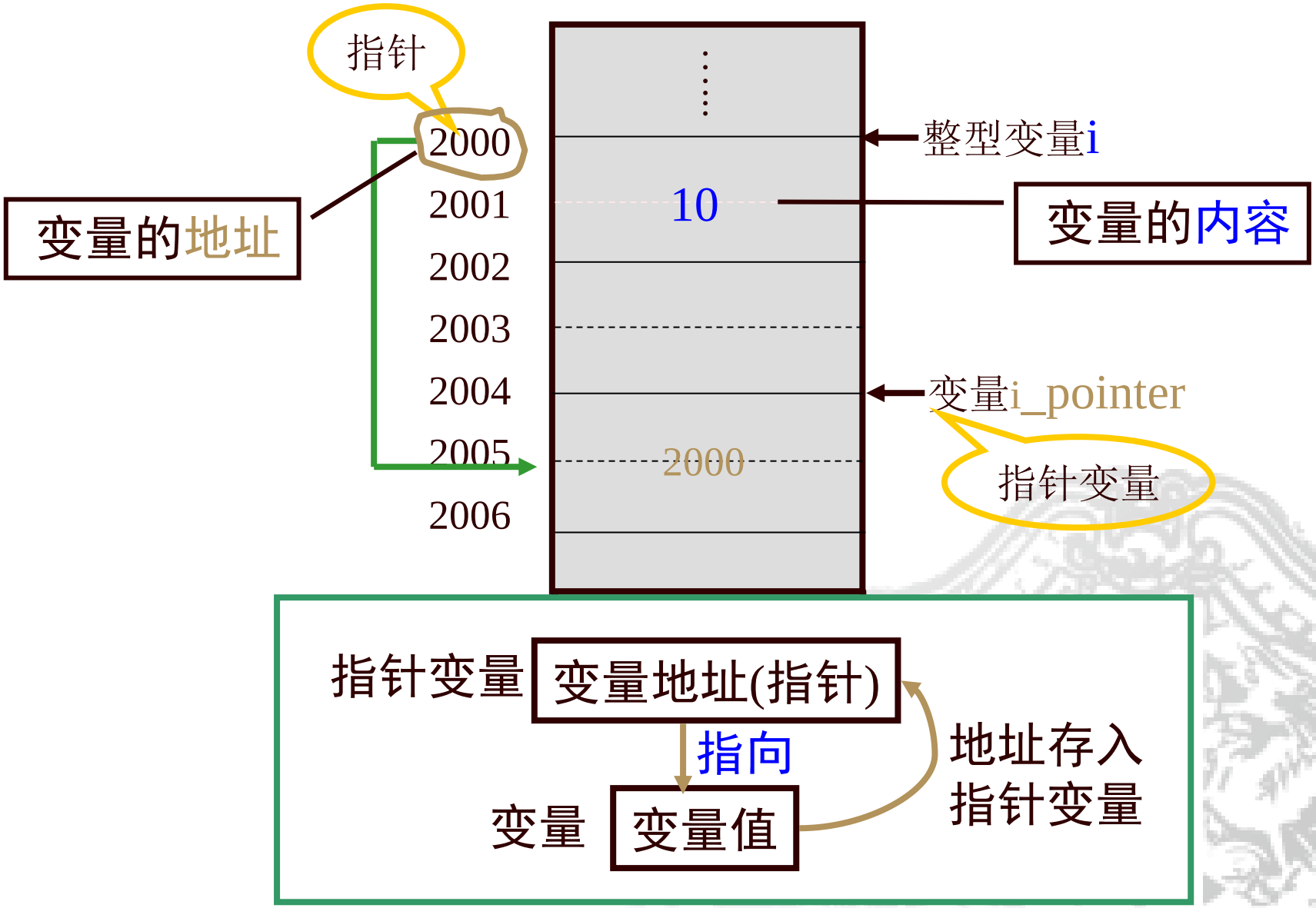
–或者说:指针指向该变量的内存单元.

（2）指针变量：用来存放其它变量的地址或指针的变量。

例如：int i=32;　　指针变量i_pointer=&i;

指针变量的特点: (1)指针变量是一种变量，　在内存中要占有一定数量的存储单元。

(2)指针变量用来存放其他变量的指针值（即地址）。


指针
2000
2001
2002
2003
2004
2005
2006
变量的地址
10
2000
整型变量i
变量的内容
变量i_pointer
指针变量
指针变量
变量地址(指针)
指向
变量
变量值
地址存入
指针变量

问题

判断正错：

§1、指针和指针变量的含义相同

§2、指针变量存放的是地址

§3、变量的值和变量的指针含义相同

说明：

p1存放a变量的地址称p1指向a.

10.2 变量的指针和指向变量的指针变量

指针变量与其所指向的变量之间的关系

在分析有关指针的程序时，画图是很好的方法：

10.2 变量的指针和指向变量的指针变量

★指针变量的定义(先定义，后使用)

指针变量在使用前必须定义，使其指向特定类型的变量，指针变量存放地址，必须定义为“指针类型”。

❖定义的一般形式：基类型 *指针变量名;

- 基类型：用来指定指针变量可以指向的变量的类型。
  将决定指针移动和运算时的移动量。

构成：[存储类型] 数据类型

- *：表示该变量为指针类型

例 int *p1,*p2;
float *q ;
static char *name;

注意：

1、int *p1, *p2; 与 int *p1, p2;

2、指针变量名是p1,p2 ,不是*p1,*p2

3、指针变量只能指向定义时所规定类型的变量

4、指针变量定义后，值不确定，应用前必须先赋值

5、指针变量的值只能是某个数据的地址，只允许取正整数，区别于整型变量。

★进一步理解&与*运算符：

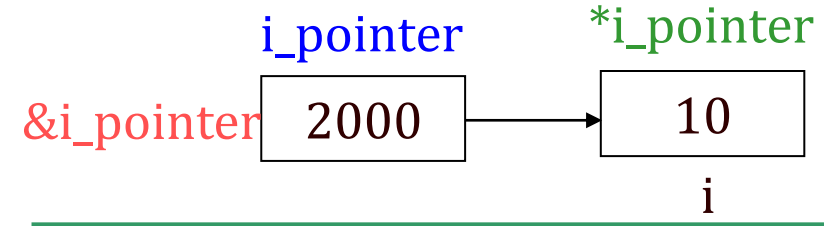
❖含义

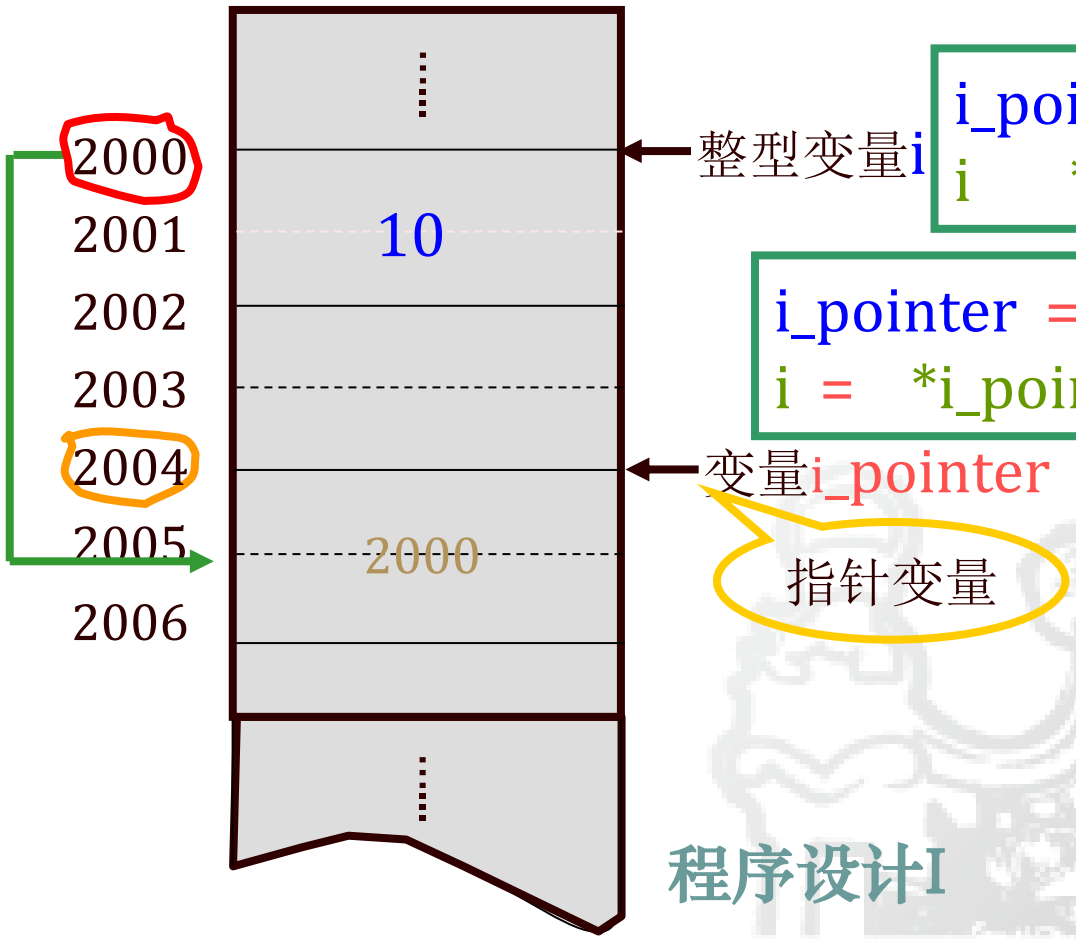
取地址运算符
含义：取变量的地址
单目运算符
优先级：2
结合性：自右向左

指针运算符（“间接访问”运算符）
含义：取指针所指向变量的内容
单目运算符
优先级：2
结合性：自右向左

★进一步理解&与*运算符：

- ❖含义
- ❖两者关系：互为逆运算
- ❖理解





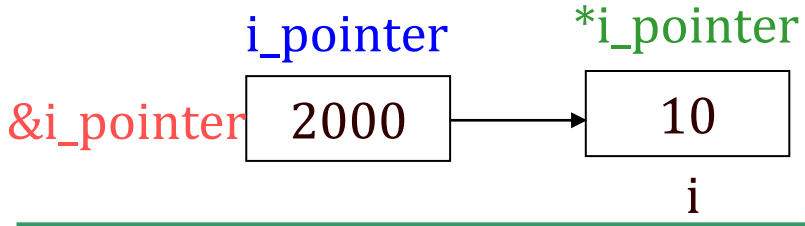
i_pointer &i &(*i_pointer)
i *i_pointer *(&i)

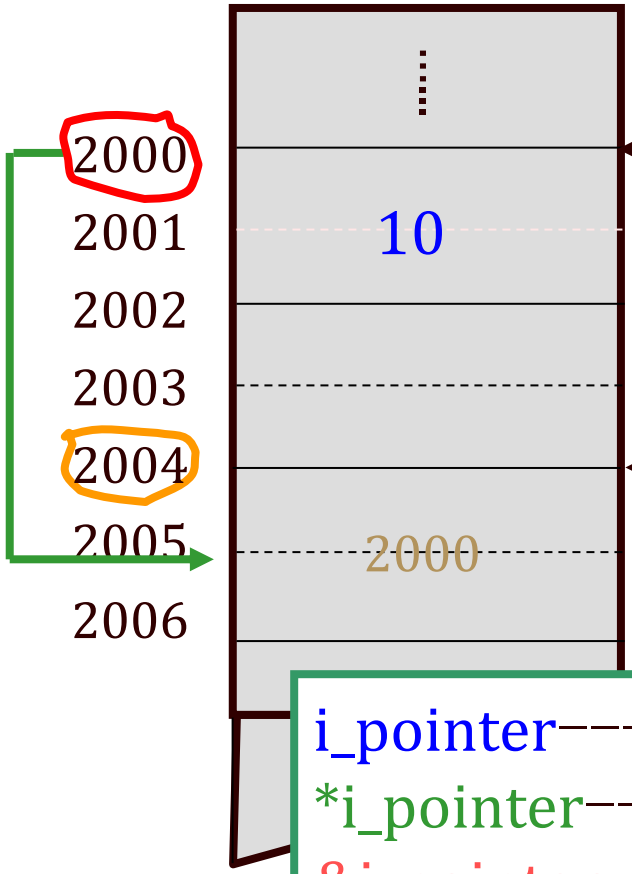
i_pointer = &i = &(*i_pointer)
i = *i_pointer = *(&i)

i ⟺ *i_pointer
&i ⟺ i_pointer
i=3 ⟺ *i_pointer=3

★进一步理解&与*运算符：

- ❖含义
- ❖两者关系：互为逆运算
- ❖理解





i_pointer &i &(*i_pointer)
i *i_pointer *(&i)

i_pointer = &i = &(*i_pointer)
i = *i_pointer = *(&i)

i ⟺ *i_pointer
&i ⟺ i_pointer
i=3 ⟺ *i_pointer=3

i_pointer------指针变量，它的内容是地址量
*i_pointer-----指针的目标变量，它的内容是数据
&i_pointer---指针变量占用内存的地址

注意事项：

&：取地址，注意与作位运算符时的不同（双目）

***：取内容 ，注意与作乘运算符时的不同（双目）**

& 既可作用于一般变量，也可作用于指针变量

*** 只能作用于指针变量**

定义指针变量时的 * 与该处的含义不同

★运算符说明

int a, b, *p;

p=&a; b=*p;

1) &和*互为逆运算

2) &和*优先级相同，结合性为右结合。

3) &*p等价于&(*p),其结果就是p

4) *&x等价于*(&x),其结果就是x.

关于 & 和 * 运算符的进一步说明：

*、&：优先级同为2级，　结合性：从右向左。

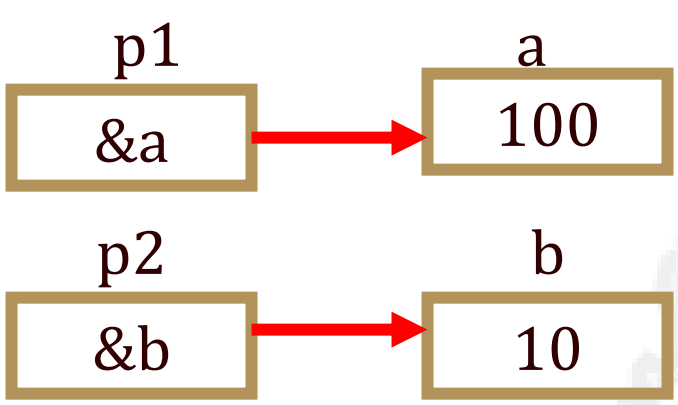
1.若已执行： int a, b, * p1, * p2;

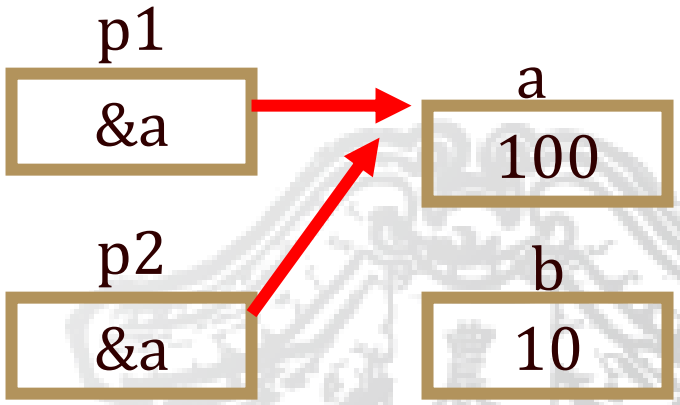
p1=&a; p2=&b; a=100; b=10;

则 ① &* p1 ↔ &a (p1)

&* p2 ↔ &b (p2)

②p2=&* p1 ↔ p2=&a





关于 & 和 * 运算符的进一步说明：

*、&：优先级同为2级，　结合性：从右向左。

1.若已执行： int a, b, * p1, * p2;

p1=&a; p2=&b; a=100; b=10;

则 ① &* p1 ↔ &a (p1)

&* p2 ↔ &b (p2)

②p2=&* p1 ↔ p2=&a

2. * & a：先进行&a得a的地址，再对a的地址进行* 运算

即指向a地址所指向的变量，就是a，其值是100

3. 运算符 *，++：优先级为2，结合性：从右到左

(* p1)++ → a++

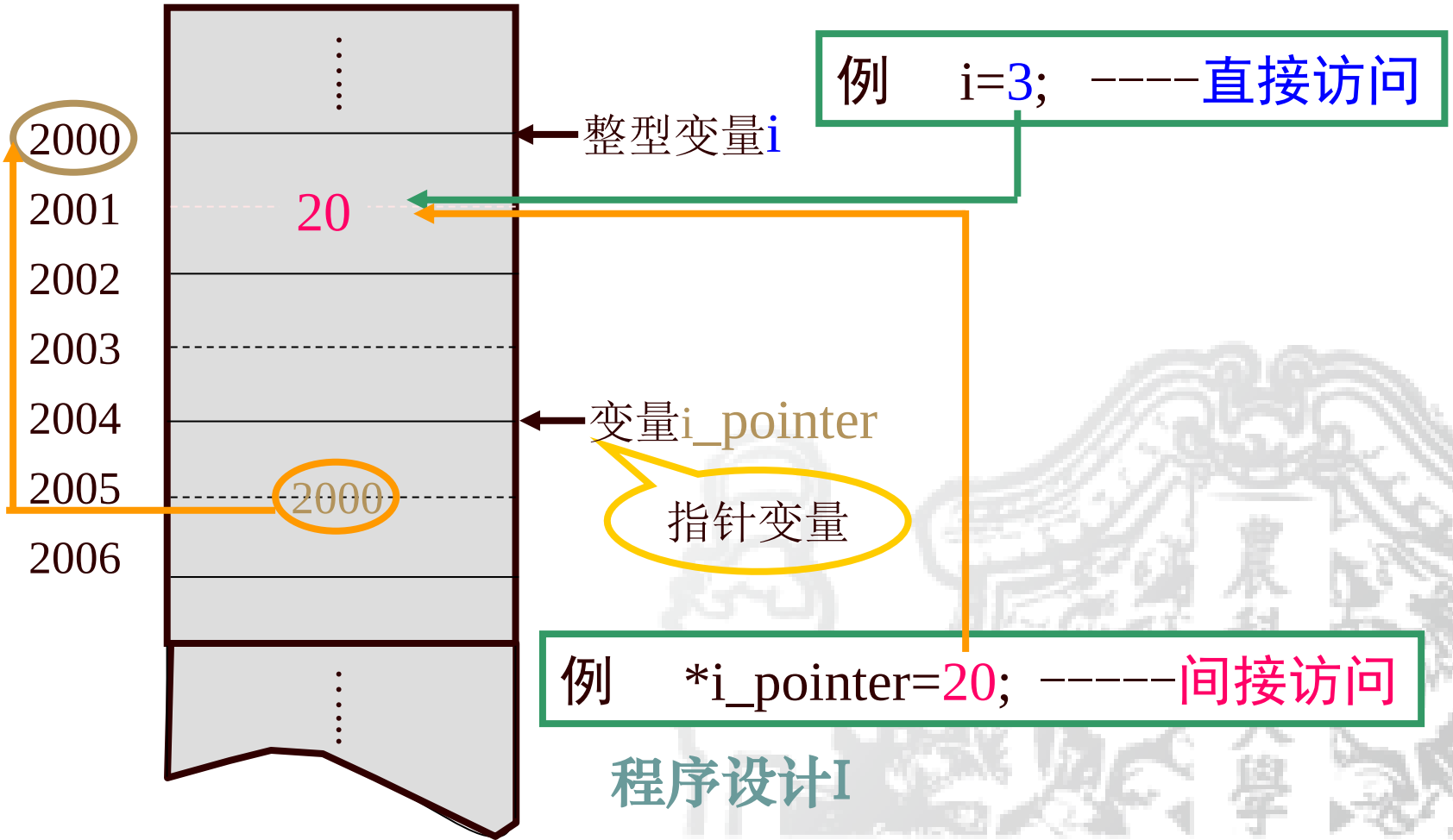
* p1++ → * (p1++)

意即：　　p1原指向a，现在指向下一个地址了。

- 直接访问与间接访问

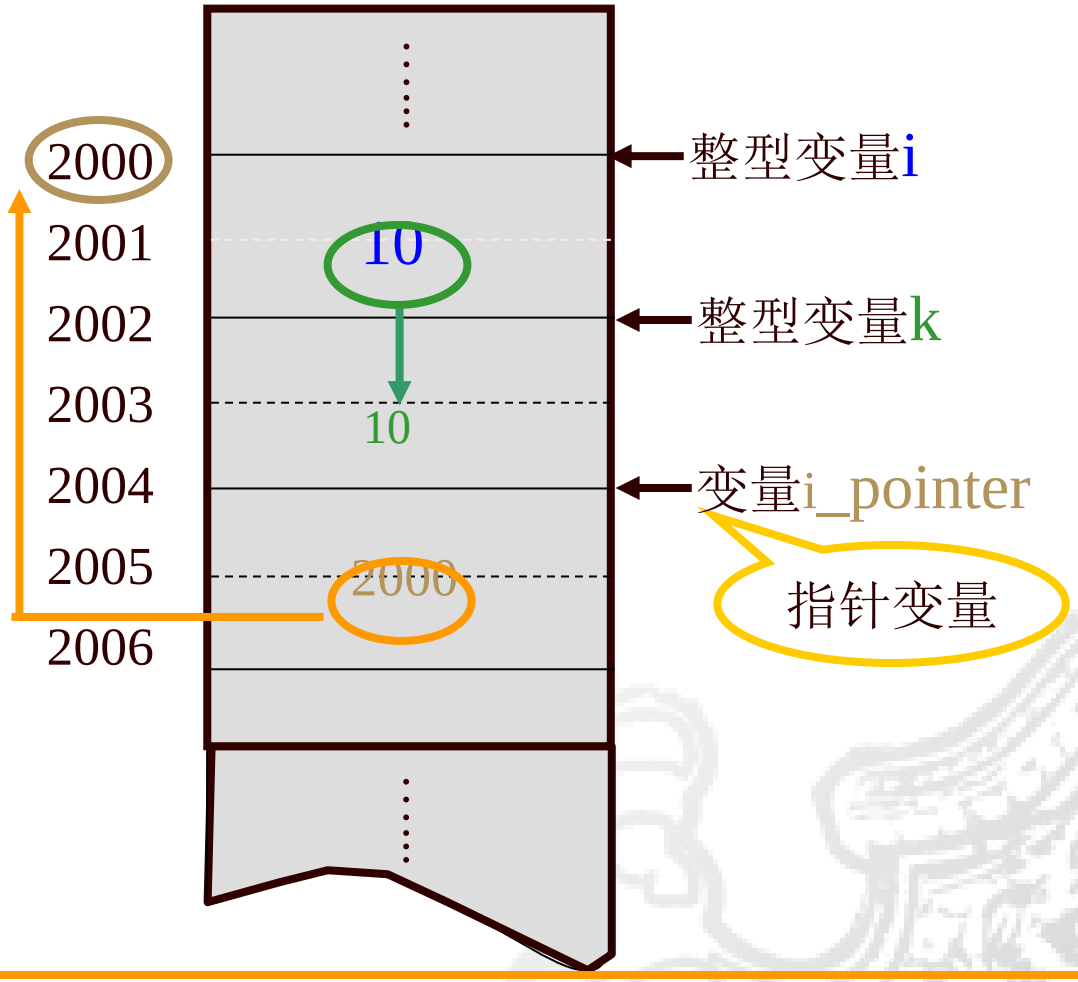
直接访问：按变量地址存取变量值

间接访问：通过存放变量地址的变量去访问变量



例　　k=i;
　　k=*i_pointer;

- 直接访问
- 间接访问



任何指针变量使用前要进行定义并赋值，否则禁止使用。

直接访问和间接访问

打个比方

间接访问的过程是：由指针得到变量的地址，根据该地址找到变量的存储区，再对该存储区的内容进行存取，从而实现了对变量的间接访问。

直接访问和间接访问

有两个上锁且放着物品的盒子A、B，

如果你有A盒子的钥匙，则可以直接打开A盒子将物品取出；

如果你有B盒子的钥匙，而A盒子的钥匙在B盒子中，要想取出A盒子中的物品，则必须先打开B盒子，取出A盒子的钥匙，再打开A盒子将物品取出。

上面两种情况就是直接访问和间接访问的概念。

间接访问的过程是：由指针得到变量的地址，根据该地址找到变量的存储区，再对该存储区的内容进行存取，从而实现了对变量的间接访问。

★指针变量的初始化

一般形式：[存储类型] 数据类型 *指针名=初始地址值;

赋给指针变量，
不是赋给目标变量

★指针变量的初始化

一般形式：[存储类型] 数据类型 *指针名=初始地址值;

```
例   int i;
     int  *p=&i;
```

变量必须已声明过
类型应一致

```
例       int  *p=&i;
         int i;
```

×

声明该指针变量时还未声明i

★指针变量的初始化

一般形式：[存储类型] 数据类型 *指针名=初始地址值;

```
例   int i;
       int  *p=&i;
```

变量必须已说明过
类型应一致

```
例   int i;
       int  *p=&i;
       int  *q=p;
```

用已初始化指针变量作初值

```
例   main( )
   {   int  i;
       static int  *p=&i;
       ..............
   }                (×)
```

不能用auto变量的地址
去初始化static型指针

★指针变量的初始化

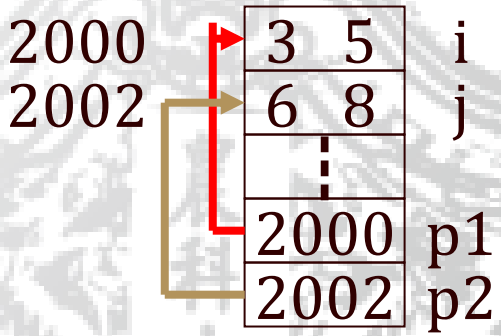
一般形式：[存储类型] 数据类型 *指针名=初始地址值;

例 int i;
int *p=&i;

例 int i;
int *p=&i;
int *q=p;

例 main( )
{ int i;
static int *p=&i;
.....
} (×)

指针变量赋值：
int i, j;
int *p1, *p2;
p1=&i; p2=&j ;
i=3; *p1=5;
j=6; *p2=8;



把一个指针的值赋给另一个指针，但不能直接用整型数据赋值给指针变量。 int *p=1000 !!!错误

```
main( ) {
int a=100 , b , *p ;
 p=&a ;
 printf("%d,%d\n" ,a ,*p);
*p=1000;
 printf("%d,%d\n",a,*p); }
```

100, 100
1000, 1000

```
main( ) {
int a=100 , b , *p ;
 *p=a ;
 printf("%d,%d\n" ,a ,*p);
*p=1000;
 printf("%d,%d\n",a,*p); }
```

使用未赋值的指针变量
类型不匹配

```
main( ) {
int  *p1,*p2,a=5;
float b=10.5;
*p1=a;
p2=&b;
printf("%d,%d\n",(*p1),(*p2));
}
```

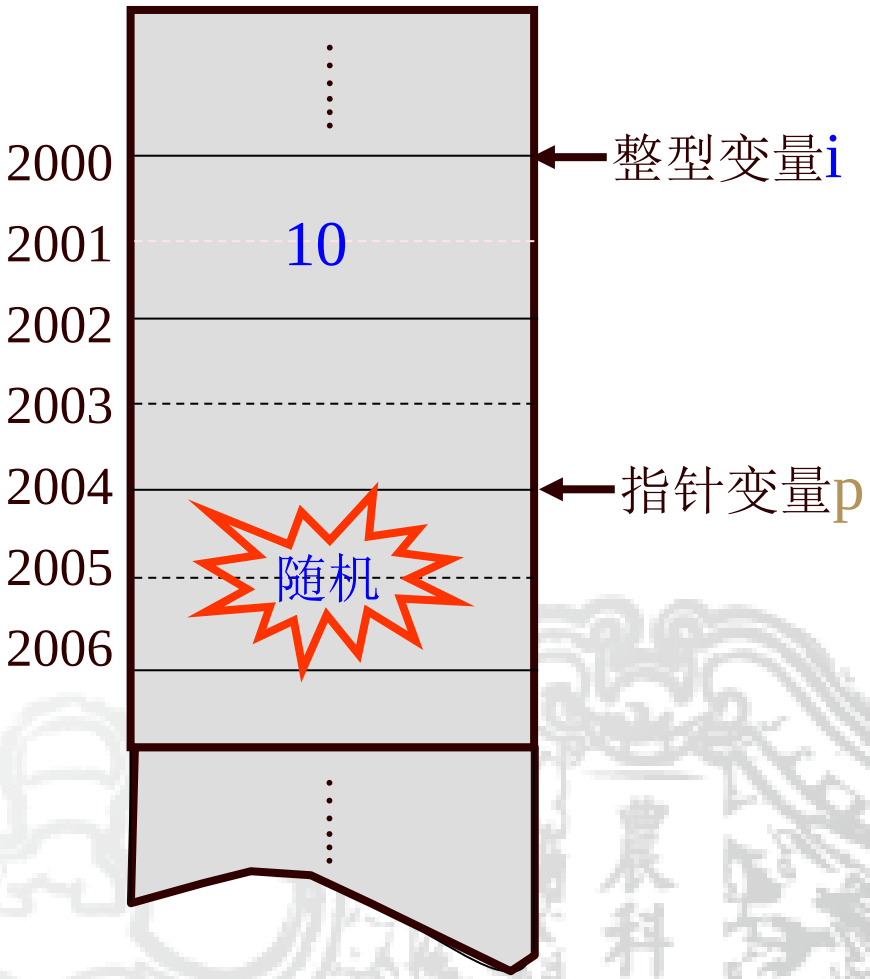
使用未赋值的指针变量
类型不匹配

指针变量必须先赋值，再使用

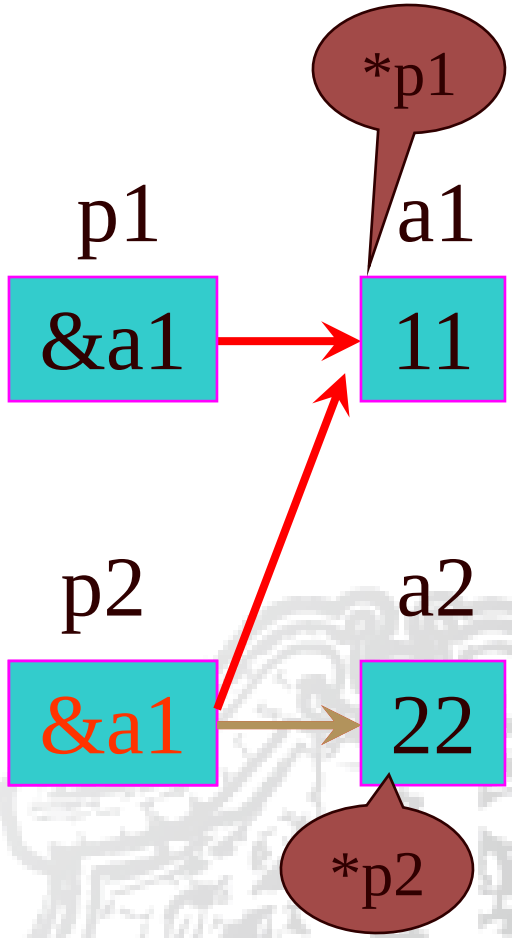
```
例   main( )
     {   int  i=10;
         int  *p;
         *p=i;
         printf("%d",*p);
     }
```

危险！

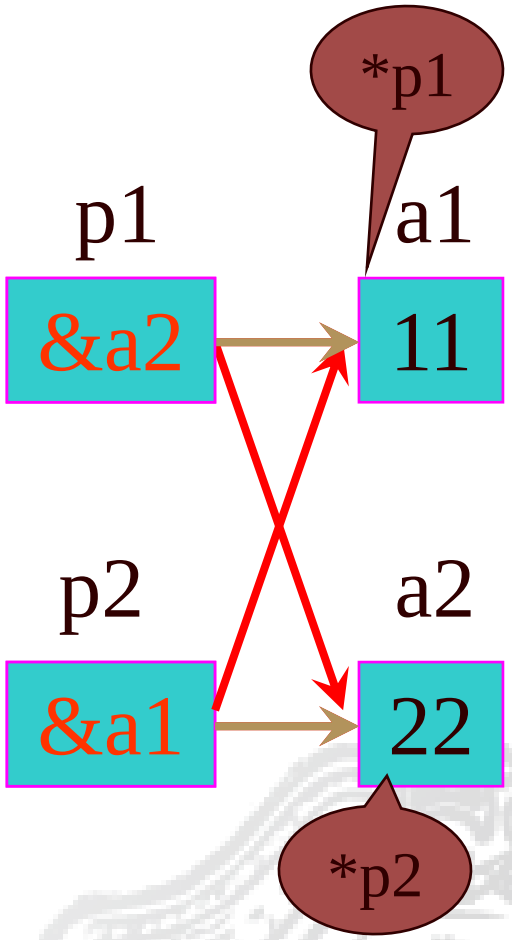
```
例   main( )
     {   int  i=10,k;
         int  *p;
         p=&k;
         *p=i;
         printf("%d",*p);
     }
```



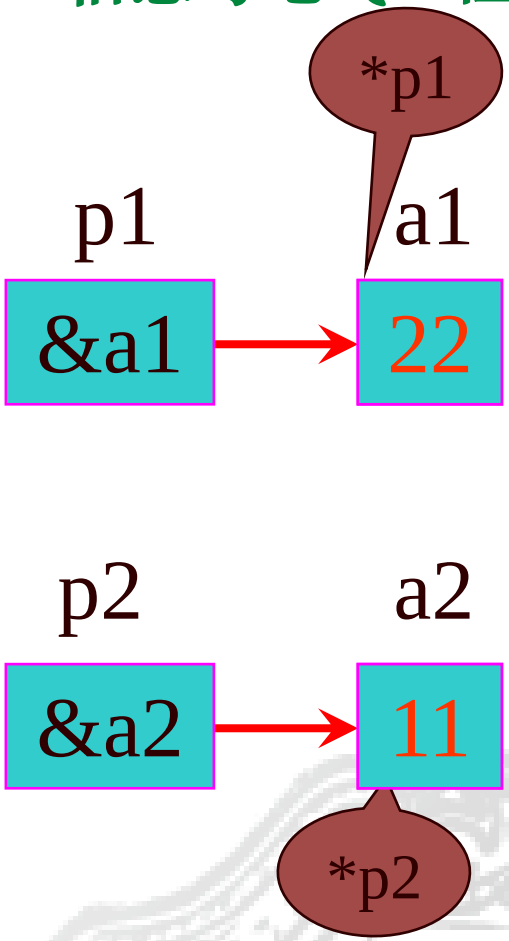
```
#include <stdio.h>
main( )
{
int  a1=11, a2=22;
int  *p1, *p2;
p1=&a1;
p2=&a2;
printf("%d,%d\n",*p1,*p2);
p2=p1;
printf("%d,%d\n",*p1,*p2);
}
```



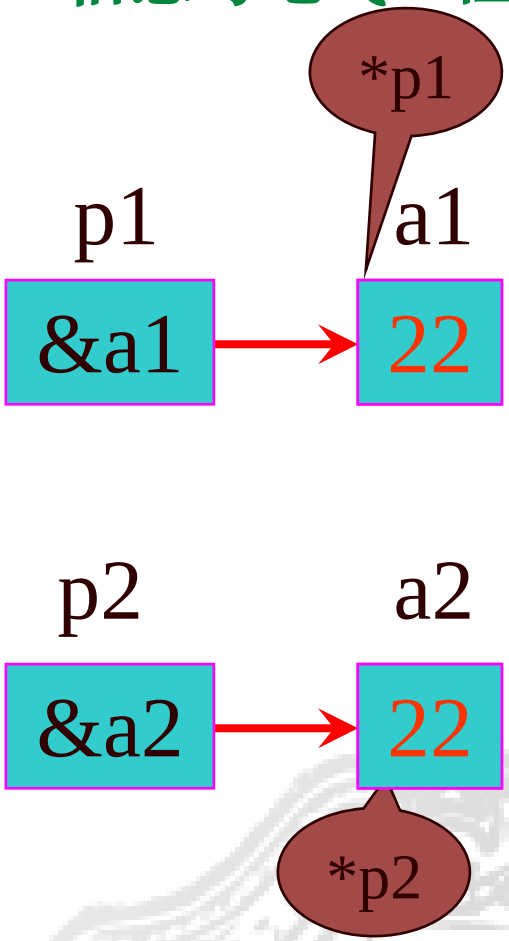
```
#include <stdio.h>
main( )
{
    int  a1=11,a2=22;
    int  *p1,*p2,*p;
    p1=&a1;
    p2=&a2;
    printf("%d,%d\n",*p1,*p2);
    p=p1; p1=p2; p2=p;
    printf("%d,%d\n",*p1,*p2);
}
```



```
#include <stdio.h>
main( )
{
int  a1=11,a2=22,t;
int  *p1,*p2;
p1=&a1;
p2=&a2;
printf("%d,%d\n",a1,a2);
t=*p1; *p1=*p2; *p2=t;
printf("%d,%d\n",a1,a2);
}
```



```
#include <stdio.h>
int main( )
{int  a1=11,a2=22;
int  *p1,*p2;
p1=&a1;
p2=&a2;
*p1 = a2;//相当于a1=a2;
*p2 = a1;
printf("%d,%d,%d,%d\n",a1,a2,*p1,*p2);
}
```



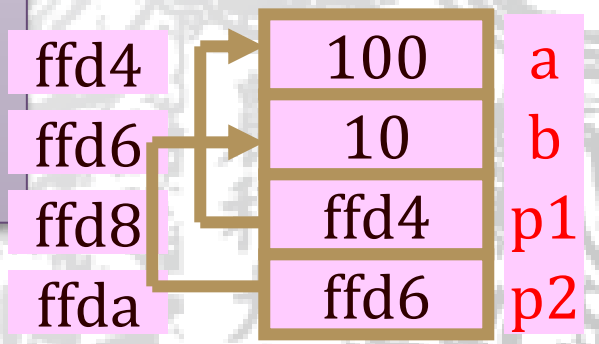
★指针变量的引用

例1 通过指针变量访问整型变量

```
#include <stdio.h>
int main( )
{int  a, b, *p1, *p2 ;
 a=100; b=10;
 p1=&a; p2=&b;
 printf("a=%d, b=%d\n",a, b);
 printf("*p1=%d, *p2=%d\n", *p1, * p2);
 printf("&a=%x,& b=%x\n",&a, &b);
 printf("p1=%x, p2=%x\n", p1,  p2);
 printf("& p1=%x, &p2=%x\n", &p1, & p2);
}
```

运行结果：

```
a=100, b=10
*p1=100, *p2=10
&a=ffd4, &b=ffd6
p1=ffd4, p2=ffd6
&p1=ffd8, &p2=ffda
```



例2 输入a和b两个整数，用指针方法按先大后小顺序输出

```
#include <stdio.h>
int main( )
{int  *p1, *p2, *p, a, b;
 scanf("%d%d",&a,&b);
 p1=&a;p2=&b;
 if(a<b)
   {p=p1; p1=p2; p2=p; }
 printf("\na=%d, b=%d\n", a, b);
 printf("max=%d, min=%d\n", *p1, *p2);
}
```

运行情况：

5,9 ↵

a=5, b=9

max=9,min=5

只交换了指针的值，
没有交换变量的值

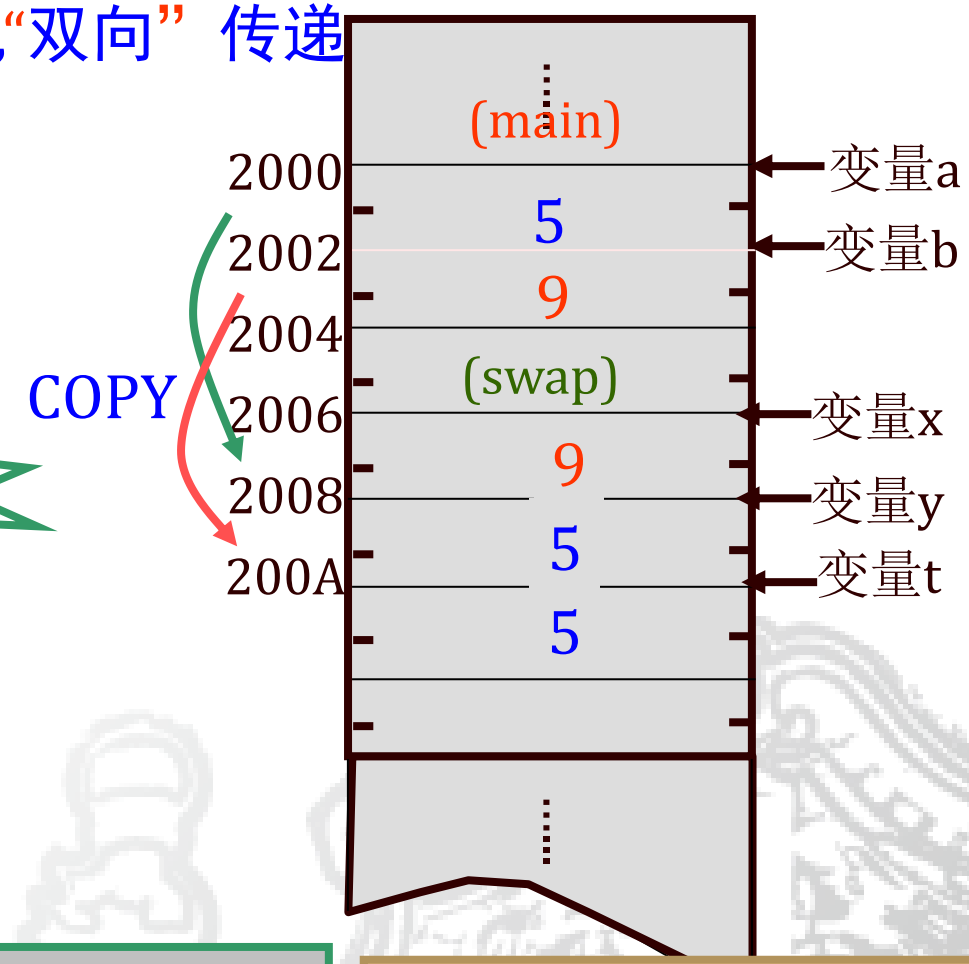
★指针变量作为函数参数——地址传递

❖特点：共享内存,“双向”传递

```
例3 将数从大到小输出
#include <stdio.h>
int swap(int  x,int y)
{ int  t;
  t=x;
  x=y;
  y=t; }

int main()
{ int a,b;
  scanf("%d,%d",&a,&b);
  if(a<b)  swap(a,b);
  printf("\n%d,%d\n",a,b);
}
```

值传递



运行结果：

5 , 9 ↵

5 , 9

函数调用结束后，分配给x，y，t单元释放

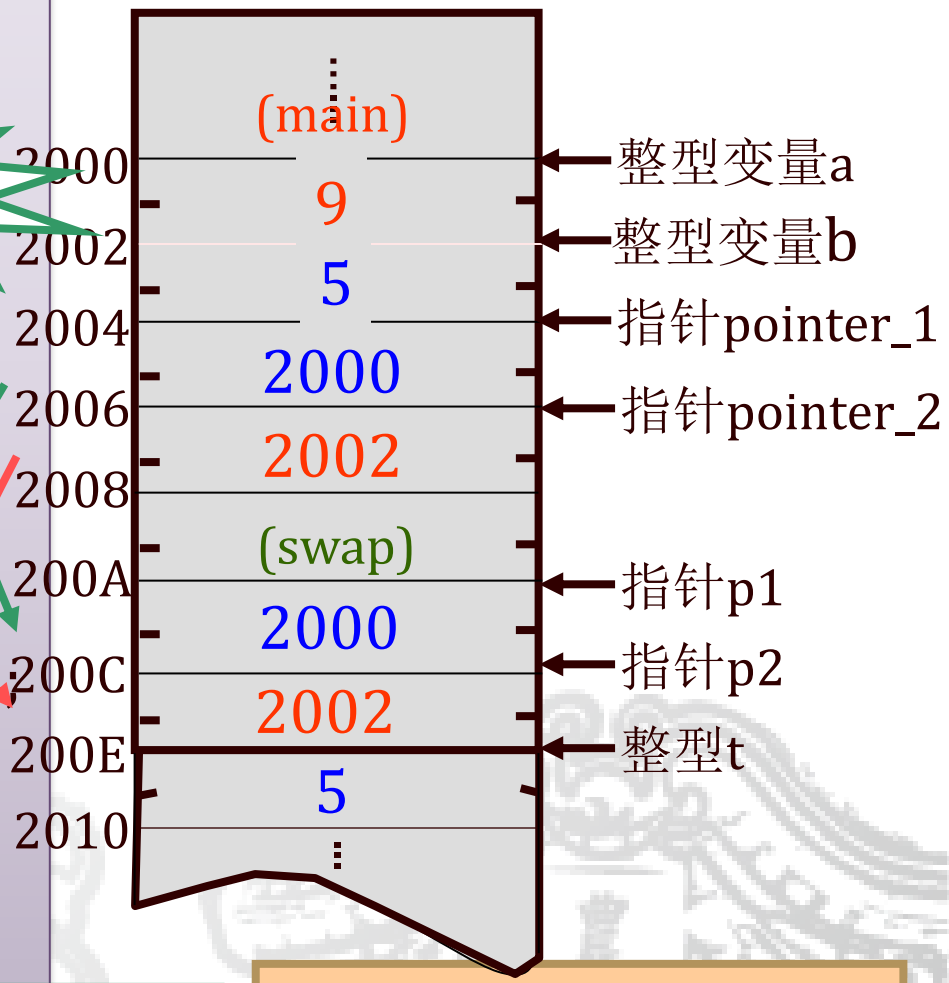
例3 将数从大到小输出
（使用指针变量作函数参数）

```
#include <stdio.h>
int main()
{ int swap(int  *p1, int  *p2);
  int a,b;
  int *pointer_1,*pointer_2;
  scanf("%d,%d",&a,&b);
  pointer_1=&a;  pointer_2=&b;
  if(a<b)swap(pointer_1,pointer_2);
  printf("\n%d,%d\n",a,b);}

int swap(int  *p1, int  *p2)
{ int t;
  t=*p1;
  *p1=*p2;
  *p2=t;}
```

地址传递

COPY



运行情况：

5,9 ↵

9,5

问题：函数调用结束后，分配给p1,p2,t单元释放否？

例3 将数从大到小输出

```
int swap(int *p1, int *p2)
{ int *p;
  *p=*p1;
  *p1=*p2;
  *p2=*p;
}

int main()
{ int a,b;
  int *pointer_1,*pointer_2;
  scanf("%d,%d",&a,&b);
  pointer_1=&a;  pointer_2=&b;
  if(a<b)  swap(pointer_1,pointer_2);
  printf("\n%d,%d\n",a,b);
}
```

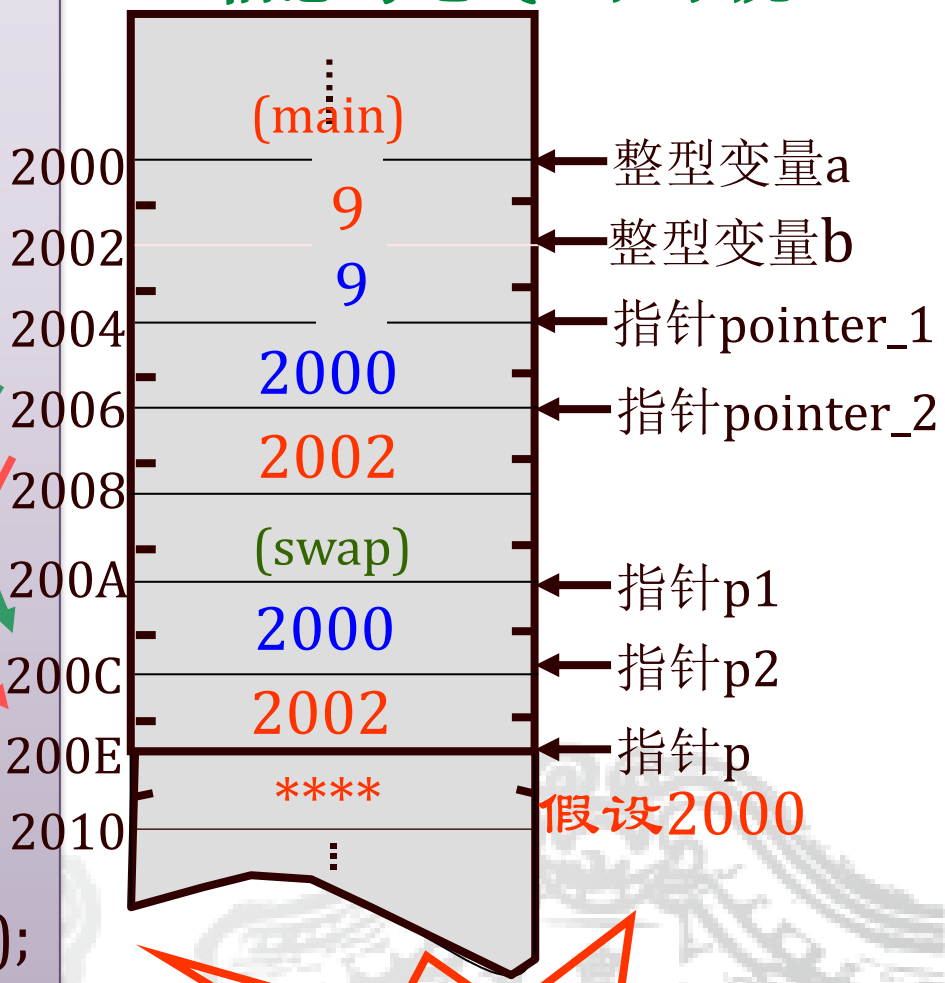
int x;

int *p=&x;

编译警告！

结果不对！

COPY



指针变量在使用前必须赋值！

运行结果：9，9

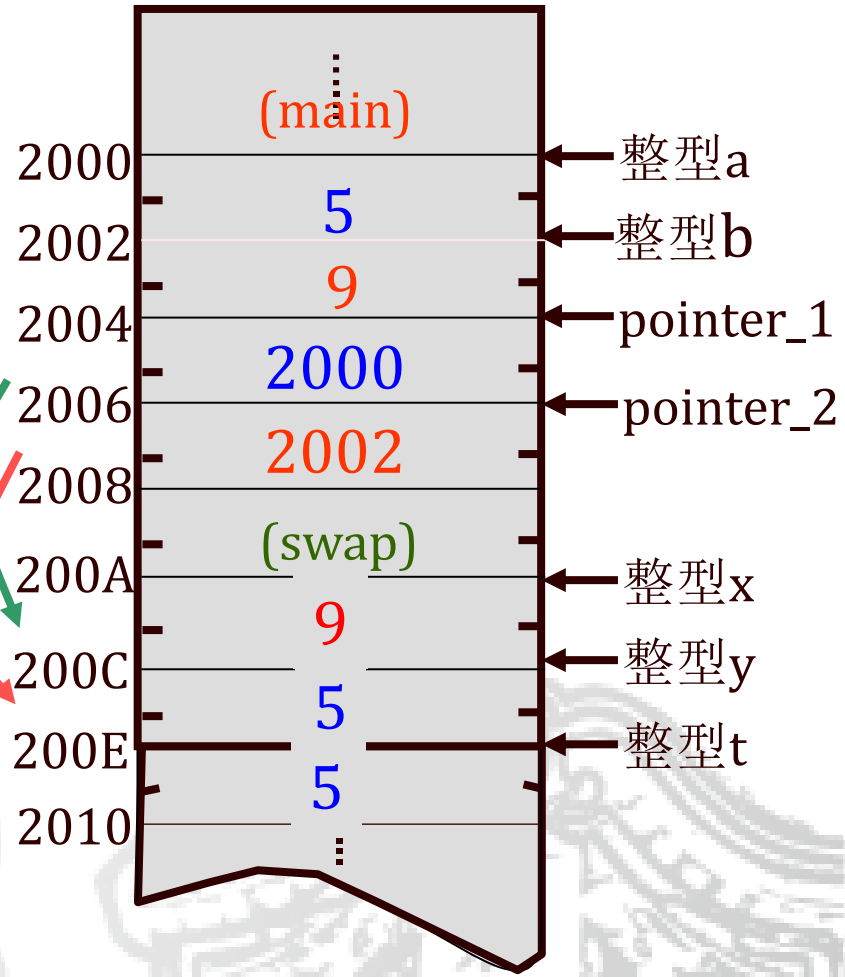
例3 将数从大到小输出

```
int swap(int x,int y)
{ int t;
  t=x; x=y;  y=t;
}

int main()
{ int a,b;
  int *pointer_1,*pointer_2;
  scanf("%d,%d",&a,&b);
  pointer_1=&a;  pointer_2=&b;
  if(a<b)
     swap(*pointer_1,*pointer_2);
  printf("\n%d,%d\n",a,b);
}
```

值传递

COPY



运行结果：5，9

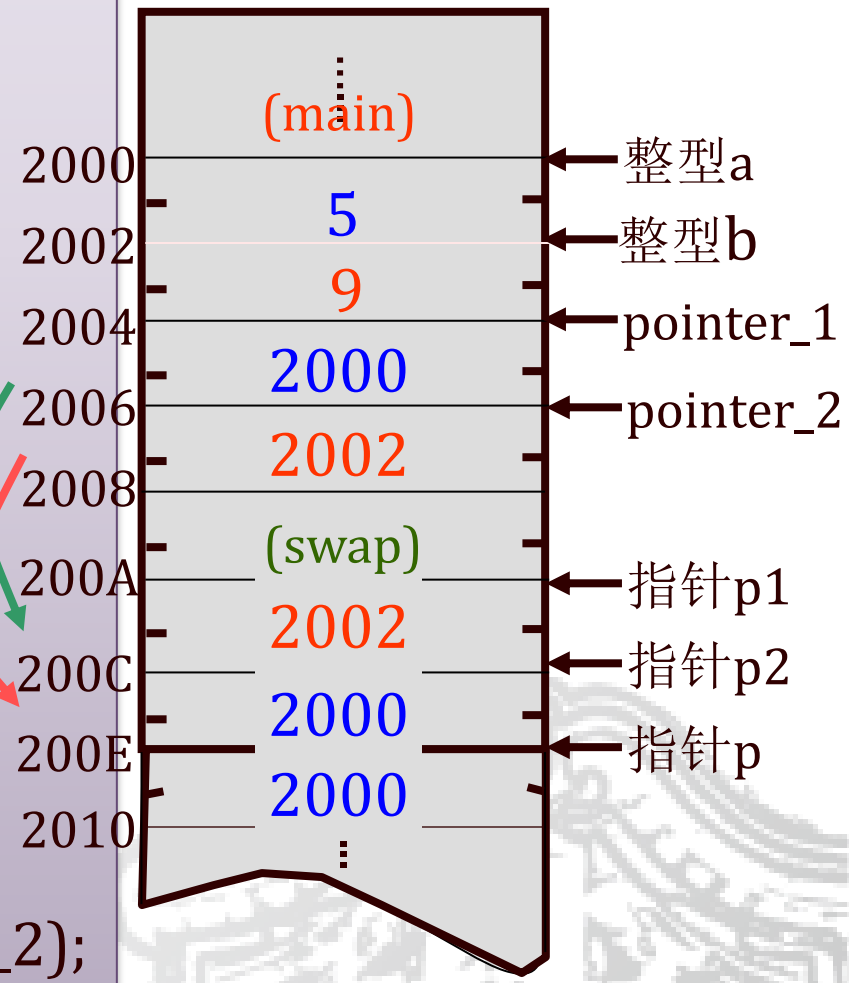
例3 将数从大到小输出

```
#include <stdio.h>
int swap(int *p1, int *p2)
{ int *p;
  p=p1;
  p1=p2;
  p2=p;}

int main()
{ int a,b;
  int *pointer_1,*pointer_2;
  scanf("%d,%d",&a,&b);
  pointer_1=&a;  pointer_2=&b;
  if(a<b)  swap(pointer_1,pointer_2);
  printf("%d,%d",*pointer_1,*pointer_2);
}
```

地址传递

COPY



运行结果：5，9

例4 输入a,b,c3个整数，按从大到小输出

```
#include <stdio.h>
int main( )
{int exchange(int *q1, int *q2, int *q3);
 int a,b,c,*p1,*p2,*p3;
 scanf("%d,%d,%d",&a,&b,&c);
 p1=&a; p2=&b; p3=&c;
 exchange(p1, p2, p3);
 printf("\n%d, %d, %d\n",a, b, c);}

int exchange(int *q1, int *q2, int *q3)
{void swap(int *pt1, int *pt2);
 if(*q1 < *q2) swap(q1, q2);
 if(*q1 < *q3) swap(q1, q3);
 if(*q2 < *q3) swap(q2, q3);}

void swap(int *pt1, int *pt2)
{int temp; temp=*pt1; *pt1=*pt2; *pt2=temp;}
```

运行情况：

9 , 0 , 10 ↵

10 , 9 , 0

ffbc	9		0	temp
ffc2	ffd0		ffd2	pt1
ffc4	ffd4		ffd4	pt2
ffca	ffd0			q1
ffcc	ffd2			q2
ffce	ffd4			q3
ffd0	9	10	10	a
ffd2	0	0	9	b
ffd4	10	9	0	c
ffd6	ffd0			p1
ffd8	ffd2			p2
ffda	ffd4			p3

§10.3 数组与指针

- **数组与一维数组**
- 指针变量只能进行赋值运算、部分算术运算及关系运算
- **数组与多维数组**

在C语言中，指针与数组有着密切的关系。对数组元素，既可以采用数组下标来引用，也可以通过指向数组元素的指针来引用。下标法直观，采用指针方法处理数组，可以产生代码长度小、占用内存少、运行速度快的程序。

10.3.1 指针与一维数组

- 数组的指针：指数组的起始地址。
- 数组元素的指针：指数组元素的地址。

  数组的地址→指针变量，指针变量就指向该数组了。
- 引用数组元素：

  ⑴下标法：a[3]

  ⑵指针法：用指针变量指向所找的数组元素。

  占内存少，运行速度快。

10.3.1 指针与一维数组

1、数组结构的分析

设有数组定义为：int a[5]; 则有：

(1) a表示数组在内存中的首地址，也就是数组中第0个元素的首地址，它是一个地址常量，其值由系统在编译时确定，程序运行期间不能改变。

(2) 数组中的各元素表示为：a[0]、 a[1] 、 a[2] 、 a[3]、a[4]

如果int *p=&a[0]; 等价于 p=a (p,a,&a[0])

p+i 指向a[i] ,*(p+i) 就是a[i]

同理 a+i是a[i]的地址，*(a+i)是a[i]的数组元素

*(a+0)（或*a）、*(a+1)、*(a+2)、*(a+3)、*(a+4)

★指向数组元素的指针

例：int a[10];

int *p;

p=&a[0]; 或 p=a; /*定义后赋值，两者等价*/

定义指针变量时赋初值：

例：int *p=&a[0];

int *p=a;

如 int i, *p;

p=1000; (×)

i=p; (×)

不能把一个整数⇒p,也不能把p的值⇒整型变量

通过指针引用数组元素

如果： int a[10];

int *p;

p=&a[1]; /* p指向数组元素a[1] */

则： *p=1

表示对p当前指向的数组元素a[1]赋予值1

而：p+1指向同一数组的下一个元素a[2]。

p的值（地址）加了2个字节，p+1=p+1×d（整型，d=2；实型，d=4；字符型d=1）指针变量所指数组元素的地址的计算，与数组数据类型有关。

设 p=&a[0]

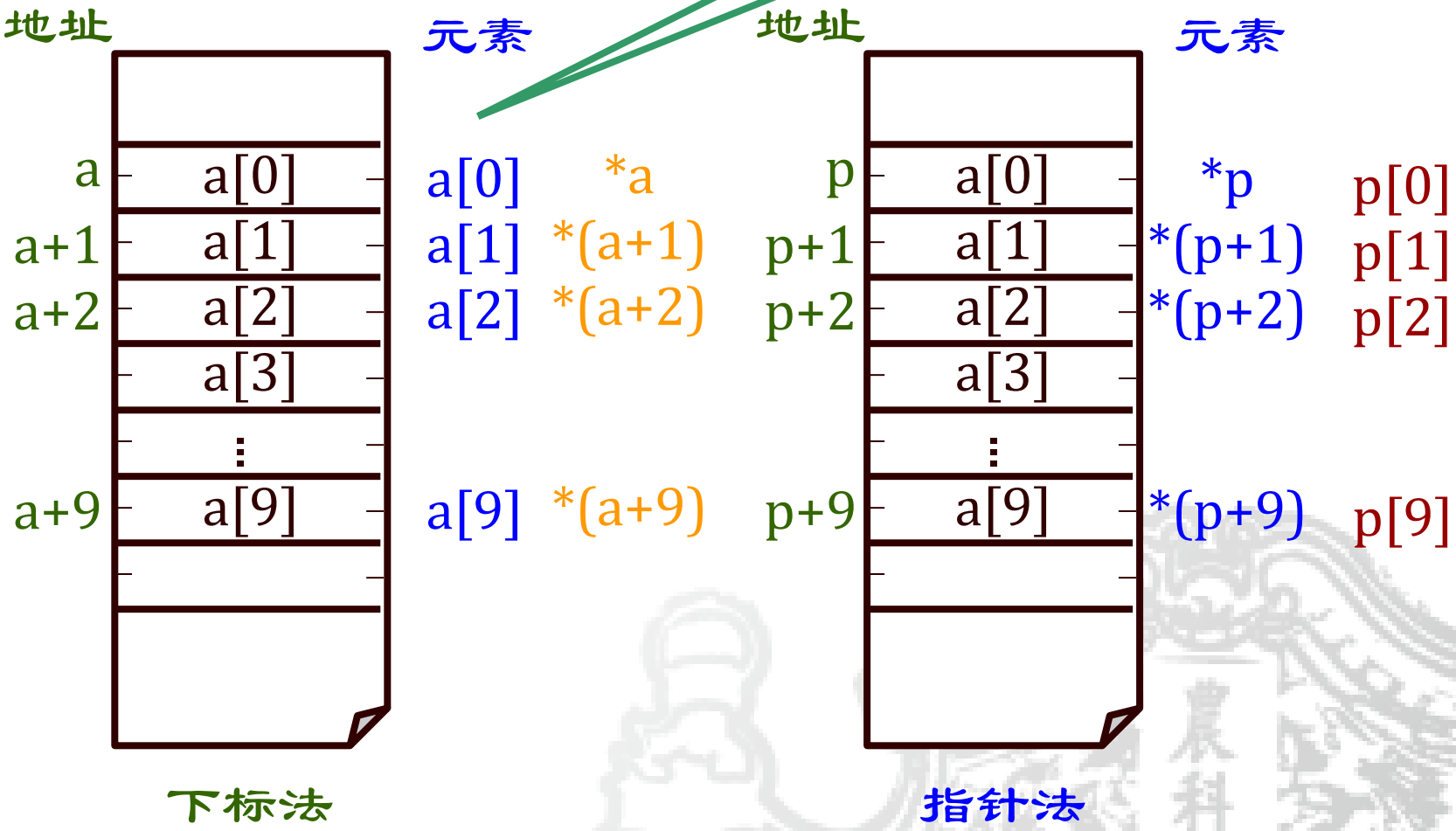
则 (1) p+i和a+i就是a[i]的地址a+i×d

(2) *(p+i)或*(a+i)是p+i或a+i指向的数组元素a[i]

(3) 指向数组的指针变量可带下标，p[i]与*(p+i)等价

表示数组元素的两种方法：

[] 变址运算符
a[i] ⇔ *(a+i)



a[i] ⇔ p[i] ⇔ *(p+i) ⇔*(a+i)

当一个指针变量指向数组的首地址后，关系图为：

指向数组元素的指针	数组int a[5]	指向数组元素的访问
a, &a[0], s, &s[0]	0	a[0], *a, s[0], *s
a+1, &a[1], s+1, &s[1]	1	a[1],*(a+1), s[1],*(s+1)
	2	
a+i, &a[i], s+i, &s[i]	3	a[i], *(a+i), s[i], *(s+i)
	4	

例5 int a[]={1,2,3,4,5,6,7,8,9,10},*p=a,i;

数组元素地址的正确表示：

（A）&(a+1)　（B）a++　（C）&p　√（D）&p[i]

指针变量是变量，值可以改变

p++,p--　(✓)

数组名是地址常量，值不能改变

a++,a--　(×)

a+1, *(a+2)　(✓)

2、指针的算术运算：按地址计算规则进行

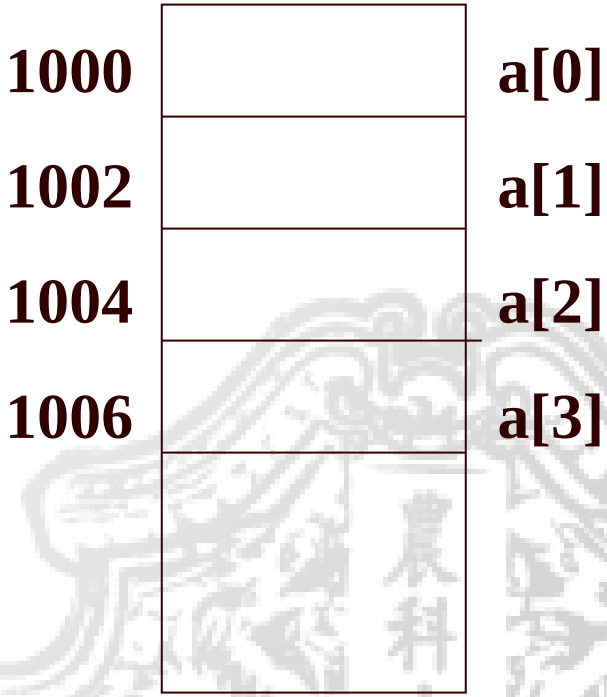
说明：指针的算术运算应考虑到指针所指向的数据类型。

(1)指针与整数的加、减运算

指向数组的指针变量，可加上或减去一个整数n,
例如 p+n　p-n　p++　++p　p--　--p是合法
含义：把指针指向的当前位置（指向某数组元素）向前或向后移动n个位置。

```
int a[5] , *p ;
p=a ; //p指向数组a，即指向数组元素a[0]
p++ ; //p指向下一个数组元素a[1]
p=a ;
p=p+2 ;//p指向a[2],即p的值为&a[2]
```

p+n的地址如下：
p中所存的地址+n*2



！！指针变量的加减运算只能对数组指针变量进行，对指向其他类型变量的指针变量进行加减运算是无意义的。

```
int main( )
{ int a[10]={0,1,2,3,4,5,6,7,8,9},x,y,*p ;
  p=&a[0] ;
  printf("%d %d %d\n",*p,*(p+2),*(p+5));
  x=*(p++) ;  //等价*(p++)
  y=*(++p); //等价 *(++p)
  printf("%d %d\n",x,y);  }
```

```
0   2   5
0   2
```

(1) 只有指向同一数组的两个指针变量之间才能进行运算，否则无意义；

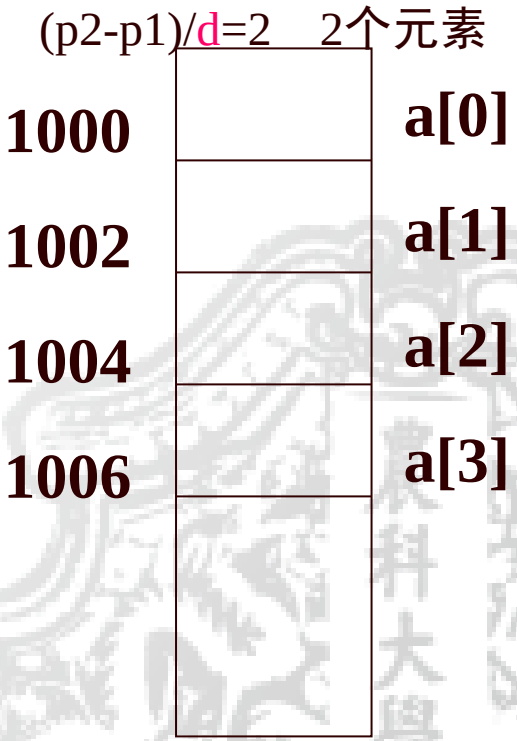
(2)*p++等价于*(p++)，先取p所指元素值，再把指针向后调整一个元素；

(3)*++p等价于*(++p)，先把指针向后调整一个元素，然后再取所指元素的值。

(2)两个指针相减：应用在字符串操作中

用于同类型的指针变量，并且两个指针变量指向同一个数组中的数组元素。

两指针变量相减的差是两个指针所指数组元素之间相差的元素个数。



！！两个指针变量不能进行相加，p1+p2是无意义的。

3、指针的关系运算：两个指针变量所指向地址的比较

要求：进行关系运算的两指针必须指向同一数组

p1和p2指向同一数组a，p1=&a[1]　p2=&a[5]

p1==p2　只有p1和p2指向同一数组元素时为真
p1<p2　p1指向的数组元素在p2指向的之前为真
p1<=p2　p1指向的数组元素在p2指向的之前或相同为真
p1>p2　p1>=p2　p1!=p2
p==0　或　p==NULL　或　p!=NULL
表明p是空指针，不指向任何变量。

使用指针变量的常见错误

1）使用未初始化的指针变量。
2）指针变量所指向的数据类型与其定义的类型不符。

1、若有定义：int x, *pb;则以下正确的赋值表达式为
[ ]
A）pb=&x B）pb=x C）*pb=&x D）*pb=*x E）*pb=x

2．执行以下程序后，a的值为【0　　】，　b 的值为【7】

```
main()
{  int  a,b,k=4,m=6;
   int  *p1=&k, *p2=&m;
   a=(p1==&m);
   b=(-1**p1)/(*p2)+7;}
```

例6 用三种方法输出数组中全部元素的值

(1) 下标法：

```
#include <stdio.h>
int main()
{int a[10];
 int i;
 for(i=0; i<10; i++)
   scanf("%d",&a[i]);
 printf("\n");
 for(i=0; i<10; i++)
   printf("%d",a[i] );}
```

(2) 用数组名：

```
#include <stdio.h>
int main()
{int a[10];
 int i;
 for(i=0; i<10; i++)
   scanf("%d",&a[i]);
 printf("\n");
 for(i=0; i<10; i++)
  printf("%d",*(a+i));}
```

(3) 指针法：

```
#include <stdio.h>
int main()
{int a[10];
 int *p,I;
 for(i=0; i<10; i++)
   scanf("%d",&a[i]);
 printf("\n");
 for(p=a; p<(a+10); p++)
    printf("%d",*p );}
```

运行情况：

1 2 3 4 5 6 7 8 9 0↵

1 2 3 4 5 6 7 8 9 0

例6 用三种方法输出数组中全部元素的值

(4) 指针法和指针下标：

```
#include <stdio.h>
int main()
{int a[10];
 int *p,I;
 for(i=0; i<10; i++)
   scanf("%d",&a[i]);
 printf("\n");
 for(p=a,i=0; i<10; i++)

printf("%d",*(p+i));}
```

运行情况：

1 2 3 4 5 6 7 8 9 0↵
1 2 3 4 5 6 7 8 9 0

使用指针变量时要注意的问题：

(1) p++: 合法，因为p是指针变量，++只能用于变量。

a++: 不合法，因为a是数组名，其值是数组元素的首地址，是常量，程序运行期间值固定不变。

(2) 指针变量使用时要注意当前值（见例10.6）.

例7 用指针变量输出元素

```
#include <stdio.h>
int main()
{ int *p,i,a[10];
  p=a;
  for(i=0;i<10;i++)
    scanf("%d",p++);
  printf("\n");
  for(i=0;i<10;i++,p++)
    printf("%d",*p);
  printf("\n");
}
```

原因？

能正确输出吗？

运行情况：

1 2 3 4 5 6 7 8 9 0↵

22153 234 0 0 30036 25202 11631 8259 8237 28483

例7 用指针变量输出元素

```
#include <stdio.h>
int main()
{ int *p,i,a[10];
  p=a;
  for(i=0;i<10;i++)
    scanf("%d",p++);
  printf("\n");
  p=a;    /*或者p=&a[0]*/
  for(i=0;i<10;i++,p++)
    printf("%d",*p);
  printf("\n");
}
```

运行情况：

1 2 3 4 5 6 7 8 9 0↵

1 2 3 4 5 6 7 8 9 0

使用指针变量时要注意的问题：

(3) 从上例可知，指针变量p可以指向数组以后的内存单元编译不作检查。

(4) 指针变量运算时要注意的几个问题：如果p=a

① p++（或p+=1），使p指向下一元素a[1]。则*p值成为a[1]。

② ++和* 优先级同为2,结合性从右向左，则*p++等价于*(p++)，即先得到p指向的变量的值*p，再使p=p+1。例10.6可改为：

例7 用指针变量输出元素

```
#include <stdio.h>
int main()
{ int *p,i,a[10];
  p=a;
  for(i=0;i<10;i++)
    scanf("%d",p++);
  printf("\n");
  p=a;    /*或者p=&a[0]*/
  for(i=0;i<10;i++,p++)
    printf("%d",*p);
  printf("\n");
}
```

例7 用指针变量输出元素

```
#include <stdio.h>
int main()
{ int *p,i,a[10];
  p=a;
  for(i=0;i<10;i++)
    scanf("%d",p++);
  printf("\n");
  p=a;
  for(i=0;i<10;i++)
    printf("%d",*p++);
  printf("\n");
}
```

使用指针变量时要注意的问题：

(3) 从上例可知，指针变量p可以指向数组以后的内存单元编译不作检查。

(4) 指针变量运算时要注意的几个问题：如果p=a

① p++（或p+=1），使p指向下一元素a[1]。则*p值成为a[1]。

② ++和* 优先级同为2,结合性从右向左，则*p++等价于*(p++)，即先得到p指向的变量的值*p，再使p=p+1。例10.6可改为：

③ *(p++)与*(++p)作用不同：

*(p++)先取*p值，再p加1；*(++p)是p先加1，再取*p

若p初值&a[0]，则*(p++)得a[0]值，*(++p)得a[1]值。

④ (*p)++表示p所指向的元素值加1。

⑤ 如果p指向a[i]元素，则：

*(p--)相当于a[i--]，先对p进行 * 运算，再使p自减（加）。

*(++p)相当于a[++i]，先使p自加（减），再作 * 运算。

*(--p)相当于a[--i]，先使p自减（加），再作 * 运算。

使用指针变量时要注意的问题：

(3) 从上例可知，指针变量p可以指向数组以后的内存单元编译不作检查。

(4) 指针变量运算时要注意的几个问题：如果p=a

① p++（或p+=1），使p指向下一元素a[1]。则*p值成为a[1]。

② ++和* 优先级同为2 结合性从右向左 则*p++等价于*(p++)，

例 输出a数组100个元素

```
p=a;                          p=a;
while(p<a+100)          或    while(p<a+100)
  printf("%d",*p++);            { printf("%d",*p);
                                  p++;}
```

可改为：

⑤ 如果p指向a[i]元素，则：

*(p--)相当于a[i--]，先对p进行 * 运算，再使p自减（加）。

*(++p)相当于a[++i]，先使p自加（减），再作 * 运算。

*(--p)相当于a[--i]，先使p自减（加），再作 * 运算。

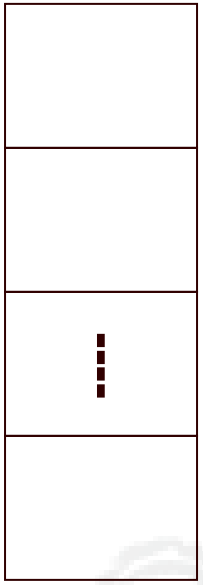
10.3.2 数组名作函数参数

- 数组名、指针变量实质上都是地址的传递，用户可根据需要自己设定.
- 如果有实参数组，想在函数中改变此数组的元素的值，实参与形参的对应关系有以下4种:

实参	形参
数组名	数组名
数组名	指针变量
指针变量	数组名
指针变量	指针变量

数组名作函数参数，是地址传递

当用数组名做函数实参时相当于将数组的首地址传给被调函数的形参，此时，形参数组和实参数组占用的是同一段内存，所以当在被调函数中对形参数组元素进行修改时，实参数组中的数据也将被修改，因为它们是同一个地址。




```
数组名作函数参数
int main( )
{f(int arr[ ], int n);
 int  array[10];
        ⋮
 f(array, 10);
        ⋮
}
int f(int arr[ ], int n)
{
 ⋮
}
```

编译时arr按指针变量处理，所以，此句与f(int *arr , int n)等价。

例8 将数组a中n个整数按相反顺序存放。

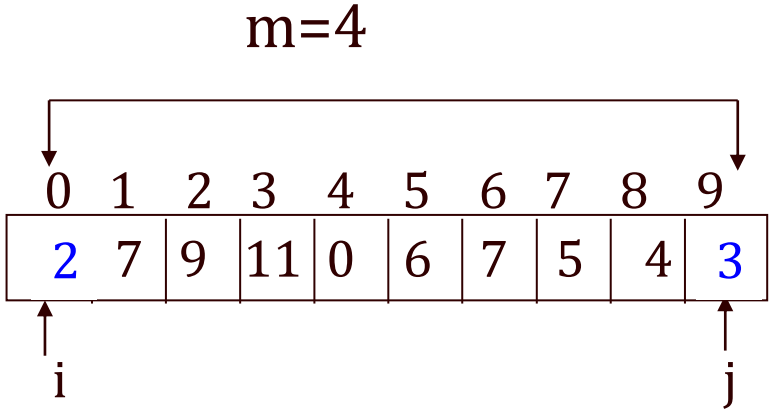
思路：数组元素头尾对调。四种调用方式。

m=4

0	1	2	3	4	5	6	7	8	9
3	7	9	11	0	6	7	5	4	2

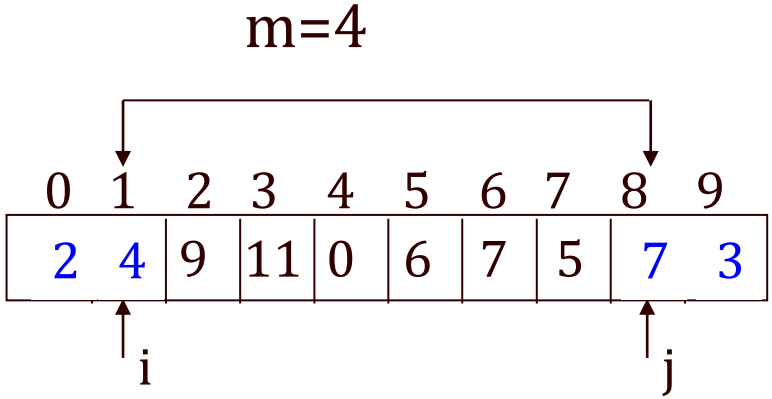
例8 将数组a中n个整数按相反顺序存放。

思路：数组元素头尾对调。四种调用方式。



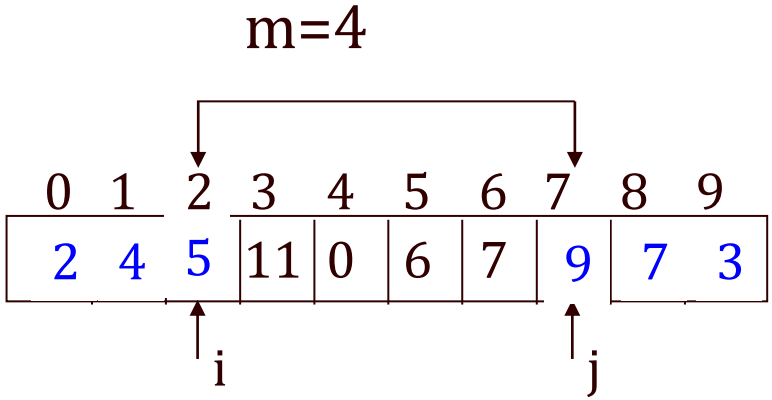
例8 将数组a中n个整数按相反顺序存放。

思路：数组元素头尾对调。四种调用方式。



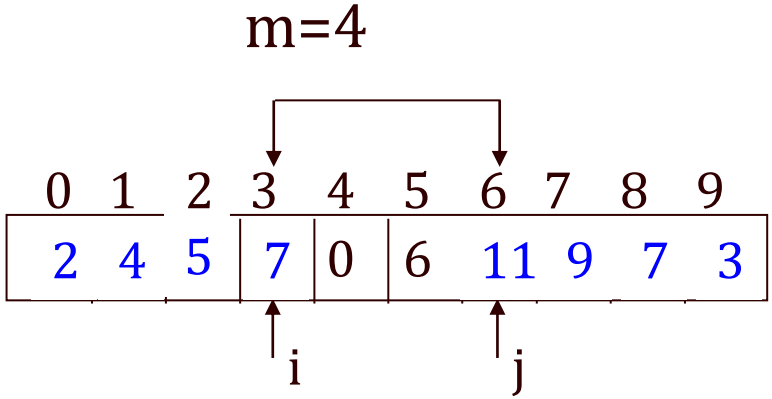
例8 将数组a中n个整数按相反顺序存放。

思路：数组元素头尾对调。四种调用方式。



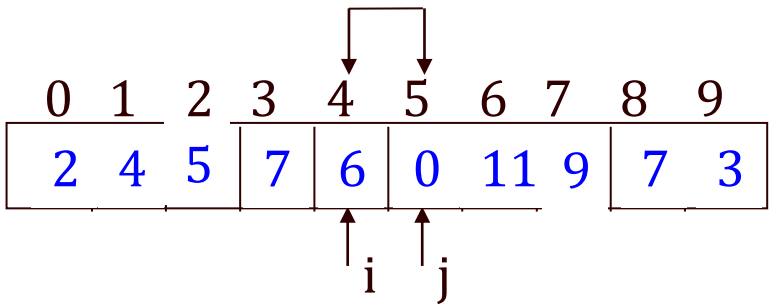
例8 将数组a中n个整数按相反顺序存放。

思路：数组元素头尾对调。四种调用方式。



例8 将数组a中n个整数按相反顺序存放。

思路：数组元素头尾对调。四种调用方式。



例8 将数组a中n个整数按相反顺序存放。

思路：数组元素头尾对调。四种调用方式。

m=4

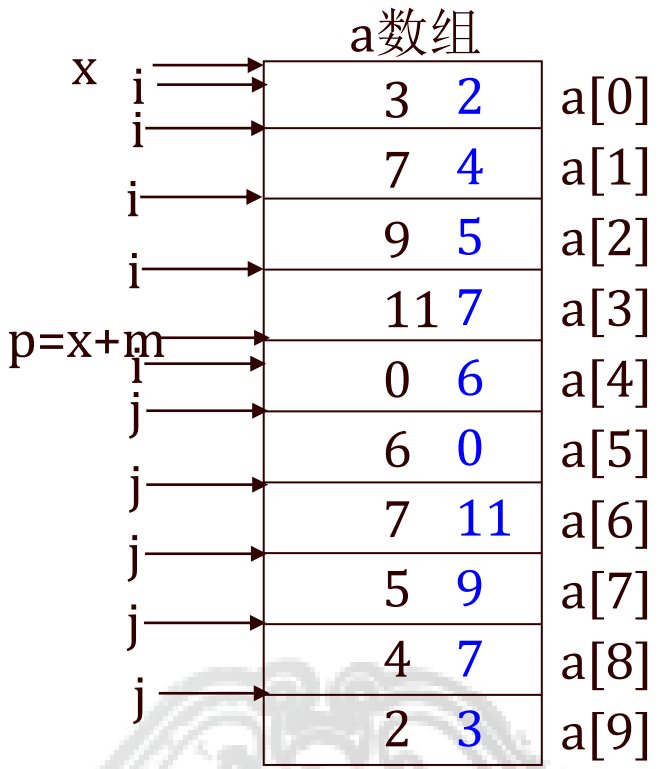
0	1	2	3	4	5	6	7	8	9
2	4	5	7	6	0	11	9	7	3

(1) 实参与形参均用数组

```
#include <stdio.h>
int main()
{int inv(int  x[ ], int n);
 int i,a[10]={3,7,9,11,0,6,7,5,4,2};
 printf("The original array:\n");
 for(i=0;i<10;i++)   printf("%d,",a[i]);
 printf("\n");
 inv(a,10);
 printf("The array has been inverted:\n");
 for(i=0;i<10;i++)   printf("%d,",a[i]);
 printf("\n");
}
void inv(int  x[ ], int n)
{ int temp,i,j,m=(n-1)/2;
 for(i=0;i<=m;i++)
 { j=n-1-i;
   temp=x[i];  x[i]=x[j];  x[j]=temp; }
 return;
}
```

(2) 实参用数组，形参用指针变量

```
#include <stdio.h>
int main()
{int inv(int  *x, int n);
  int i,a[10]={3,7,9,11,0,6,7,5,4,2};
  printf("The original array:\n");
  for(i=0;i<10;i++)   printf("%d,",a[i]);
  printf("\n");
  inv(a,10);
  printf("The array has been inverted:\n");
  for(i=0;i<10;i++)   printf("%d,",a[i]);
  printf("\n");
}
int inv(int  *x, int n)
{ int temp,*p,*i,*j,m=(n-1)/2;
  i=x;  j=x+n-1;  p=x+m;
  for(;i<=p;i++,j--)
    { temp=*i;  *i=*j;  *j=temp; }
  return;
}
```



(3) 实参与形参均用指针变量

```
#include <stdio.h>
int main()
{int inv(int *x, int n);
  int i,arr[10],*p=arr;
  printf("The original array:\n");
  for(i=0;i<10;i++,p++)
     scanf("%d",p);
  p=arr;
 inv(p,10);
  printf("The array has been inverted:\n");
  for(p=arr;p<arr+10;p++)
     printf("%d",*p);
  printf("\n");
}
void inv(int *x, int n)
{ int *p, m, temp,*i,*j;
   m=(n-1)/2;
  i=x;  j=x+n-1;  p=x+m;
  for(;i<=p;i++,j--)
  { temp=*i; *i=*j; *j=temp; }
  return;}
```

此句用意？

⑷ 实参用指针变量，形参用数组

```
#include <stdio.h>
int main()
{int inv(int  x[ ], int n);
 int i,a[10],*p=a;
 for(i=0;i<10;i++,p++)
   scanf("%d",p);
 p=a;
 inv(p,10);
 printf("The array has been inverted:\n");
 for(p=arr;p<arr+10;p++)
   printf("%d ",*p);
 printf("\n ");
}
int inv(int  x[ ], int n)
{ int t,i,j,m=(n-1)/2;
 for(i=0;i<=m;i++)
  { j=n-1-i;
    t=x[i];  x[i]=x[j];  x[j]=t; }
 return;
}
```

例9 从10个数中找出其中最大值和最小值

为了得到两个结果值，用两个全局变量max和min。

⑴ 实参和形参均用数组

```
int  max, min;       /* 全局变量*/
int max_min_value(int array[ ],int n)
{ int  *p, *array_end;
  array_end=array+n;   /*指向数组最后一个元素的后面 */
  max=min=*array;      /* 相当于max=min=array[0] */
  for(p=array+1; p<array_end; p++)    /* p指向array[1] */
     if(*p > max) max=*p;
     else if(*p<min) min=*p;  }
int  main( )
{ int  i, number[10];
  printf("enter 10 integer numbers:\ n");
  for(i=0;i<10;i++)
     scanf("%d",&number[i]);
  max_min_value(number,10);
  printf("\nmax=%d, min=%d\n", max, min);
}
```

(2) 实参和形参均用指针变量

```
int  max, min;       /* 全局变量*/
int max_min_value(int *array,int n)
{ int  *p, *array_end;
  array_end=array+n;  /*指向数组最后一个元素的后面 */
  max=min=*array;     /* 相当于max=min=array[0] */
  for(p=array+1; p<array_end; p++)   /* 使p指向array[1] */
    if(*p > max) max=*p;
    else  if(*p<min) min=*p;  }
int main( )
{ int  i, number[10],*p;
  p=number;    /* 使p指向number数组 */
  printf("enter 10 integer numbers:\ n");
  for(i=0;i<10;i++,p++)
    scanf("%d",p);
  printf("the 10 integer numbers:\n");
  for(p=number,i=0;i<10;i++,p++)  printf("%d ",*p);
  p=number;
  max_min_value(p,10);
  printf("\nmax=%d, min=%d\n", max, min);  }
```

归纳：用数组做函数参数有如下四种情况：

1、实参形参都用数组名：

```
int a[10];                    inv(int x[ ],int n)
inv(a,10)                     { ..... }
```

2、实参用数组名，形参用指针变量：

```
int a[10];                    inv(int *x,int n)
inv(a,10)                     { ..... }
```

3、实参形参都用指针变量：

```
int a[10];                    inv(int *x,int n)
int *p=a;                     {.....}
inv(p,10)
```

4、实参用指针变量，形参用数组名：

```
int a[10];                    inv(int x[ ],int n)
int *p=a;                     {.....}
inv(p,10)
```

❖一级指针变量与一维数组的关系

int *p 与 int q[10]

- 数组名是指针（地址）常量
- p=q; p+i 是q[i]的地址
- 数组元素的表示方法:下标法和指针法，

  即：若p=q,

  则：p[i] ⇔ q[i] ⇔ *(p+i) ⇔ *(q+i)
- 形参数组实质上是指针变量。

  即：int q[ ] ⇔ int *q
- 在定义指针变量（不是形参）时，不能把int *p 写成int p[];
- 系统只给p分配能保存一个指针值的内存区(一般2字节）；而给q分配2*10字节的内存区

例10 用选择法对10个整数排序

(2) 实参和形参均用指针变量

```
int sort(int  *x, int n)
{ int i,j,k,t;
  for(i=0;i<=n-1;i++)
    { k=i;
      for(j=i+1;j<n;j++)
        if(*(x+j)>*(x+k)) k=j;
      if(k!=i)
      {t=*(x+i);*(x+i)=*(x+k);*(x+k)=t;}
    }
}
```

(1) 实参用指针变量，形参用数组

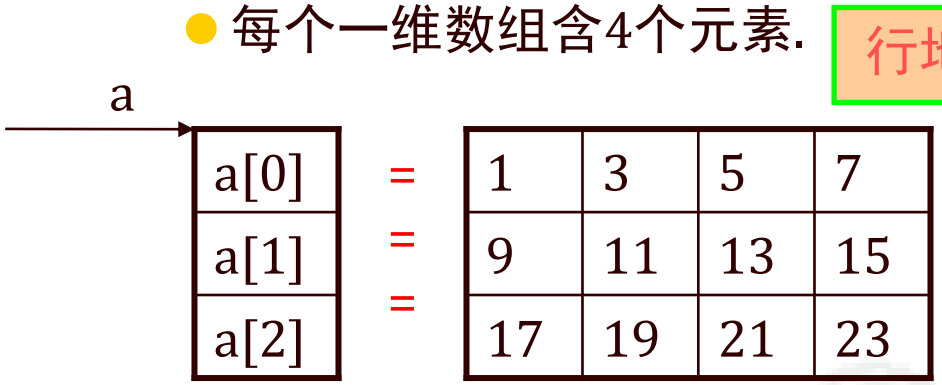
```
#include <stdio.h>
int main()
{int sort(int  x[ ], int n);
  int *p,i,a[10];
  p=a;
  for(i=0;i<10;i++)
     scanf("%d",p++);
  p=a;    sort(p,10);
  for(p=a,i=0;i<10;i++)
     {printf("%d ",*p); p++;}
}
int sort(int  x[ ], int n)
{ int i,j,k,t;
  for(i=0;i<=n-1;i++)
    { k=i;
      for(j=i+1;j<n;j++)
        if(x[j]>x[k]) k=j;
      if(k!=i)
       {t=x[i];x[i]=x[k];x[k]=t;}
} }
```

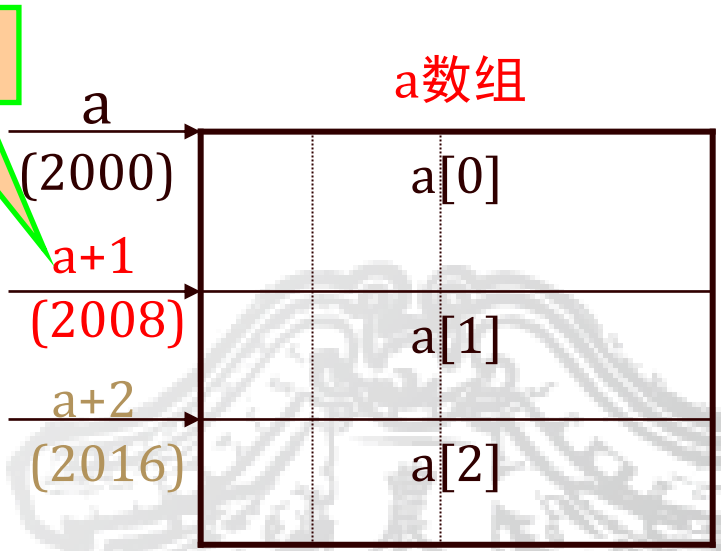
10.3.3 指针与多维数组

❖多维数组的地址（以二维数组为例）

如：int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};

- 多维数组数据存储按先行后列顺序。
- 每行都是一个一维数组，a[0],a[1],a[2]是一维数组名。
- 每个一维数组含4个元素.





a:二维数组首地址，0行首地址。

a+1:第1行首地址为: a+1×4×2=2008

a+2:第2行首地址为: a+2×4×2=2016

a+i:第i行首地址为: a+i×4×2

- 数组名可以代表数组首地址，所以a[0]代表0行0列地址，&a[0][0]。a[1]的值是&a[1][0]，a[2]的值是&a[2][0].
- a, a[0], a[1] ,a[2]本身不占内存，不存放数据，仅表示一个地址。
- 列元素地址的表示：
- 用地址法表示列元素的值

行地址

列元素地址

a[0]+0 a[0]+1 a[0]+2 a[0]+3

a+0	2000 1	2002 3	2004 5	2006 7
a+1	2008 9	2010 11	2012 13	2014 15
a+2	2016 17	2018 19	2020 21	2022 23

a[2]+0

列元素地址：a[1]+3 ⇔ *(a+1)+3 ⇔ &a[1][3]

a[1]+3

列元素值：*(a[2]+3) ⇔ *(*(a+2)+3) ⇔ a[2][3]

a[i] ⇔ *(a+i) 等价：第i行第0列的元素地址&a[i][0]

a[i]+j ⇔ *(a+i)+j等价：第i行第j列的元素地址&a[i][j]

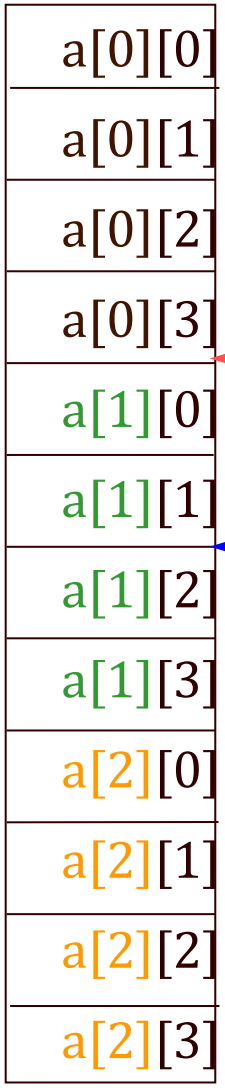
列元素值：*(a[i]+j) ⇔ *(*(a+i)+j) ⇔ a[i][j]

行指针与列指针

int a[3][4];

- 行指针前加*转换为列指针：a→*a，a+1→*(a+1)
- 列指针前加&转换为行指针：a[0]→&a[0]

a+i=&a[i]=a[i]=*(a+i) =&a[i][0]，值相等，含义不同。



地址表示：

(1) a+1 ← 行指针

(2) &a[1][0]

(3) a[1]

(4) *(a+1) ← 列指针

◆a+i ⇔ &a[i],表示第i行首地址，指向行

◆a[i] ⇔ *(a+i) ⇔ &a[i][0]，表示第i行第0列元素地址，指向列

地址表示：

(1) &a[1][2]

(2) a[1]+2

(3) *(a+1)+2

(4) &a[0][0]+1*4+2

二维数组元素内容表示：

（1）a[1][2]

（2）*(a[1]+2)

（3）*(*(a+1)+2)

（4）*(&a[0][0]+1*4+2)

表示形式	含义	地址
a	二维数组名，数组首地址	2000
a[0],*(a+0),*a	第0行第0列元素地址	2000
a+1	第1行首地址	2008
a[1],*(a+1)	第1行第0列元素地址	2008
a[1]+2,*(a+1)+2,&a[1][2]	第1行第2列元素地址	2012
(a[1]+2),(*(a+1)+2),a[1][2]	第1行第2列元素值	13

例11 输出二维数组有关的值

```
#include <stdio.h>
int main()
{ int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  printf("%d,%d\n",a,*a);
  printf("%d,%d\n",a[0],*(a+0));
  printf("%d,%d\n",&a[0],*&a[0][0]);
  printf("%d,%d\n",a[1],a+1);
  printf("%d,%d\n",&a[1][0],*(a+1)+0);
  printf("%d,%d\n",a[2],*(a+2));
  printf("%d,%d\n",&a[2],a+2);
  printf("%d,%d\n",a[1][0],*(*(a+1)+0));
}
```

运行结果：

```
158,158
158,158
158,158
166,166
166,166
174,174
174,174
9,9
```

❖指向多维数组的指针变量

●指向数组元素的指针变量

例12 用指针变量输出数组元素的值

```
#include <stdio.h>
int main()
{ static int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int *p;
  for(p=a[0];p<a[0]+12;p++)
   { if((p-a[0])%4==0)  printf("\n");
       printf("%4d  ",*p);
   }
}
```

运行结果：

```
 1   3   5   7
 9  11  13  15
17  19  21  23
```

p=*a;
p=&a[0][0];
p=*(a+0);
p=a;

❖指向多维数组的指针变量

●指向数组元素的指针变量

例12 用指针变量输出数组元素的值

```
#include <stdio.h>
int main()
{ static int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int *p;
  for(p=a[0];p<a[0]+12;p++)
   { if((p-a[0])%4==0)  printf("\n");
       printf("%4d  ",*p);
   }
}
```

运行结果：

```
 1   3   5   7
 9  11  13  15
17  19  21  23
```

p=*a; ✓

p=&a[0][0]; ✓

p=* (a+0); ✓

p=a; ✗

由此可见：顺序输出数组元素方法简单

而指定输出数组元素则要进行地址的计算

如二维数组为 n × m (n为行, m 为列) 首元素地址为 a[0]

a[i][j]在数组中相对位置的计算公式:

i * m + j (m为每行元素个数)

位移量的计算: a[1][1]=1*4+1=5

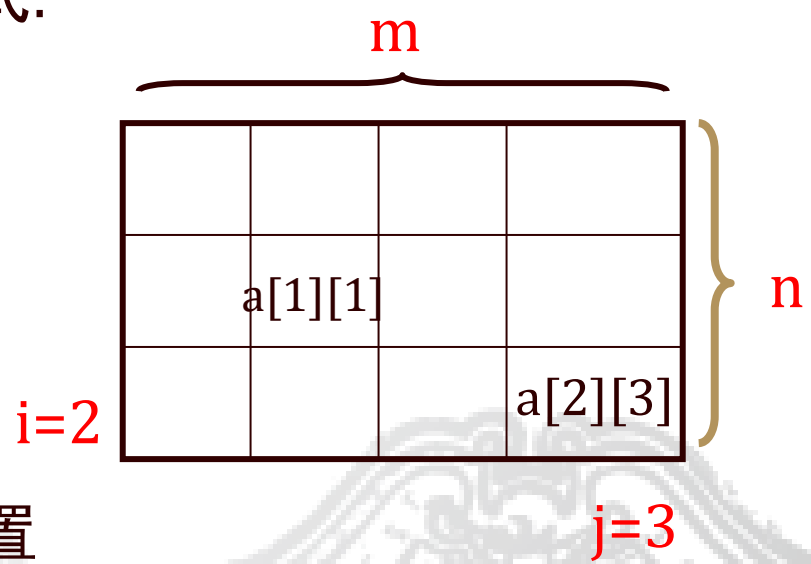
a[2][3]=2*4+3=11

若初值: p=a[0]

则: *(p+1*4+1)=*(p+5) →a[1][1]

*(p+2*4+3)=*(p+11) →a[2][3]

数组下标从0开始便于计算相对位置



- 指向由m个元素组成的一维数组的指针变量
  - 定义形式： 数据类型 (*指针名)[一维数组维数];

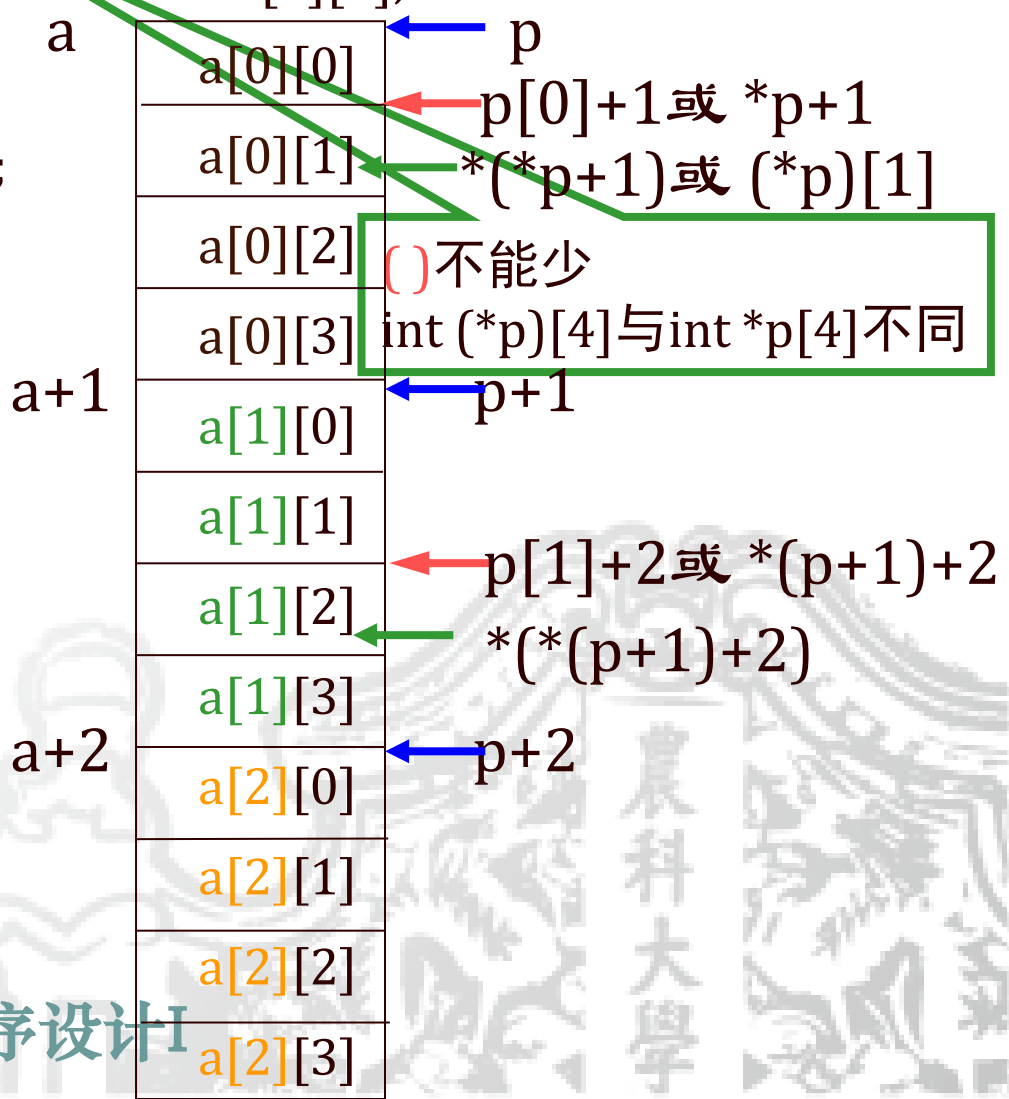
例 int (*p)[4];

  - 可让p指向二维数组某一行

如 int a[3][4], (*p)[4]=a;

p的值是一维数组的首地址，p是行指针

一维数组指针变量维数和二维数组列数必须相同



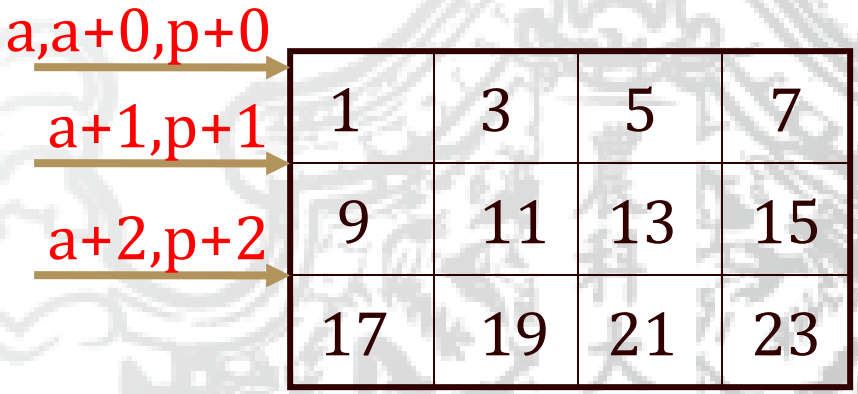
例13 输出二维数组任一行任一列元素的值

```
#include <stdio.h>
int main()
{ static int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
  int  (*p)[4], i, j;
  p=a;
  scanf("i=%d, j=%d", &i, &j);
  printf("a[%d][%d]=%d\ n", i, j, *(*(p+i)+j));
}
```

运行结果：

i=1, j=2↵

a[1][2]=13



❖多维数组的指针作函数参数

- 用指向变量的指针变量
- 用指向一维数组的指针变量
- 用二维数组名

若int a[3][4]; int (*p1)[4]=a; int *p2=a[0];

实参	形参
数组名a	数组名int x[][4]
数组名a	指针变量int (*q)[4]
指针变量p1	数组名int x[][4]
指针变量p1	指针变量int (*q)[4]
指针变量p2	指针变量int *q

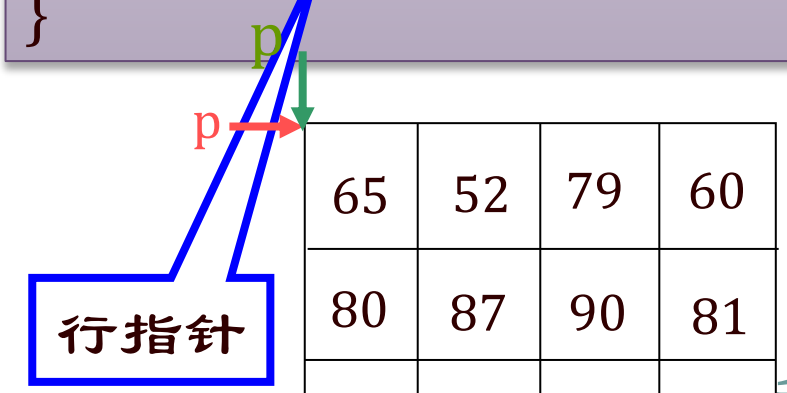
例14 3个学生各学4门课，计算总平均分，输出第n个学生成绩

```
#include <stdio.h>
int main()
{ int average(float  *p,int  n);
  int search(float  (*p)[4],int  n);
  float score[3][4]=
 {{65,67,79,60},{80,87,90,81},
 {90,99,100,98}};
 average(*score,12);
 search(score,2);
}
```

函数说明

列指针

行指针



65	52	79	60
80	87	90	81
90	99	100	98

```
int average(float *p,int n)
{ float  *p_end, sum=0,aver;
  p_end=p+n-1;
  for(;p<=p_end;p++)
    sum=sum+(*p);
  aver=sum/n;
  printf("average=%5.2f\n",aver);
}
int search(float  (*p)[4], int n)
{ int i;
  printf("score of  No.%d  :\n",n);
  for(i=0;i<4;i++)
    printf("%5.2f  ",*(*(p+n)+i));
}
```

float p[][4]

⇔ p[n][i]

例15 3个学生各学4门课，计算总平均分，并查找一门以上课程不及格学生， 输出其各门课成绩

```
int search(float  (*p)[4], int  n)
{ int i,j,flag;
  for(j=0;j<n;j++)
  { flag=0;
    for(i=0;i<4;i++)                    ⇔ p[j][i]
      if(*(*(p+j)+i)<60)  flag=1;
    if(flag==1)
     { printf("No.%d is fail,his scores are:\n",j+1);
       for(i=0;i<4;i++)
            printf("%5.1f ",*(*(p+j)+i));
       printf("\n");
     }
  }
}
```

p →

65	52	79	60
80	87	90	81
90	99	100	98

```
#include <stdio.h>
int main()
{ int search(float  (*p)[4], int n);
  float score[3][4]={{...},{...},{...}};
  search(score,3);  }
```

❖总结：二维数组与一维数组指针变量的关系

如 int a[5][10] 与 int (*p)[10];

- 二维数组名是一个指向有10个元素的一维数组的指针常量
- p=a+i 使 p指向二维数组的第i行
- *(*(p+i)+j) ⇔ a[i][j]
- 二维数组形参实际上是一维数组指针变量，

  即 int x[ ][10] ⇔ int (*x)[10]
- 变量定义(不是形参）时两者不等价
- 系统只给p分配能保存一个指针值的内存区(一般2字节）；而给a分配2*5*10字节的内存区

§ 10.4 字符串与指针

★字符串的表示形式

字符串: 用双引号括起的一串字符。
可赋给字符型的数组或指针变量,
可通过字符型数组名或字符型指针变量输出。

❖用字符数组实现

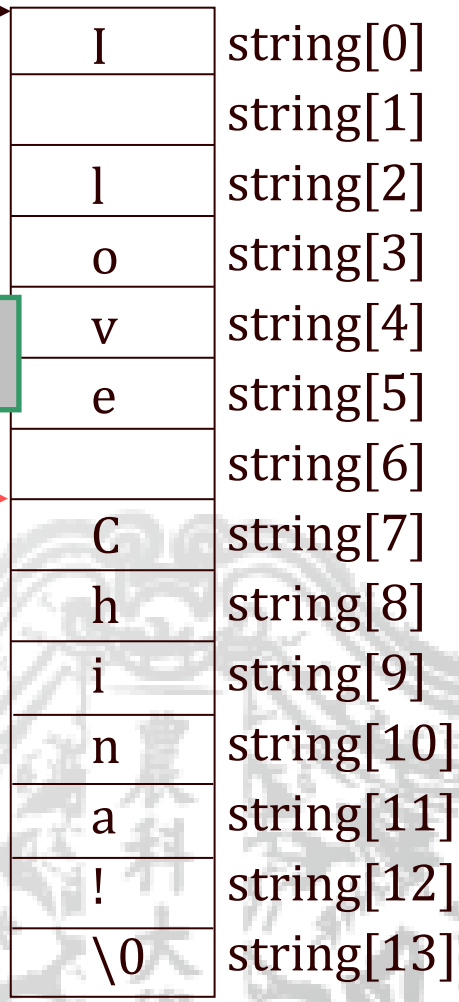
```
例16 定义字符数组
#include <stdio.h>
int main( )
{ char string[]="I love China! ";
  printf("%s\n",string);
  printf("%s\n",string+7);
}
```

数组名，数组首地址

string[4] ⇔ *(string+4)

输出：
I love China!
China!

此句输出？



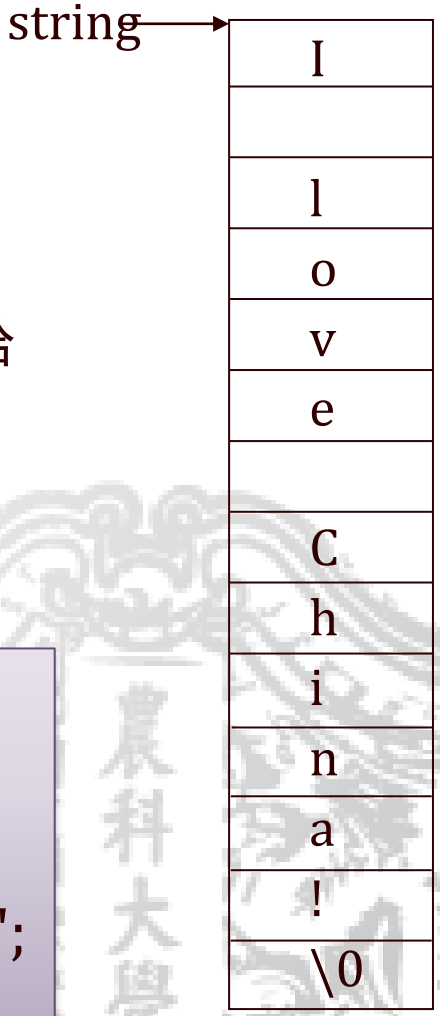
❖用字符指针实现

- 字符串的指针就是字符串的首地址,即第一个字符的地址,可以使用字符指针变量来保存这个地址。
- 使用字符指针可以处理字符串
- 字符指针的定义及使用
  - ◆定义和初始化。

  例： char *string="I love China!";
  - ◆在程序中可以直接把字符串常量赋给一个指针变量。

  例：char *string;

  string="I love China!";

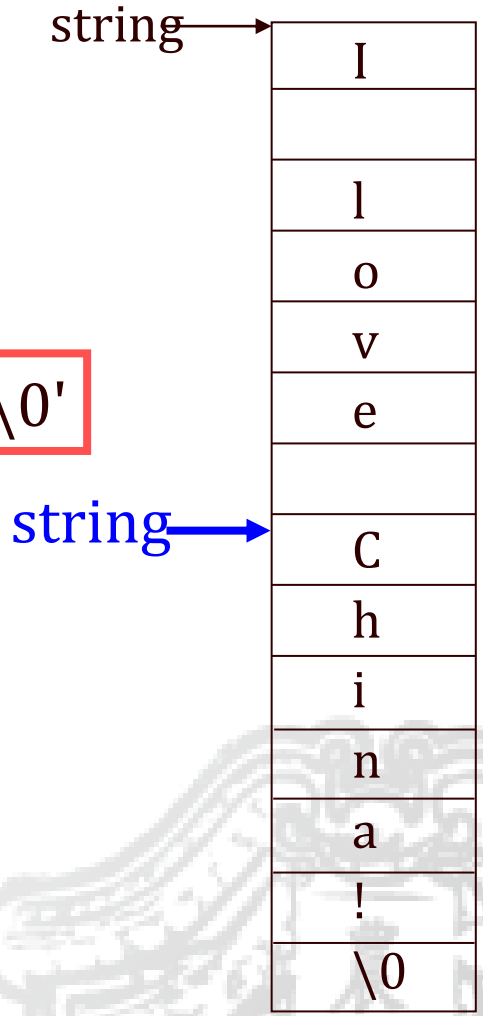



```
例17  定义字符指针
#include <stdio.h>
int main( )
{ char  *string="I  love China! ";
  printf("%s\n",string);}
```

改动后的例17

```
#include <stdio.h>
int main( )
 { char  *string="I  love China! ";
   printf("%s\n",string);
   string+=7;
   while(*string)
    {    putchar(string[0]);
         string++;
    }
 }
```

*string!='\0'



输出：

I love China!

China!

❖用下标法存取字符串中的字符

地址访问:
a[ ]复制到b[ ]

```
例18 将字符串a复制为字符串b
#include <stdio.h>
int main( )
{ char   a[ ]="I am a boy.",b[20];
  int i;
  for(i=0;*(a+i)!='\0';i++)
     *(b+i)=*(a+i);
  *(b+i)='\0';
  printf("string a is: %s\n",a);
  printf("string b is: ");
  for(i=0;b[i]!='\0';i++)
     printf("%c",b[i]);
  printf("\n");
}
```

*(a+i) = a[i]

⇔ b[i]=a[i]

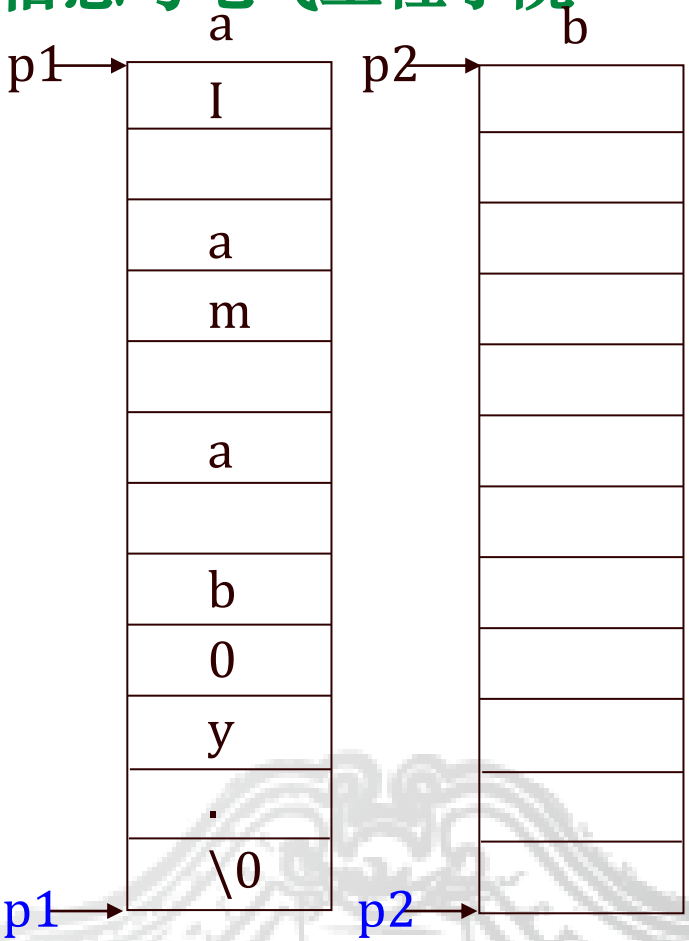
下标法输出

运行结果:
string a is: I am boy.
string b is: I am boy.

例19 用指针变量实现例18

```
#include <stdio.h>
int main( )
{ char   a[ ]="I am a boy. ",b[20];
  char   *p1=a,*p2=b;
  int i;
  for(   ; *p1!='\0' ; p1++,p2++)
     {*p2=*p1;}
  *p2='\0';
  printf("string a is: %s\n", a  );
  printf("string b is: %s\n", b  );
}
```

p1=a; p2=b;



运行结果：

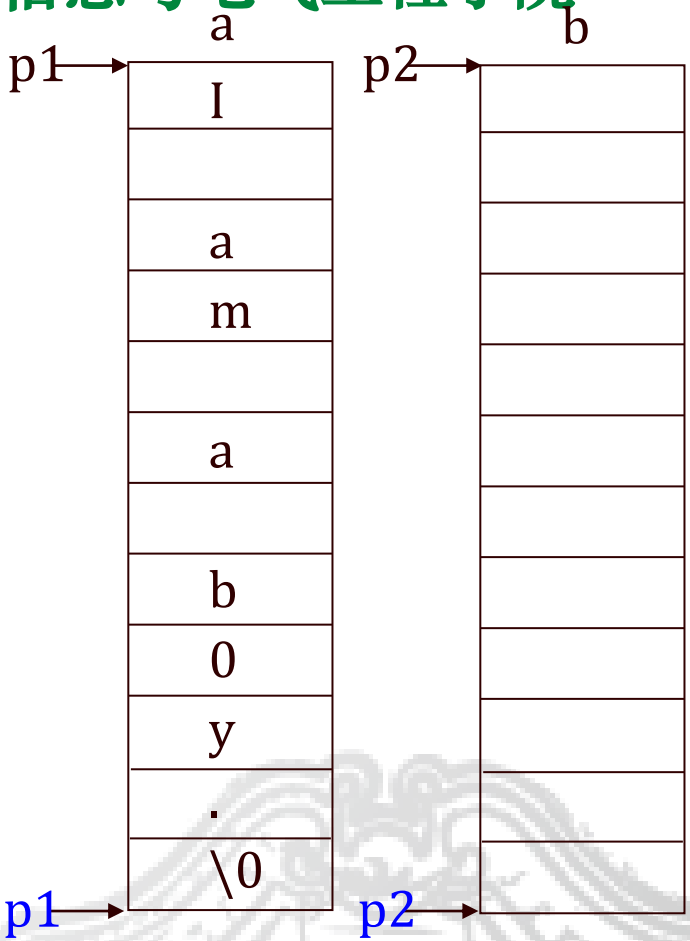
string a is:

string b is:

例19 用指针变量实现例18

```
#include <stdio.h>
int main( )
{ char   a[ ]="I am boy. ",b[20];
  char   *p1=a,*p2=b;
  int i;
  for(   ; *p1!='\0' ; p1++,p2++)
     {*p2=*p1;}
  *p2='\0';
  printf("string a is: %s\n", p1
  printf("string b is: %s\n", p2
}
```

p1=a; p2=b;



运行结果：

string a is:

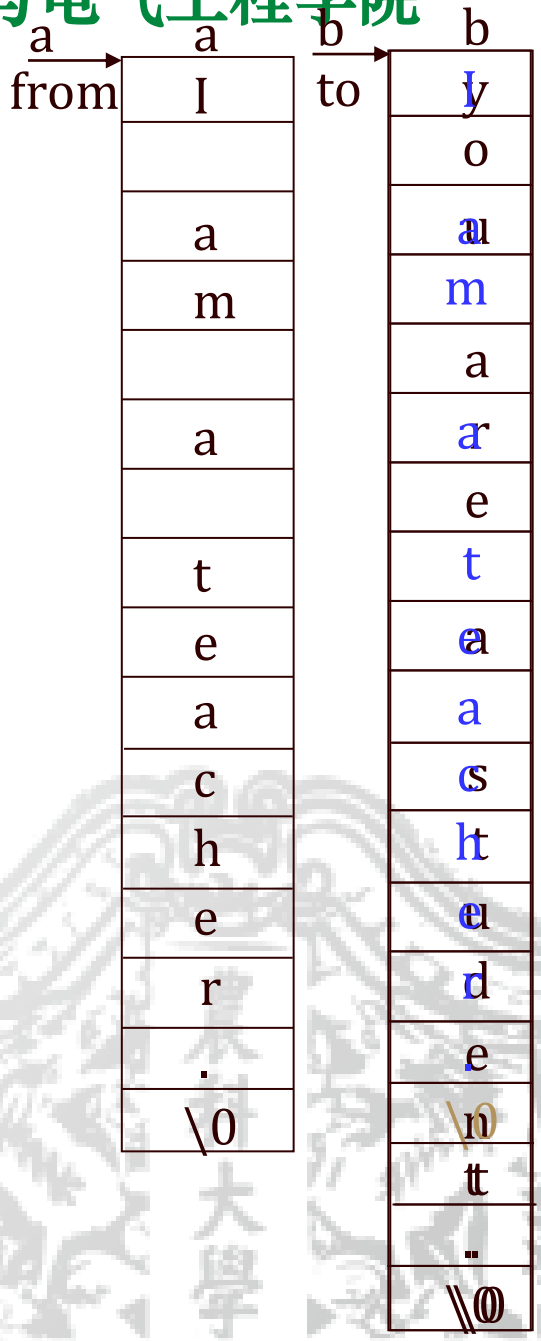
string b is:

★字符指针作函数参数

例20 用函数调用实现字符串的复制

(1)用字符数组作参数

```
#include <stdio.h>
int main()
{int copy_string(char  from[],char to[]);
  char a[ ]="I am a teacher.";
  char b[ ]="You are a student.";
  printf("string_a=%s\n string_b=%s\n",a,b);
  printf("copy string_a to string_b: \n");
  copy_string(a,b);   /* 数组名作参数是地址传递*/
  printf("\nstring_a=%s\nstring_b=%s\n",a,b);
}
int copy_string(char  from[],char to[])
{ int i=0;
  while(from[i]!='\0')
    { to[i]=from[i];  i++;  }
  to[i]='\0';
}
```



⑵字符指针变量作形参

```
#include <stdio.h>
int main()
{ int copy_string(char *from,char  *to);
  char *a="I am a teacher.";
  char *b="You are a student.";
  printf("string_a=%s\n string_b=%s\n",a,b);
  printf("copy string_a to string_b: \n");
  copy_string(a,b);
 printf("\nstring_a=%s\nstring_b=%s\n",a,b);
}
int copy_string(char *from,char  *to)
{  for(;*from!='\0';from++,to++)
      {*to=*from;}
    *to='\0';
}
```

运行结果：

```
string_a  is: I am a teacher.
string _b is: You are a student.
copy string a to string b:
string_a  is: I am a teacher.
string _b is: I am a teacher.
```

对使用字符指针变量和字符数组的讨论

char *cp; 与 char str[20]; 的区别

- str由若干元素组成，每个元素放一个字符；而cp中存放字符串首地址
- 赋值方式：
  - 字符数组只能对元素赋值。 char str[20];
    str="I love China! "; (×)
  - 字符指针变量可以用： char *cp;
    cp="I love China! "; (✓)
  - 赋初值：

    char *cp="China!"; 等价 char *cp; cp="China! ";

    char str[14]={"China"};不等价char str[14]; str[ ]="China" (×)

❖字符指针变量接受键入字符串时,必须先开辟存储空间（赋地址）。

例 char str[10];
scanf("%s",str); (✓)
而 char *cp;
scanf("%s", cp); (×)

```
char *cp , str[10];
cp=str;
scanf("%s",cp);    (✓)
```

❖数组名str代表的是地址常量不能改变；
指针变量cp是地址变量，值可以改变。

例21 (✓)

```
#include <stdio.h>
int main( )
{ char *a="I love China!";
  a=a+7;
  printf("%s",a);}
```

改为： (×)

```
#include <stdio.h>
int main( )
{ char str[ ]={"I love China!"};
  str=str+7;
  printf("%s",str);}
```

❖字符指针变量接受键入字符串时,必须先开辟存储空间（赋地址）。

例22 用下标法引用指针变量所指的字符串中的字符

```
#include <stdio.h>
int main( )
{ char *a="I love China!"; int i;
 printf("The sixth charcter is %c\n",a[5]);
 for(i=0;a[i]!='\0';i++)
   printf("%c",a[i]);
 printf("\n");}
```

例21 (✓)

```
#include <stdio.h>
int main( )
{ char *a="I love China!";
 a=a+7;
 printf("%s",a);}
```

改为： (×)

```
#include <stdio.h>
int main( )
{ char str[ ]={"I love China!"};
 str=str+7;
 printf("%s",str);}
```

程序

- 用指针变量指向的格式字符串代替printf中的格式字符串（可变格式输出函数）

```
char *format;
format="a=%d,b=%f\n";
printf(format,a,b);
```

相对于：

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

可以用字符数组实现：

```
char format[ ]="a=%d,b=%f\n";
printf(format,a,b);
```

❖判断和修正：

char str[]={"Hello! "}; (√)
char str[]="Hello! "; (√)
char str[]={'H', 'e', 'l', 'l', 'o', '! '}; (√)
char *cp="Hello"; (√)
int a[]={1,2,3,4,5}; (√)
int *p={1,2,3,4,5}; (×) ⟹ int *p=&a;

char str[10],*cp;
int a[10],*p;
str="Hello"; (×) ⟹ char str[10]="Hello";
cp="Hello! "; (√)
a={1,2,3,4,5}; (×) ⟹ a[10]={1,2,3,4,5};
p={1,2,3,4,5}; (×) ⟹ p=&a;

§ 10.5 指向函数的指针

★用函数指针变量调用函数。

❖函数指针：函数在编译时被分配的入口地址,用函数名表示。函数指针指向的是程序代码存储区。

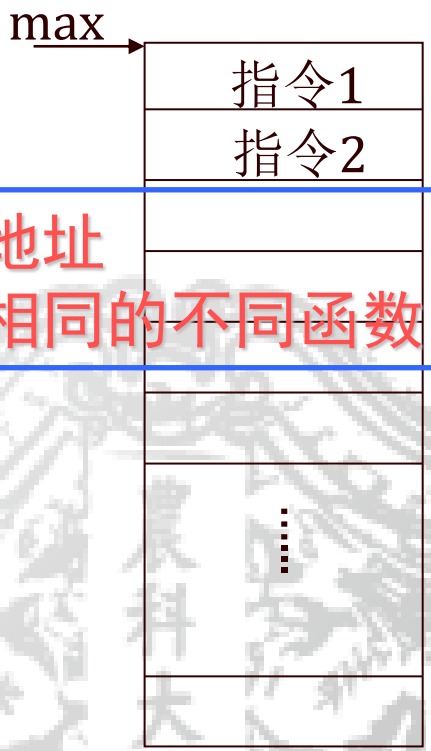
❖函数指针变量定义形式：

数据类型 (*指针变量名)( );

如 int (*p)();

函数返回值的数据类型

专门存放函数入口地址
可指向返回值类型相同的不同函数



( )不能省
int (*p)() 与 int *p() 不同

程序设计

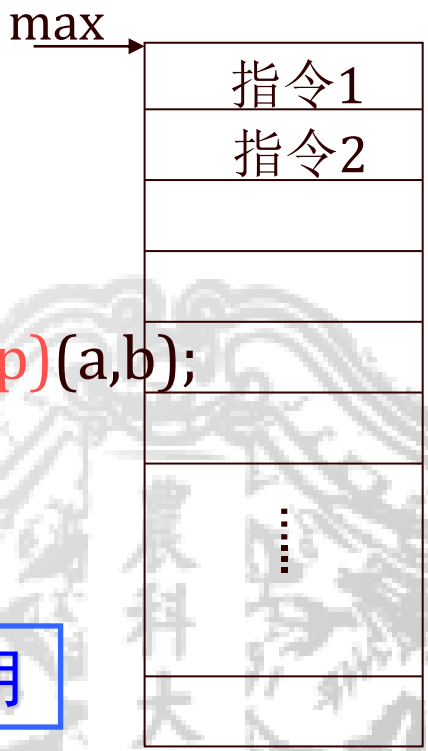
§ 10.5 指向函数的指针

★用函数指针变量调用函数。

- ❖函数指针：函数在编译时被分配的入口地址,用函数名表示。函数指针指向的是程序代码存储区。
- ❖函数指针变量定义形式：

  数据类型 (*指针变量名)( );

  如 int (*p)();
- ❖函数指针变量赋值：如 p=max;
- ❖函数调用形式： c=max(a,b); ⇔ c=(*p)(a,b);
- ❖对函数指针变量p±n, p++, p--无意义



函数指针变量指向的函数必须有函数说明

例23　求a和b中的大者

⑴一般方法：

```
#include <stdio.h>
int main()
{ int max(int ,int);
  int a,b,c;
  scanf("%d,%d",&a,&b);
  c=max(a,b);
  printf("a=%d,b=%d,max=%d\n",a,b,c);
}
int  max(int x,int y)
{ int z;
  if(x>y)  z=x;
  else  z=y;
  return(z);}
```

例23　求a和b中的大者

⑵通过指针变量访问函数

```
#include <stdio.h>
int main()
{ int max(int , int);
 int (*p)(int , int);
  int a,b,c;
  p=max;
  scanf("%d,%d",&a,&b);
  c=(*p)(a,b);
  printf("a=%d,b=%d,max=%d\n",a,b,c);
}
int  max(int x,int y)
{ int z;
  if(x>y)  z=x;
  else     z=y;
  return(z);}
```

★用指向函数的指针作函数参数（了解）

函数指针变量通常用途是将指针作为参数传递到其他函数，实现对不同函数的调用。

例24 用函数指针变量作参数，求最大值、最小值和两数之和

```
#include <stdio.h>
int main()
{int max(int,int),min(int,int),add(int,int);
 void process(int,int,int (*fun)(int,int));
 int a,b;
 printf("enter a and b:");
 scanf("%d,%d",&a,&b);
 process(a,b,max);
 process(a,b,min);
 process(a,b,add);
}
void process(int x,int y,int (*fun)(int,int))
{ int result;
 result=(*fun)(x,y);
 printf("%d\n",result);
}
```

```
int max(int x,int y)
{  printf("max=");
    return(x>y?x:y);
}
int min(int x,int y)
{   printf("min=");
    return(x<y?x:y);
}
int add(int x,int y)
{  printf("sum=");
    return(x+y);
}
```

§ 10.6 返回指针值的函数

★函数定义形式：

类型标识符 *函数名(参数表);

例 int *f(int x, int y)

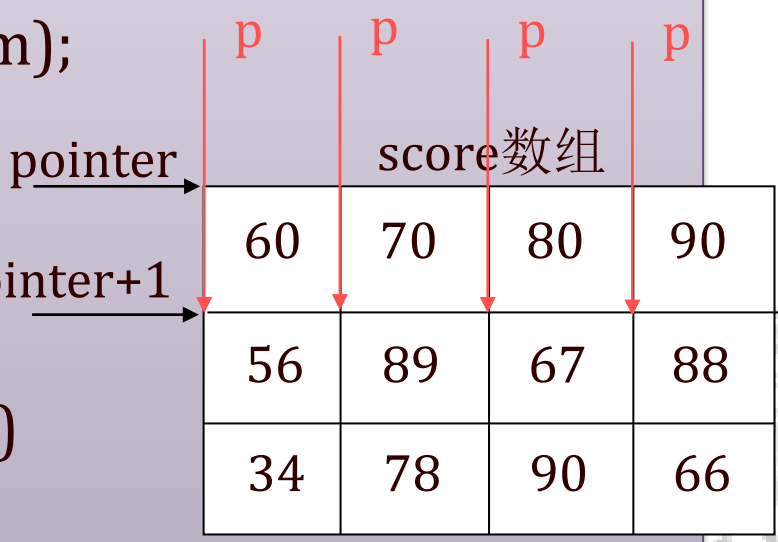
例25

```
#include <stdio.h>
int main()
{ float score[ ][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
  float  *search(float (*pointer)[4],int n);
  float *p;
  int i,m;
  printf("Enter the number of student: ");
  scanf("%d",&m);
  printf("The scores of No.%d are:\n",m);
  p=search(score,m);
  for(i=0;i<4;i++)
     printf("%5.2f\t",*(p+i));
  printf("\n");}
float *search(float (*pointer)[4], int  n)
{ float *pt;
  pt=*(pointer+n);
  return(pt);}
```

接受search函数返回的指向某学生0门课程地址的指针变量

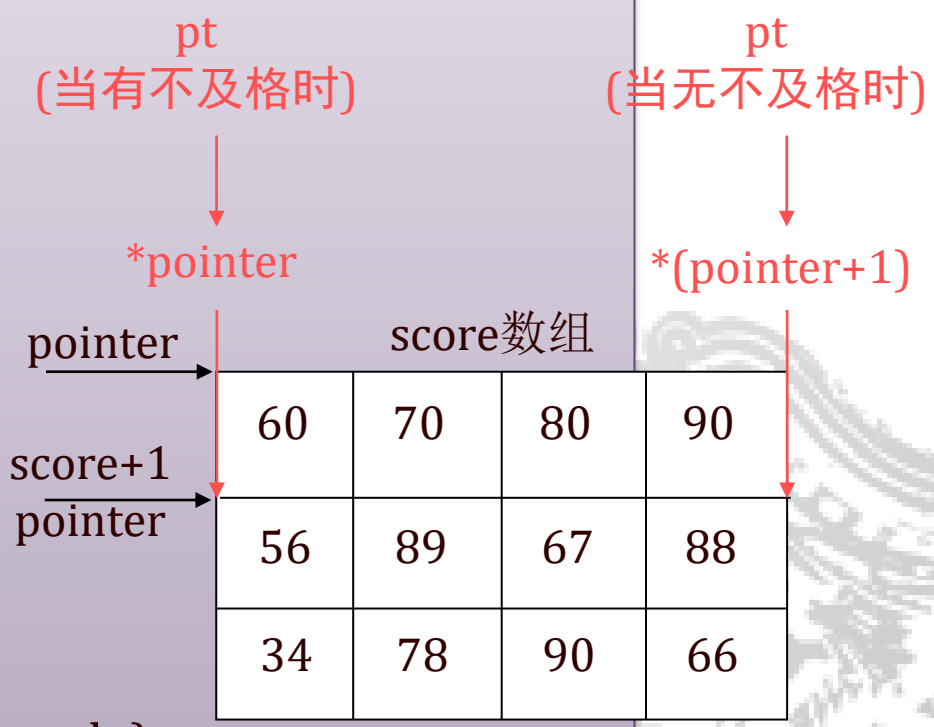
指向实型变量的指针变量

形参指向一维数组的指针变量



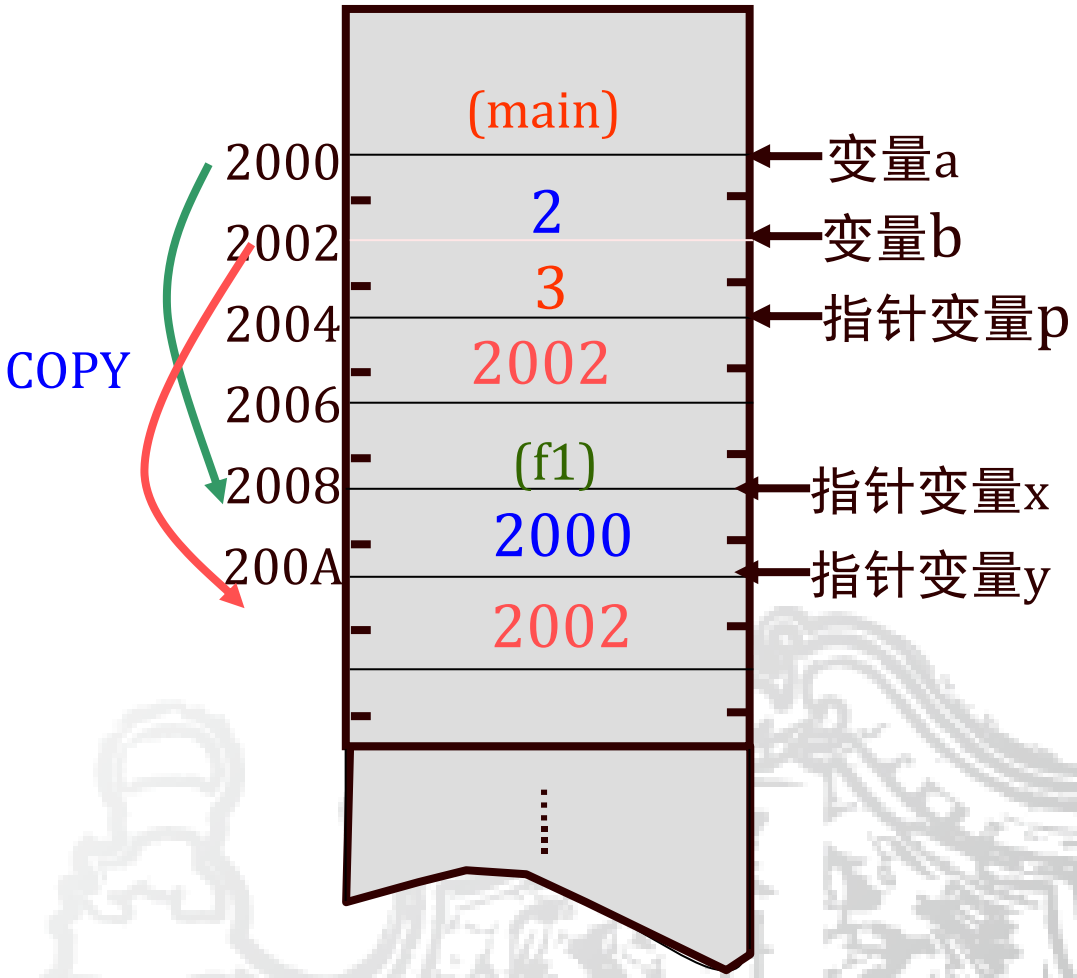
例26　找出例25中有不及格课程的学生及其学号

```
#include <stdio.h>
int main()
{ float score[ ][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
  float  *search(float (*pointer)[4]);
  float *p;    int i,j;
  for(i=0;i<3;i++)
     { p=search(score+i);
       if(p==*(score+i));
          { printf("No. %d scores: ",i);
            for(j=0;j<4;j++)
               printf("%5.2f\t",*(p+j));}
}   }
float *search(float (*pointer)[4])
{ int i;
  float *pt;
  pt=*(pointer+1);
  for(i=0;i<4;i++)
    if(*(*pointer+i)<60 {pt=*pointer;break;}
  return(pt); }
```



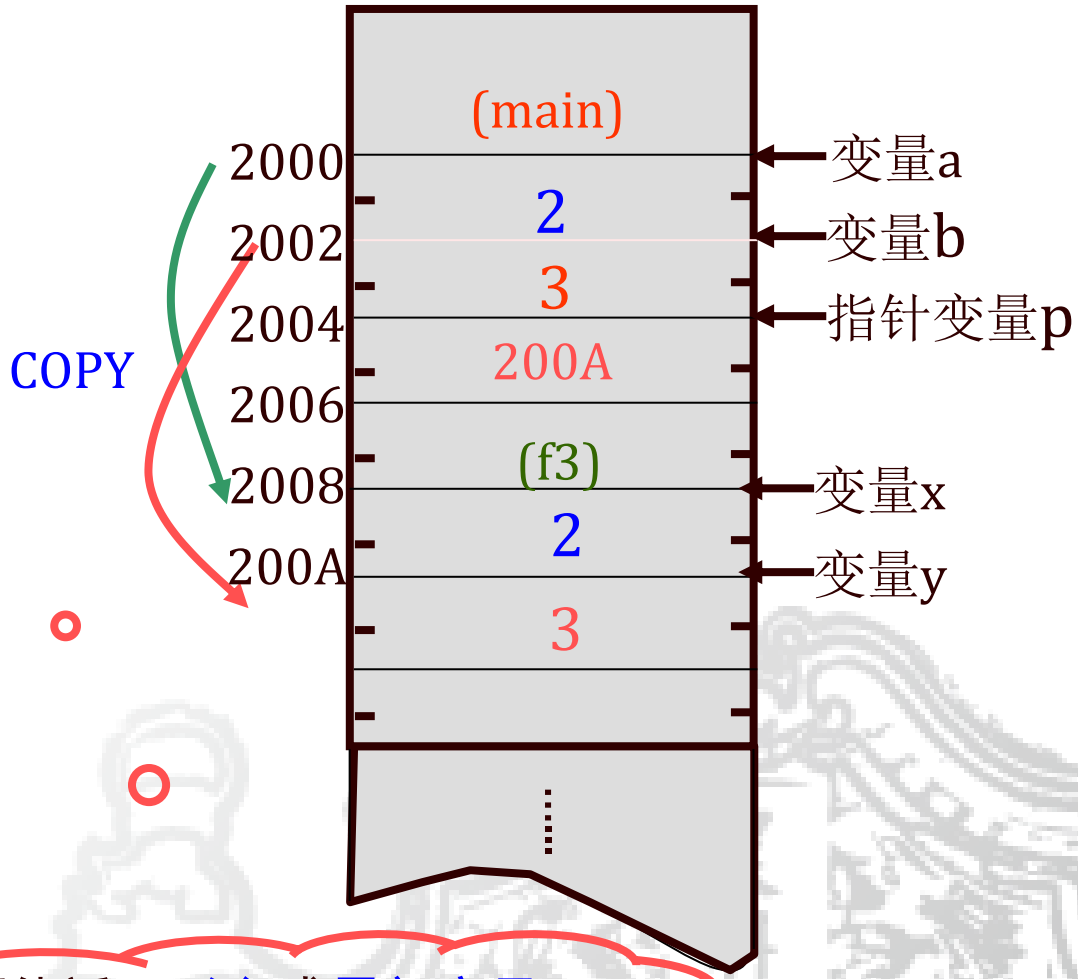
例27 写一个函数，求两个int型变量中居于较大值的变量的地址

```
main()
{ int a=2,b=3;
  int *p;
  p=f1(&a,& b);
  printf("%d\n",*p);
}
int *f1(int x,int y)
{ if(x>y)
    return  x;
  else
    return  y;
}
```



例27 写一个函数，求两个int型变量中居于较大值的变量的地址

```
main()
{ int a=2,b=3;
  int *p;
  p=f3(a,b);
  printf("%d\n",*p);
}
int *f3(int x,int y)
{ if(x>y)
    return  &x;
  else
    return  &y;
}
```



不能返回形参或局部变量的地址作函数返回值

§10.7 指针数组和指向指针的指针

用于处理二维数组或多个字符串

★指针数组的概念

- ❖定义：数组中的元素均为指针变量
- ❖定义形式：[存储类型] 类型名 *数组名[数组长度]；

例 int *p[4];

指针所指向变量的数据类型

区分int *p[4]与int (*p)[4]

指针本身的存储类型

§10.7 指针数组和指向指针的指针

用于处理二维数组或多个字符串

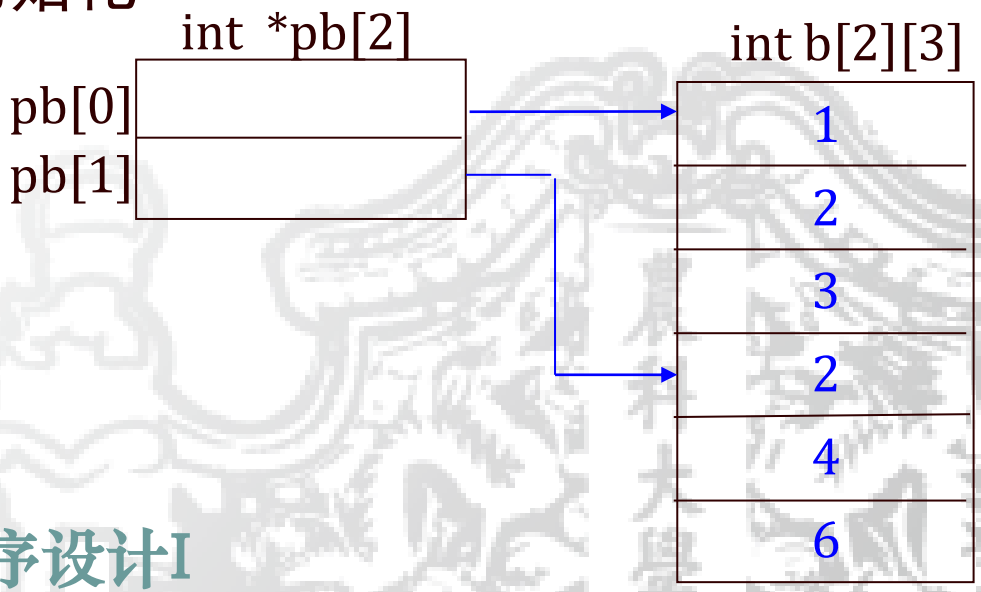
★指针数组的概念

- ❖定义：数组中的元素均为指针变量
- ❖定义形式：[存储类型] 类型名 *数组名[数组长度]；

例 int *p[4];

- ❖指针数组赋值与初始化

```
赋值:
main()
{ int b[2][3],*pb[2];
  pb[0]=b[0];
  pb[1]=b[1];
        ┆
}
```



§10.7 指针数组和指向指针的指针

用于处理二维数组或多个字符串

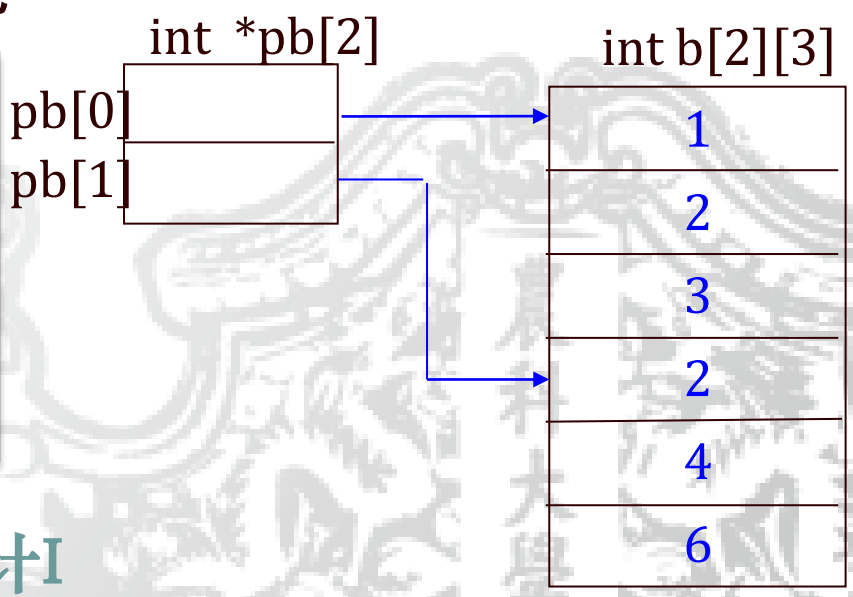
★指针数组的概念

- ❖定义：数组中的元素均为指针变量
- ❖定义形式：[存储类型] 类型名 *数组名[数组长度]；

例 int *p[4];

- ❖指针数组赋值与初始化

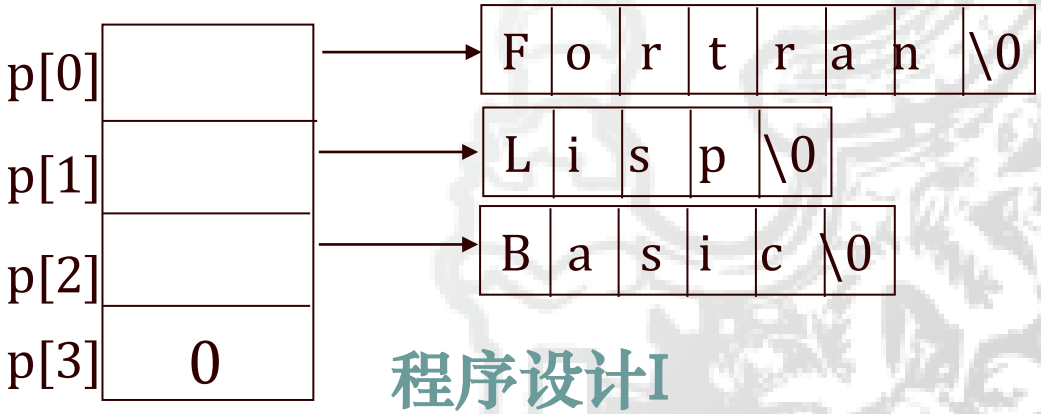
```
初始化:
main()
{   int b[2][3],*pb[ ]={b[0],b[1]};
    ……..
}
```



❖指针数组赋值与初始化

```
赋值:
main()
{ char a[]="Fortran";
  char b[]="Lisp";
  char c[]="Basic";
  char *p[4];
  p[0]=a; p[1]=b; p[2]=c; p[3]=NULL;
     ……..
}
```

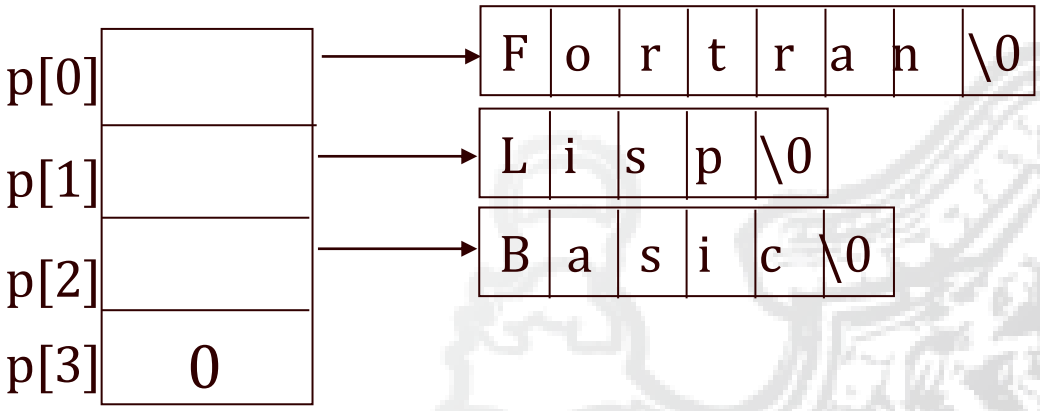
```
或:
main()
{ char *p[4];
  p[0]= "Fortran";
  p[1]= "Lisp";
  p[2]= "Basic";
  p[3]=NULL;
     ……..
}
```



❖指针数组赋值与初始化

初始化:

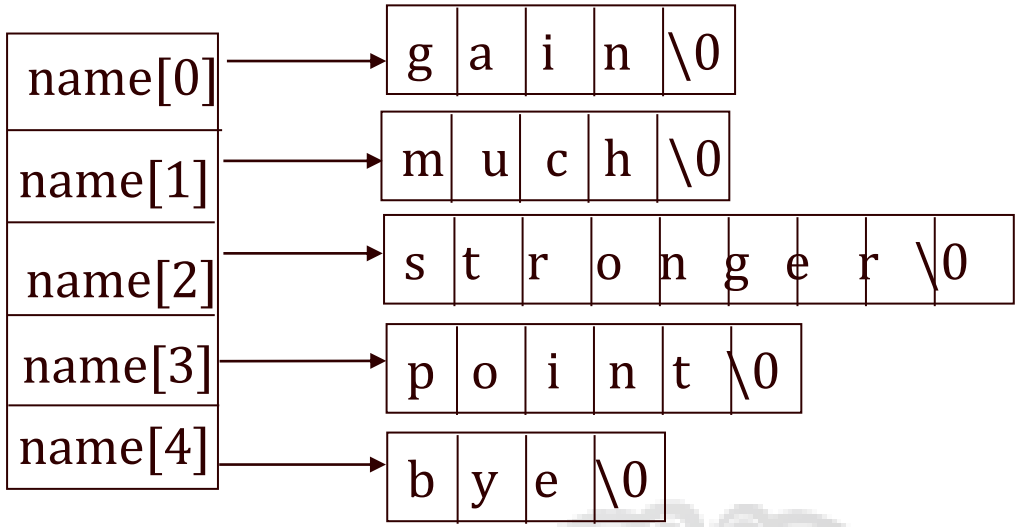
```
main()
{ char *p[]={"Fortran", "Lisp", "Basic",NULL};
    ……..
}
```



❖二维数组与指针数组区别：

```
char  name[5][9]={"gain", "much", "stronger", "point", "bye"};
```

g	a	i	n	\0				
m	u	c	h	\0				
s	t	r	o	n	g	e	r	\0
p	o	i	n	t	\0			
b	y	e	\0					




```
char *name[5]={"gain", "much", "stronger", "point", "bye"};
```

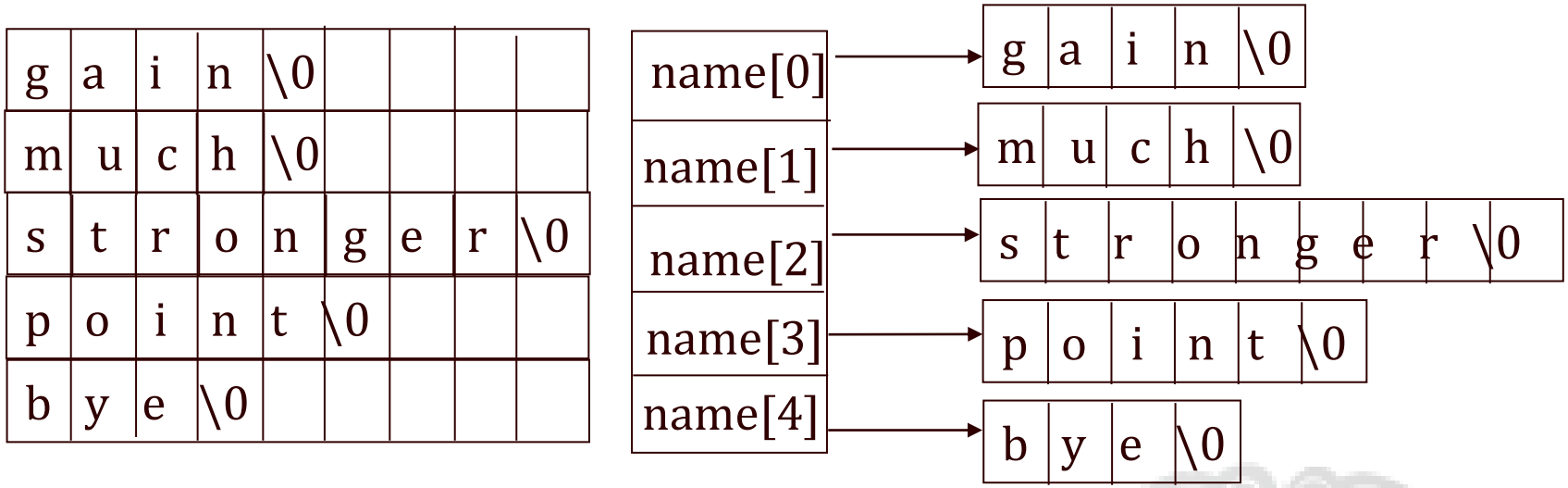
二维数组存储空间固定

字符指针数组相当于可变列长的二维数组

分配内存单元=数组维数*2+各字符串长度

❖二维数组与指针数组区别：

char name[5][9]={"gain", "much", "stronger", "point", "bye"};



char *name[5]={"gain", "much", "stronger", "point", "bye"};

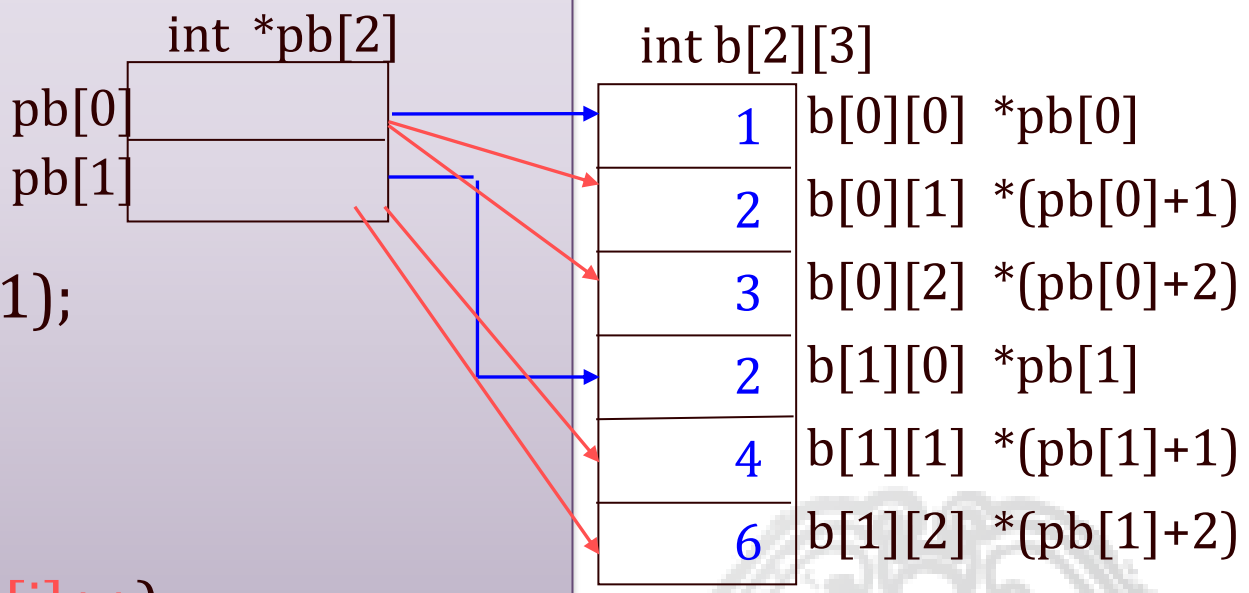
指针数组元素的作用相当于二维数组的行名

但指针数组中元素是指针变量

二维数组的行名是地址常量

例28　用指针数组处理二维数组

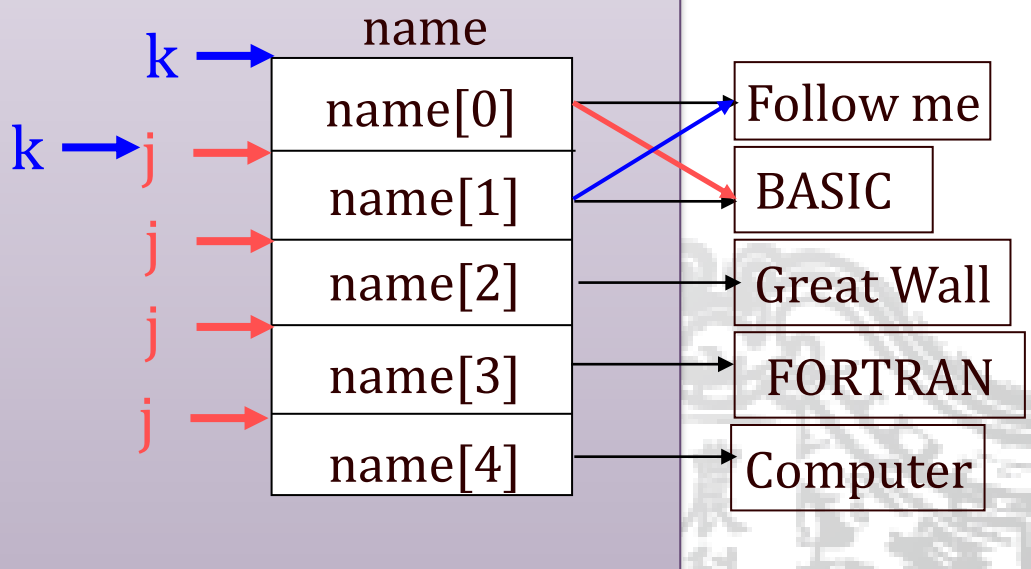
```
main()
{ int b[2][3],*pb[2];
  int i,j;
  for(i=0;i<2;i++)
    for(j=0;j<3;j++)
      b[i][j]=(i+1)*(j+1);
  pb[0]=b[0];
  pb[1]=b[1];
  for(i=0;i<2;i++)
    for(j=0;j<3;j++,pb[i]++)
      printf("b[%d][%d]:%2d\n",i,j,*pb[i]);
}
```



例29 对字符串排序（简单选择排序）

```
#include <stdio.h>
#include <string.h>
int main()
{ int sort(char  *name[ ],int n);
  int print (char  *name[ ],int n);
  char *name[]={"Follow me","BASIC",
     "Great Wall","FORTRAN","Computer design"};
  int n=5;
  sort(name,n);
  print(name,n);
}
int sort(char *name[ ],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++) if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)  { temp=name[i];  name[i]=name[k]; name[k]=temp;}
} }
```

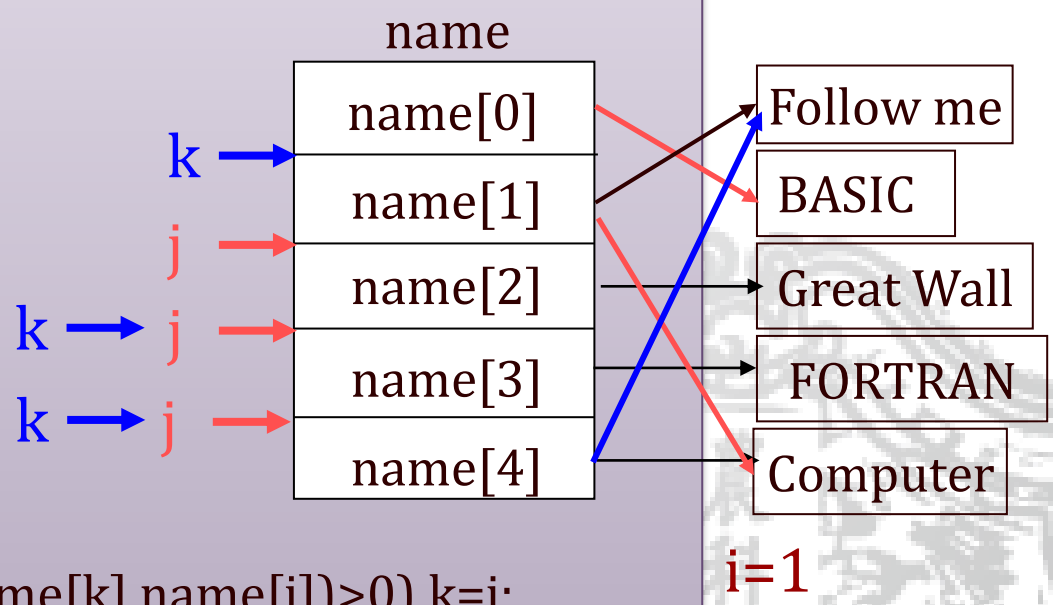
```
int print(char *name[ ], int n)
{ int i ;
  for(i=0; i<n; i++)
    printf("%s\n",name[ i ];
}
```



例29 对字符串排序（简单选择排序）

```
#include <stdio.h>
#include <string.h>
int main()
{ int sort(char  *name[ ],int n);
  int print (char  *name[ ],int n);
  char *name[]={"Follow me","BASIC",
     "Great Wall","FORTRAN","Computer design"};
  int n=5;
  sort(name,n);
  print(name,n);
}
int sort(char *name[ ],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++) if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)  { temp=name[i];  name[i]=name[k]; name[k]=temp;}
} }
```

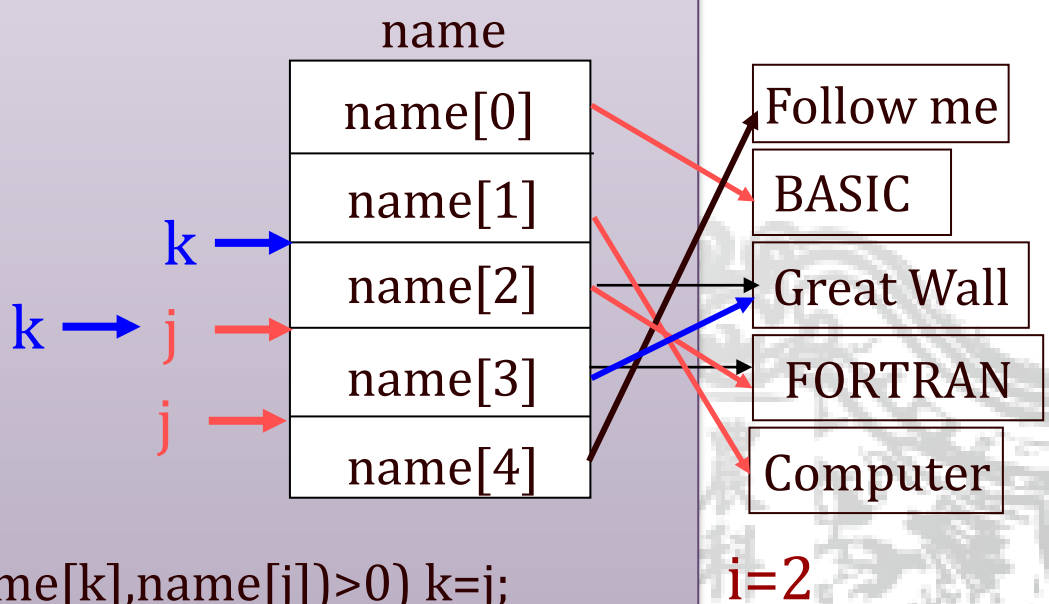
```
int print(char *name[ ], int n)
{ int i ;
  for(i=0; i<n; i++)
    printf("%s\n",name[ i ];
}
```



例29 对字符串排序（简单选择排序）

```
#include <stdio.h>
#include <string.h>
int main()
{ int sort(char  *name[ ],int n);
  int print (char  *name[ ],int n);
  char *name[]={"Follow me","BASIC",
     "Great Wall","FORTRAN","Computer design"};
  int n=5;
  sort(name,n);
  print(name,n);
}
int sort(char *name[ ],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++) if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)  { temp=name[i];  name[i]=name[k]; name[k]=temp;}
} }
```

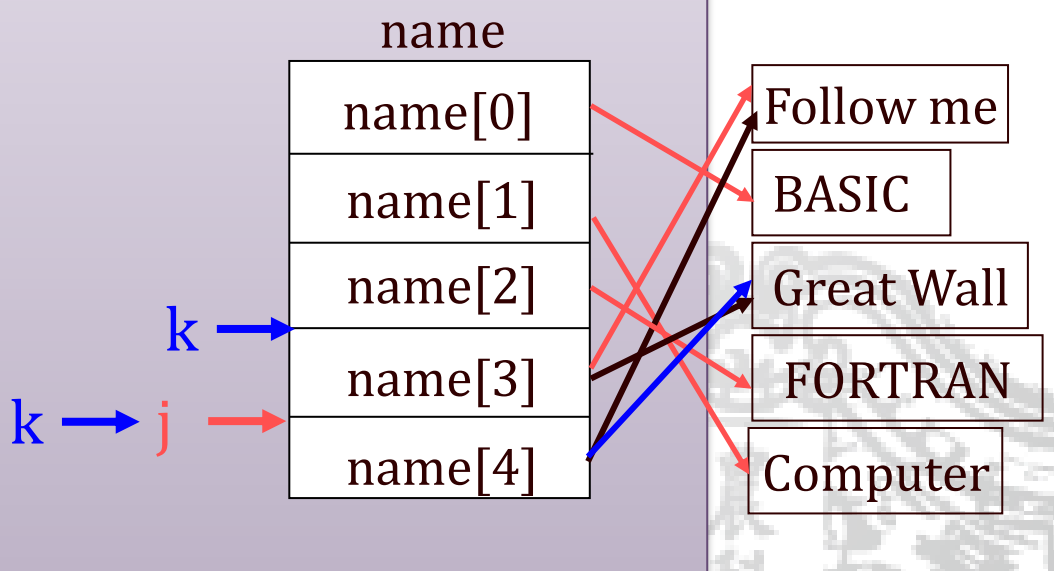
```
int print(char *name[ ], int n)
{ int i ;
  for(i=0; i<n; i++)
    printf("%s\n",name[ i ];
}
```



例29 对字符串排序（简单选择排序）

```
#include <stdio.h>
#include <string.h>
int main()
{ int sort(char  *name[ ],int n);
  int print (char  *name[ ],int n);
  char *name[]={"Follow me","BASIC",
     "Great Wall","FORTRAN","Computer design"};
  int n=5;
  sort(name,n);
  print(name,n);
}
int sort(char *name[ ],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++) if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)  { temp=name[i];  name[i]=name[k]; name[k]=temp;}
} }
```

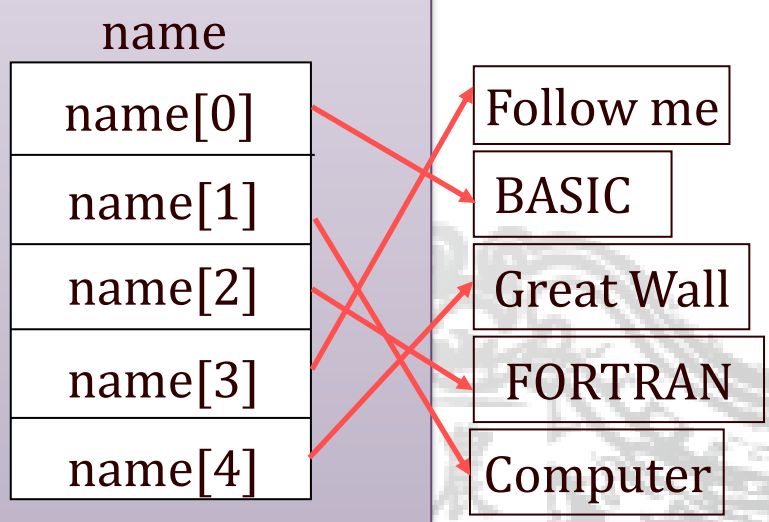
```
int print(char *name[ ], int n)
{ int i ;
  for(i=0; i<n; i++)
    printf("%s\n",name[ i ];
}
```



例29 对字符串排序（简单选择排序）

```
#include <stdio.h>
#include <string.h>
int main()
{ int sort(char  *name[ ],int n);
  int print (char  *name[ ],int n);
  char *name[]={"Follow me","BASIC",
     "Great Wall","FORTRAN","Computer design"};
  int n=5;
  sort(name,n);
  print(name,n);
}
int sort(char *name[ ],int n)
{ char *temp;
  int i,j,k;
  for(i=0;i<n-1;i++)
  { k=i;
    for(j=i+1;j<n;j++) if(strcmp(name[k],name[j])>0) k=j;
    if(k!=i)  {temp=name[i];  name[i]=name[k]; name[k]=temp;}
} }
```

```
int print(char *name[ ], int n)
{ int i ;
  for(i=0; i<n; i++)
    printf("%s\n",name[ i ];
}
```

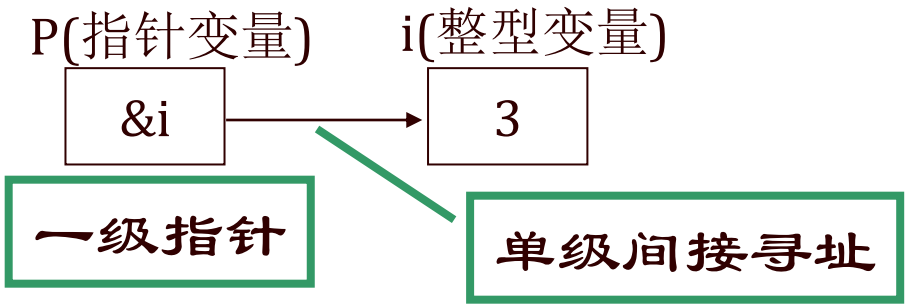


< >

★指向指针的指针—多级指针（了解）

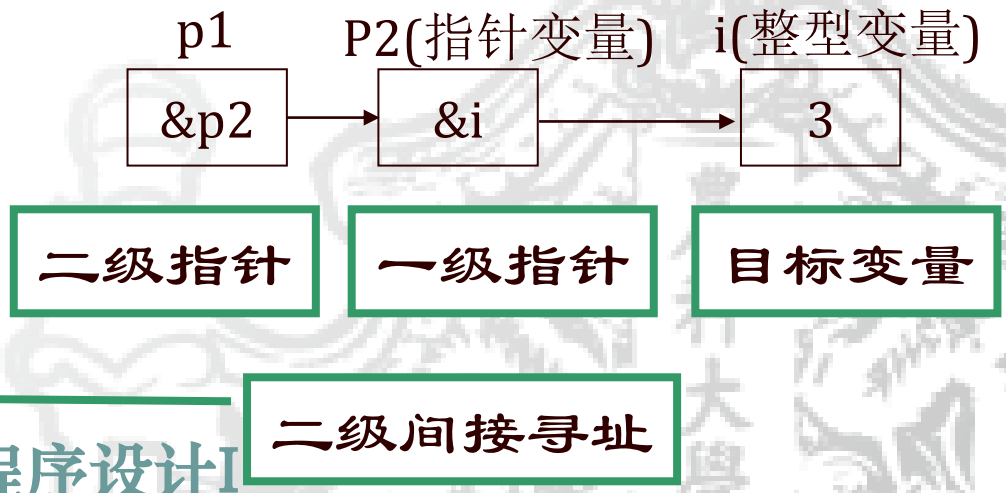
❖一级指针:指针变量中存放目标变量的地址

```
例 int  *p;
   int i=3;
   p=&i;
   *p=5;
```



❖二级指针:指针变量中存放一级指针变量的地址

```
例 int  **p1;
   int  *p2;
   int i=3;
   p2=&i;
   p1=&p2;
   **p1=5;
```



指针本身的存储类型

❖定义形式：[存储类型] 数据类型 **指针名；

如：char **p

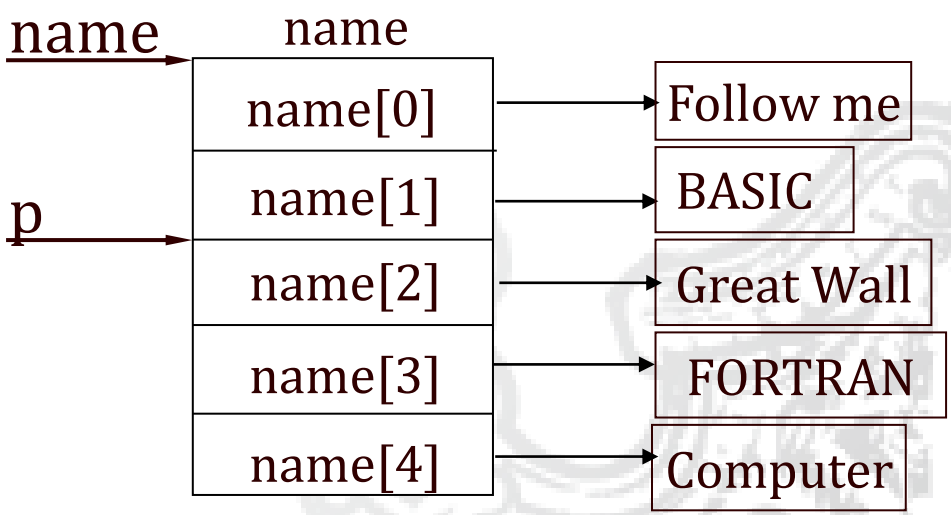
如果：char *name[ ]={"Follow me".....};

char **p;

p=name+2;

最终目标变量的数据类型

则：printf("%s\n",*p); 输出：Great Wall



例30 用指向指针的指针处理字符串

```
#include <stdio.h>
int main( )
{ char *name[ ]={"Follow me","BASIC",
     "Great Wall","FORTRAN","Computer design"};
  char **p;
  int i;
  for(i=0;i<5;i++)
    { p=name+i;
      printf("%s\n",*p);
    }
}
```

*p是name[i]的值，即第i个字符串的起始地址

运行结果：

```
Follow me
BASIC
FORTRAN
Great Wall
Computer design
```

例30 用指向指针的指针处理整型数据

```
#include <stdio.h>
int main( )
{ int a[5]={1,3,5,7,9};
  int *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};
  int **p, i;
  p=num;
  for(i=0;i<5;i++)
     { printf("%d  ",**p);
       p++;
     }
  printf("\n");
}
```

*p是p间接指向对象的地址
**p是p间接指向对象的值

运行结果：
1 3 5 7 9

❖二级指针与指针数组的关系

int **p 与 int *q[10]

- 指针数组名是二级指针常量
- p=q; p+i 是q[i]的地址
- 指针数组作形参，int *q[ ]与int **q完全等价；但作为变量定义两者不同
- 系统只给p分配能保存一个指针值的内存区；而给q分配10块内存区，每块可保存一个指针值

★指针数组作main函数的形参（掌握）

- ❖main函数的一般调用形式： main( )
- ❖main函数的有参数调用形式：

  main(int argc , char *argv[ ])

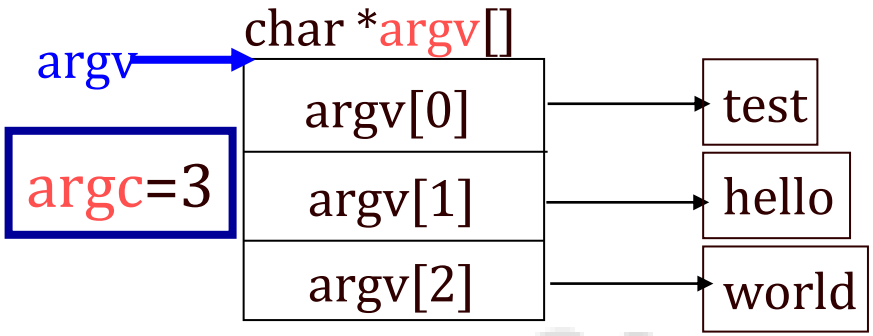
- ❖其中(int argc , char *argv[ ] 是形参，可以多个：
  - int argc;　　命令名和所有参数个数之和
  - char *argv[ ];　各元素是指针，分别指向各参数字符串
- ❖在实际运行程序时，实参和命令名（C程序编译和连接后得到的可执行文件名）以命令行形式给出：

  命令名 参数1 参数2 ..... 参数n [回车]

例31 用命令行参数方式运行程序

```
/*test.c*/
main(int argc, char *argv[])
{ while(argc>1)
   { ++argv;
     printf("%s\n",*argv);
     --argc;
   }
}
```

```
main(int argc, char *argv[])
{  while(argc-->0)
     printf("%s\n",*argv++);
}
```



1. 编译、链接test.c，生成可执行文件test.exe
2. 在DOS状态下运行(test.exe所在路径下)

例如： C:\TC> test[.exe] hello world!

运行结果：hello
world!

运行结果：test
hello
world!

§10.8 有关指针的数据类型和指针运算的小结

★有关指针的数据类型的小结

定　义	含　义
int i;	定义整型变量i
int *p；	p为指向整型数据的指针变量
int a[n]；	定义有n个元素的整型数组a
int *p[n]；	定义由n个指向整型数据的指针元素组成的指针数组p
int (*p)[n]；	p为指向含n个元素的一维数组的指针变量
int *p( )；	p为返回一个指针的函数，该指针指向整型数据
int (*p)( )；	p为指向函数的指针，该函数返回一个整型值
int **p；	p为一个指针变量，它指向一个指向整型数据的指针变量
int f( )；	f为可返回整型函数值的函数

★ 指针运算小结

(1) 指针变量加（减）一个整数

如：p++；p--；p+i；p-i；p+=i；p-=i等，加减的值与类型有关。

(2) 指针变量赋值

p=&a （将变量a的地址赋给p)

p=array；(将数组array首地址赋给p)

p=&array[i]；(将数组array第i个元素的地址赋给p)

p=max；(max为已定义的函数，将max函数的入口地址赋给p)

p1=p2；(p1和p2都是指针变量，将p2的值赋给p1)

(3) 指针变量不指向任何变量，即取空值。表示为：p=NULL；

(4) 两个指针变量可以相减

如果两个指针变量指向同一个数组为元素，则两个指针变量值之差是两个指针之间的元素个数。但p1+p2并无实际意义。

(5) 两个指针变量比较

如果两个指针变量指向同一个数组为元素，则可以进行地址比较。

例　下列定义的含义

(1) int *p[3]; ← 指针数组

(2) int (*p)[3]; ← 指向一维数组的指针

(3) int *p(int); ← 返回指针的函数

(4) int (*p)(int); ← 指向函数的指针，函数返回int型变量

(5) int *(*p)(int);

(6) int (*p[3])(int); ← 函数指针数组，函数返回int型变量

(7) int *(*p[3])(int); ← 函数指针数组，函数返回int型指针

★int指针类型

- ❖ 定义时不指定指向哪一类数据
- ❖用动态存储分配函数时返回int指针
- ❖它的值赋给另一指针变量时，要强制类型转换

例：char *p1;

int *p2;

⋮

p1=(char *)p2;

或p2=(int *)p1;

int *fun(char ch1,char ch2)　返回“空类型”地址

P1=(char *)fun(ch1,ch2);　　引用时要类型转换

最后一课对同学们的祝福

感谢大家一学期以来的陪伴和支持，希望大家能够有所收获。

C语言只是计算机类知识学习的起步，今后路还很长；但是不必担心，“注重实践，多加练习”是学习的重要法宝。

以后可能的合作的机会：
0 本学期课程考试未通过
1 春季学期的“最优化方法”
（大数据专业必修，人工智能专业选修）
2 对科研有兴趣，愿意提前接触
3 URP/北创/国创（每年12月份左右，需提前联系）
4 硕士/博士研究生（需提前联系）
5 其他

zhaiweixin@cau.edu.cn
信电楼717