

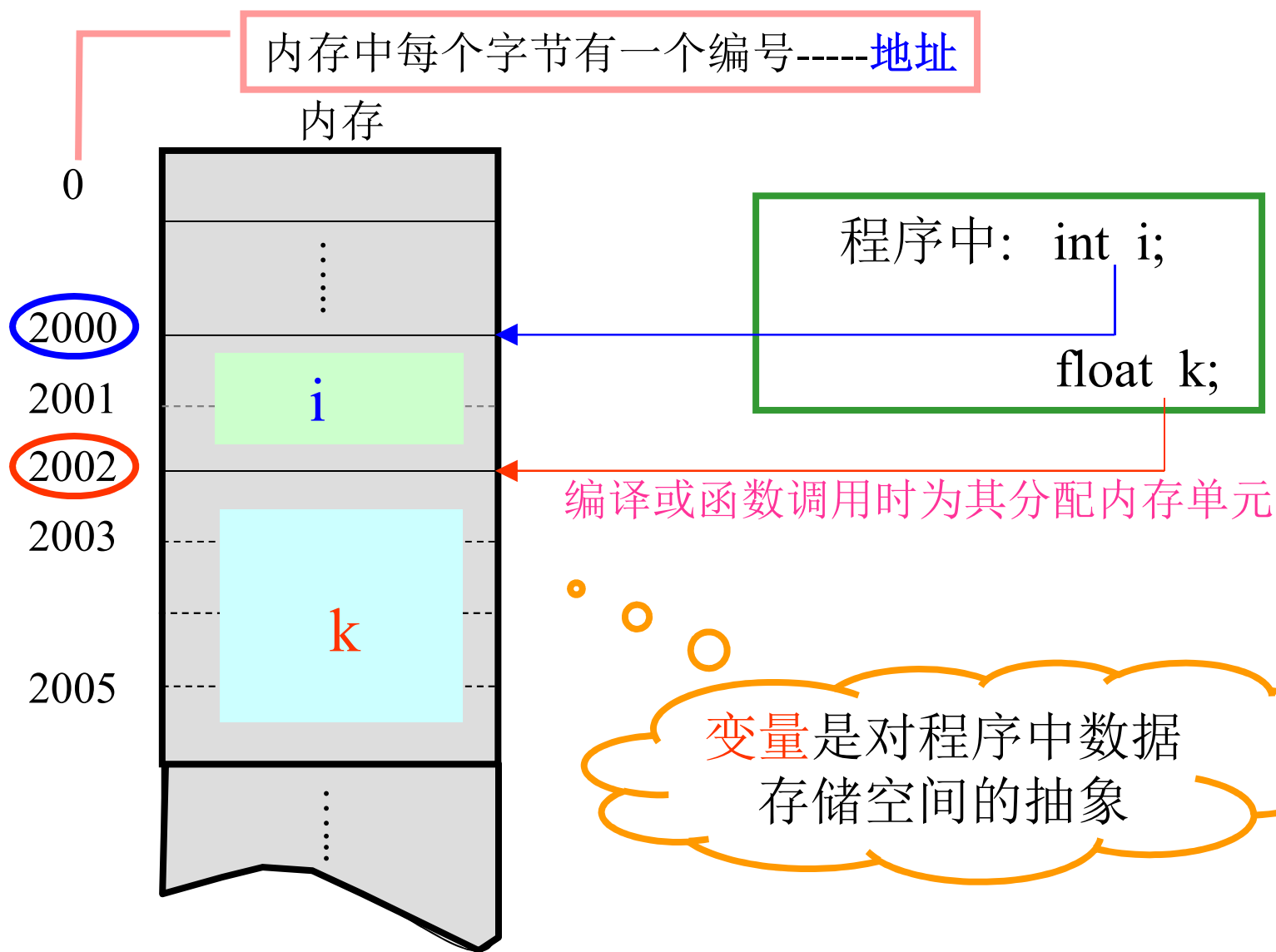
第六章 指针

C++程序设计中可以使用指针可以:

- 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
- 得到多于一个的函数返回值

6.1 指针的基本概念

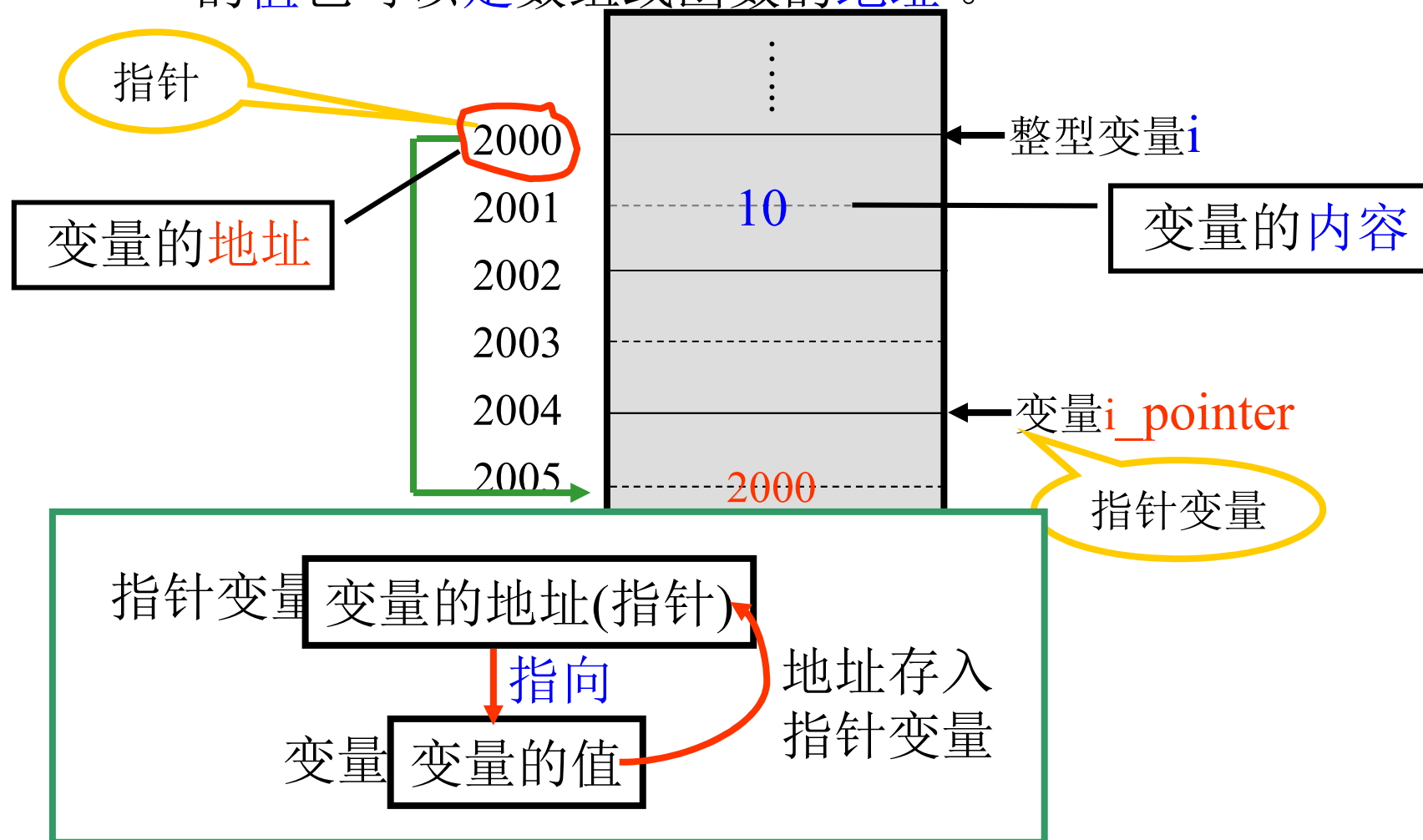
1. 变量与地址



2. 指针与指针变量

❖ 指针：一个变量的地址，它是一个整数形式的常量。

❖ 指针变量：专门用来存放地址的变量叫指针变量，它的值也可以是数组或函数的地址。



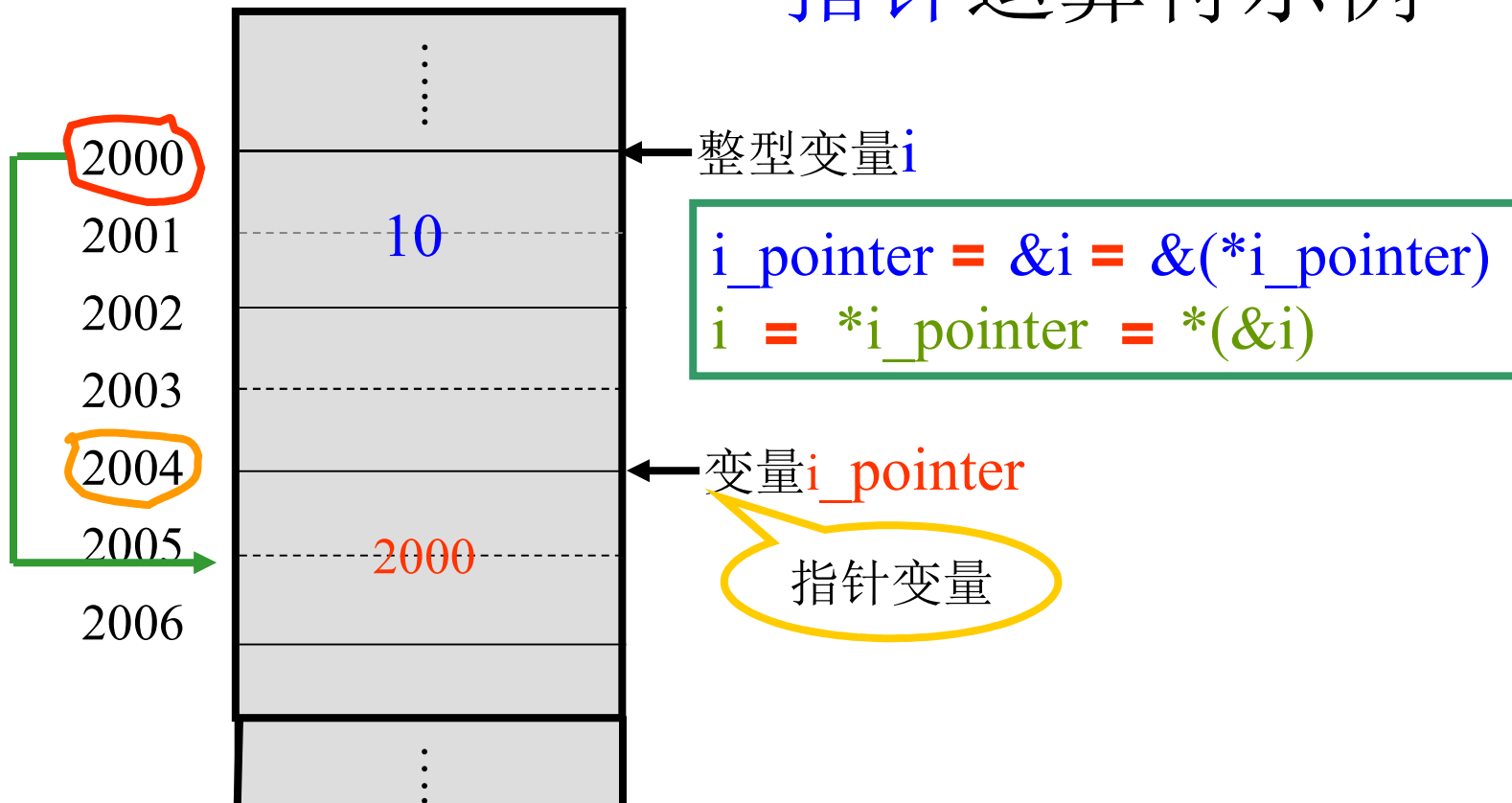
3. 取地址运算符&与指针运算符*

含义: 取变量的地址
单目运算符
优先级: 14 (第二高的级别)
结合性: 自右向左

含义: 从某个地址中获取数据
单目运算符
优先级: 14
结合性: 自右向左

两者关系: 互为逆运算

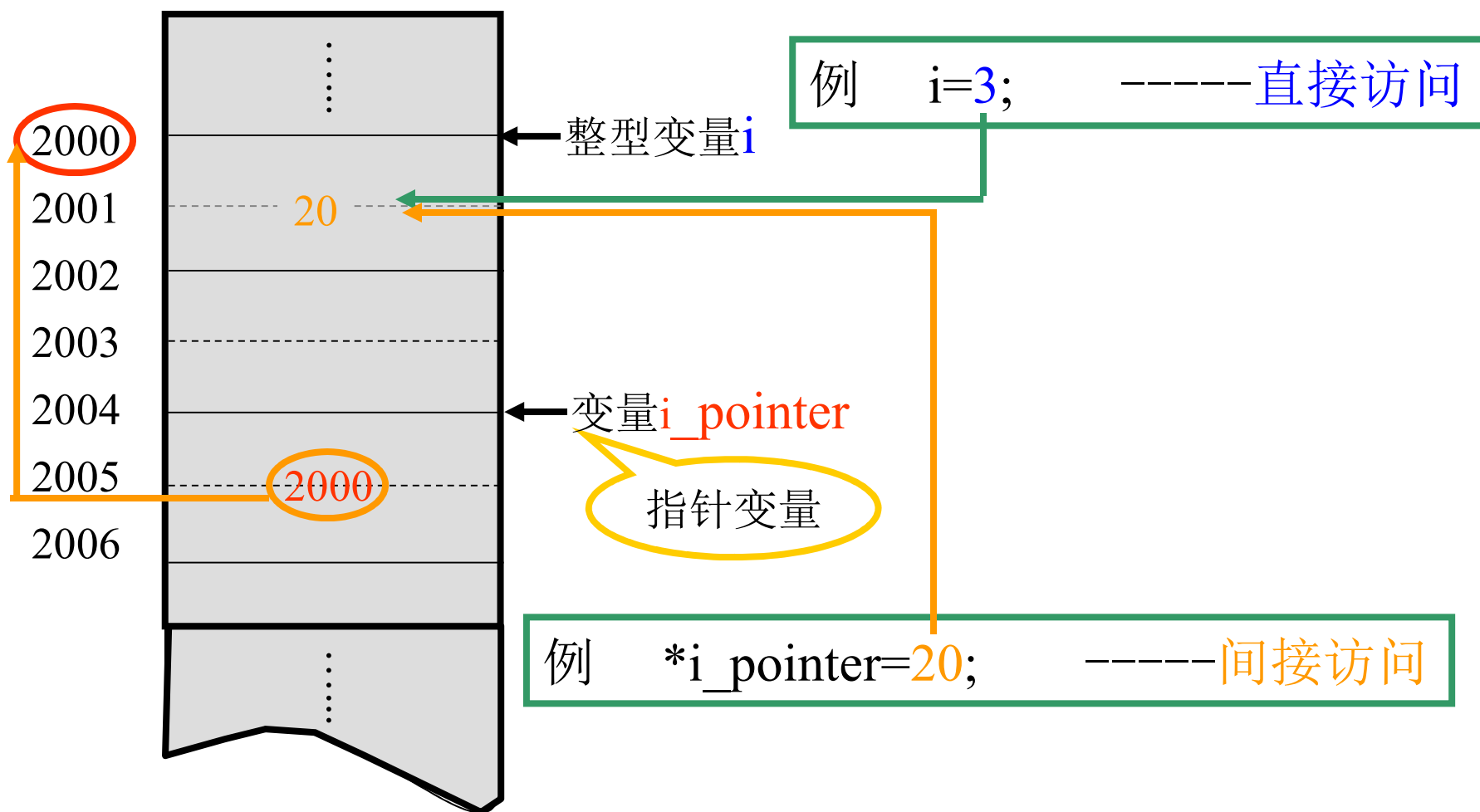
指针运算符示例



`i_pointer`-----指针变量，它的内容是地址量2000
`*i_pointer`----指针的**目标变量**`i`，它的内容是数据10
`&i_pointer`---指针变量占用内存的地址：2004

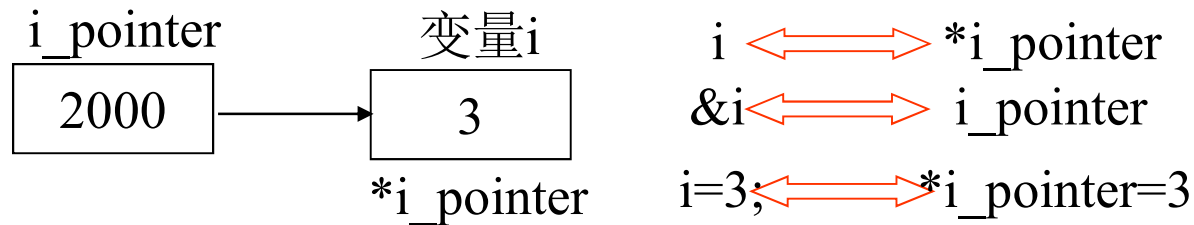
4. 直接访问与间接访问

- ❖ 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



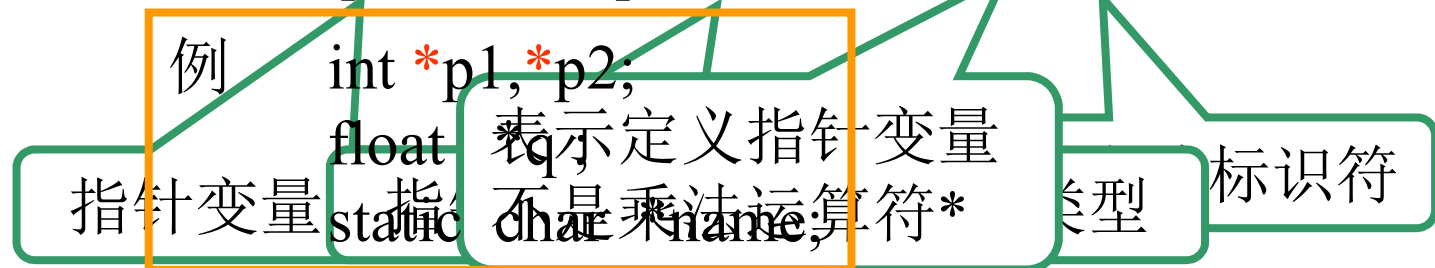
6.2 指针变量的定义与引用

1. 指针变量与其所指向的变量之间的关系



2. 指针变量的定义

❖ 一般形式: [存储类型] 数据类型 *指针名;



注意:

- 1、`int *p1, *p2;` 与 `int *p1, p2;` 不一样。
- 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, 变量值不确定, 应用前必须先赋值。

3.对指针变量的操作

(1) 指针变量的初始化

将地址值赋给指针变量

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；
并要求两者类型一致。

例 int i;
 int *p=&i;
 int *q=p;

例 int *p=&i;
 int i; X

用已初始化指针变量作初值

例 main()
 { int i;
 static int *p=&i; (X)

 }

不能用auto变量的地址
去初始化static型指针

3.对指针变量的操作(续)

(2) 指针变量 \pm 整数 \rightarrow 新的地址

```
int a,b,c,d,*p,*q;  
p=&b;  
q=p+1;
```

所加的数值: 整数 \times 字节数

$q=p-1$;

$p++$; $++p$;

注意: $*p++$; $*++p$;

不同于 $(*p)++$; $++(*p)$;

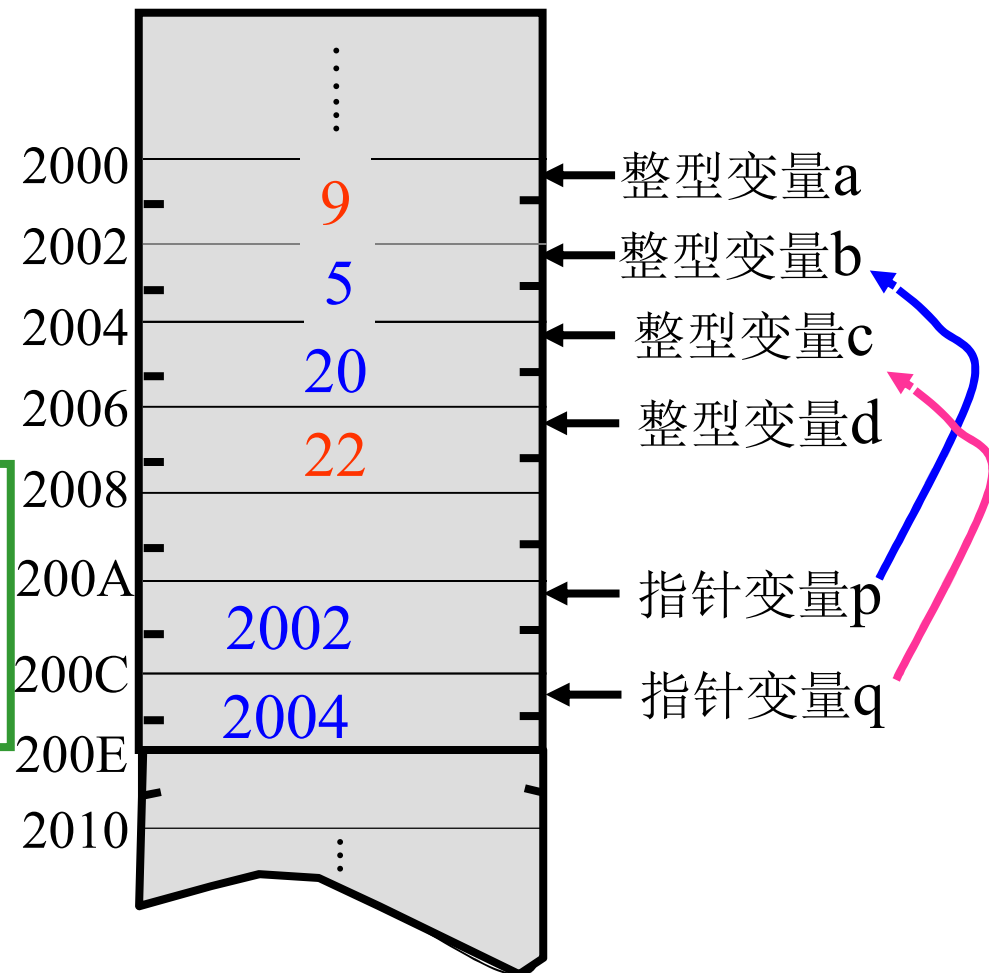
(3) 指针变量 - 指针变量

\rightarrow 整数(多少个数据)

$q-p \rightarrow 1$;

(4) 关系运算

$p < q \rightarrow 1$; $p == q \rightarrow 0$;

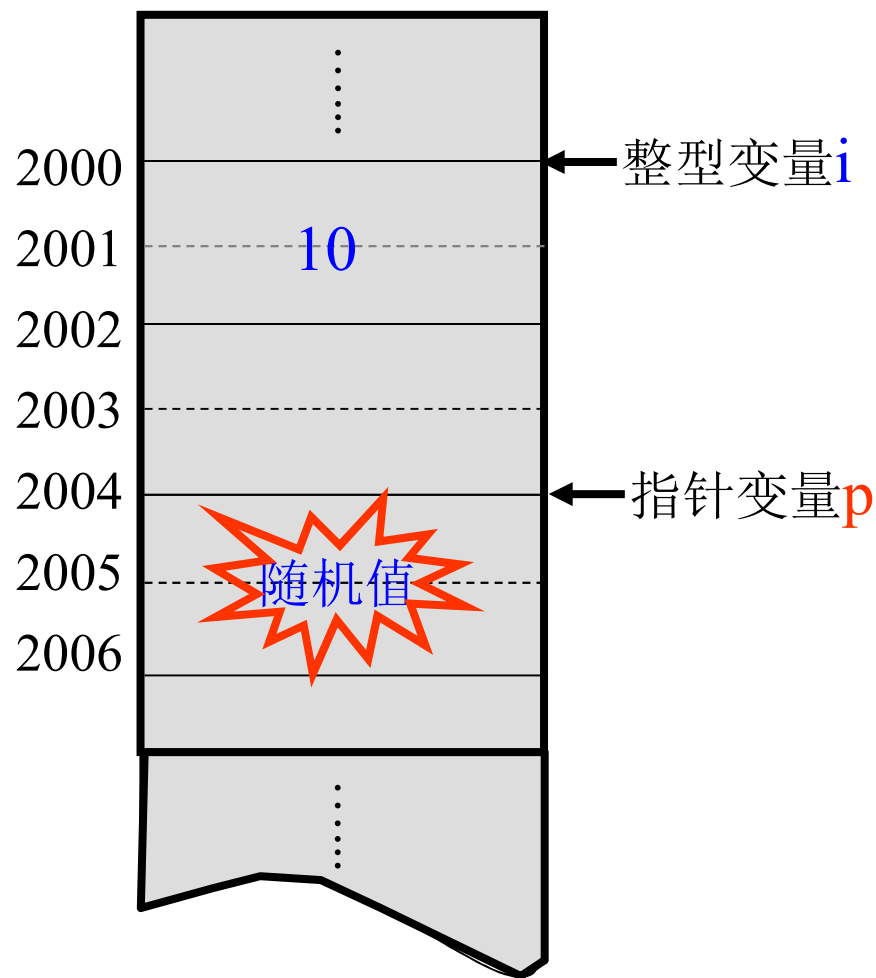


4. 指针变量必须先赋值, 再使用!

```
例 main( )  
{   int i=10;  
    int *p;  
    *p=i;  
    cout<<*p;  
}
```

危险!

```
例 main( )  
{   int i=10,k;  
    int *p;  
    p=&k;  
    *p=i;  
    cout<<*p;  
}
```



5. 零指针

★ 零指针与空类型指针

❖ 零指针：(空指针)

- 定义: 指针变量值为零

例如: `int *p=0;`

p指向地址为0的内存单元;
系统保证该单元不作它用;
表示指针变量的值没有意义。

```
#define NULL 0  
int *p=NULL;
```

- p=NULL与未对p赋值不同
- 用途:
 - ◆ 避免指针变量的非法引用
 - ◆ 在程序中常作为状态比较

```
例  int *p;  
    .....  
    while(p!=NULL)  
    { .....  
    }
```

6. 空类型指针

一般形式： `void *类型指针;`

例如： `void *p;`

表示不指定p是指向哪一种类型数据的指针变量。使用时要进行强制类型转换。

```
例 char *p1;  
    void *p2;  
    p1=(char *)p2;  
    p2=(void *)p1;
```

TC中分配内存空间的函数返回一个空类型的指针。

```
void *malloc(int n);
```

```
例如： int *p= (int *) malloc(2);
```

例6.3 按先大后小的顺序输出a和b两个整数。

```
main()
{  int *p1,*p2,*p,a,b;
    a=5;b=9;
    p1=&a; p2=&b;
    if(a<b)
    { p=p1; p1=p2; p2=p;}
    cout<<a<<b;

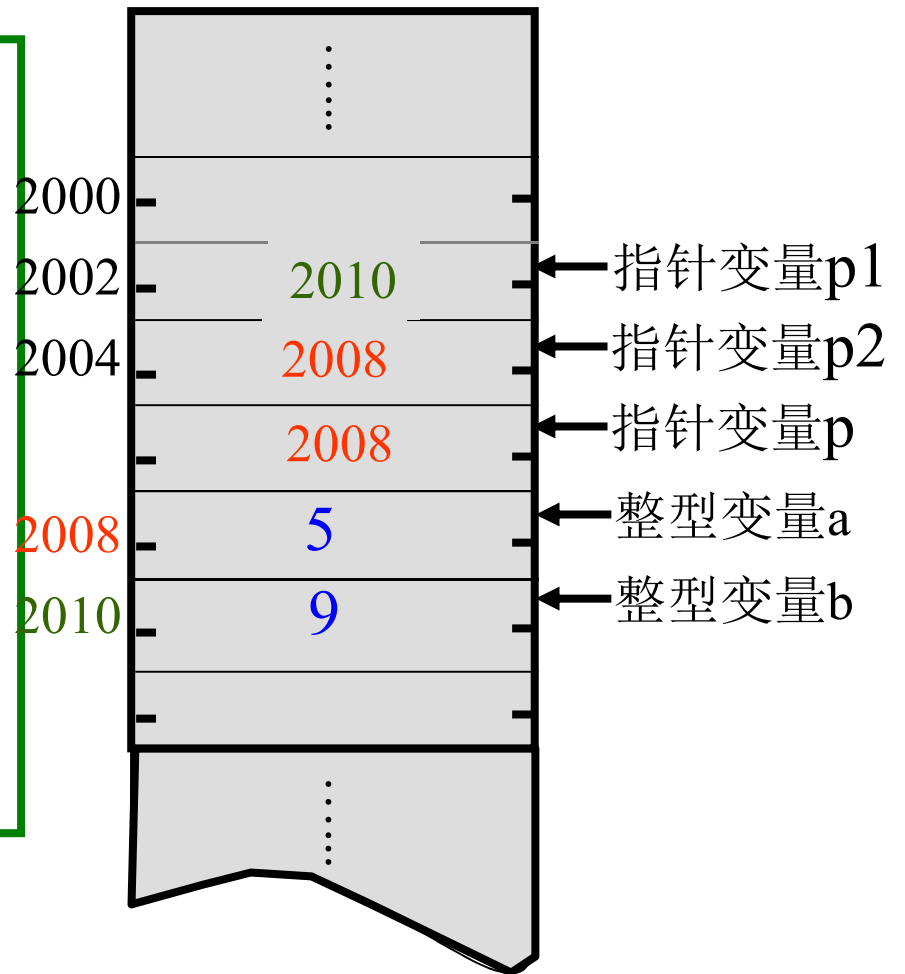
    cout<<"max="<<"min="<<*p1<<*p2;
}
```

运行结果:

5,9<CR>

a=5,b=9

max=9,min=5



7. 多重指针

一个指针变量的内容就是内存中某个存储区域的地址，这个存储区域中存放的值可以是一个基本数据类型的数据，也可以是另一个存储区域的地址。我们把这种类型的指针叫做多重指针。

二重指针(指向指针的指针)的一般说明形式为：

类型说明符 **指针变量名；

二重指针的使用。

```
main()
{ int *p1, **p2, i = 10;
  p1=&i; p2=&p1;
}
```

6.3 函数之间地址值的传递

1. 形参为指针变量：传递的是指针变量的值----地址。

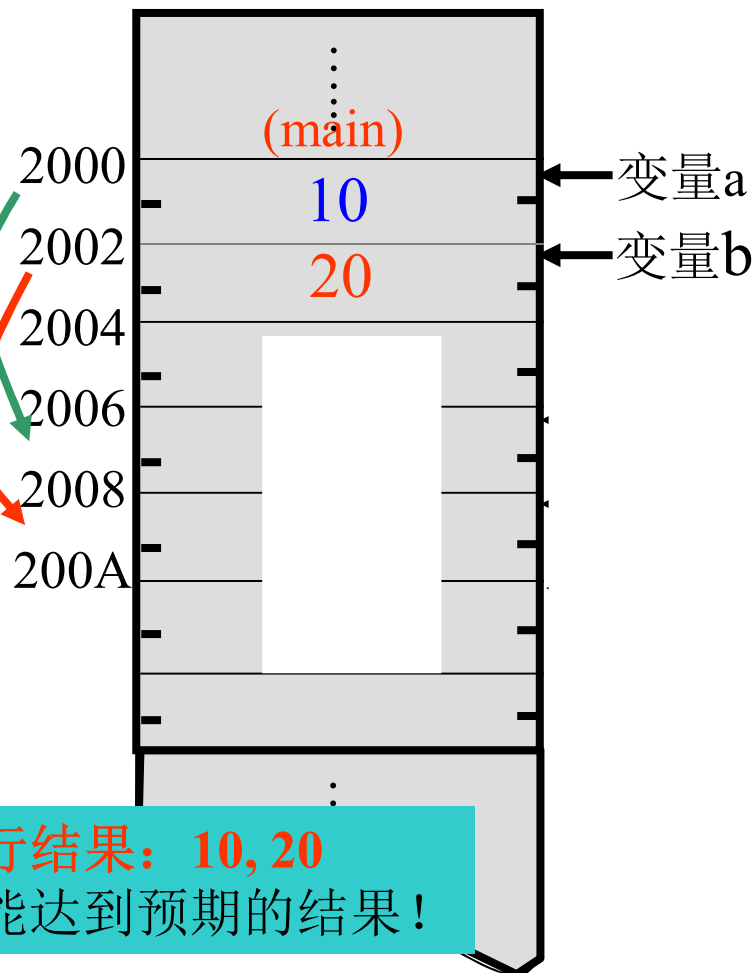
特点：共享内存, 相当于“双向”传递！

例6.5 将数从大到小输出(用变量作函数的参数)

```
void swap(int x,int y)
{ int temp;
  temp=x;
  x=y;
  y=temp;
}
void main()
{ int a=10,b=20;
  if(a<b) swap(a,b);
  cout<<a<<b;
}
```

值传递

COPY



例6.5 将数从大到小输出(用指针作函数的参数)



Ch9_3. c

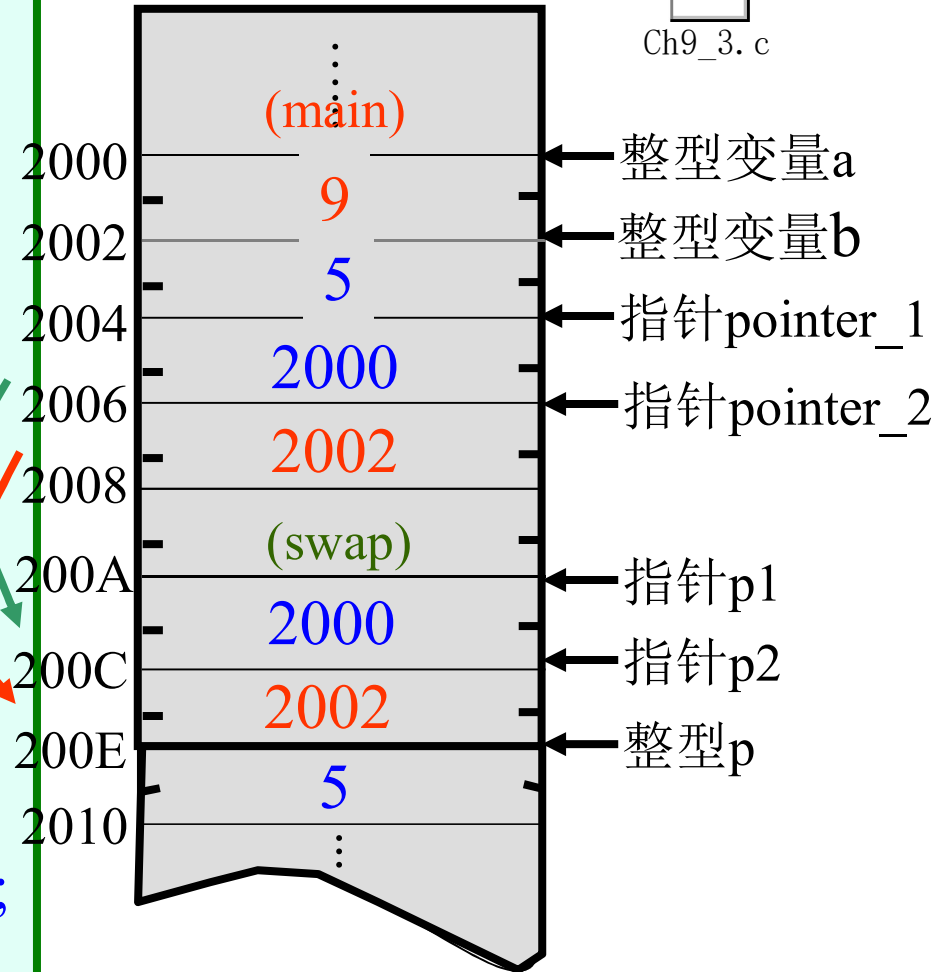
```
void swap(int *p1, int *p2)
```

```
{  int p;  
    p=*p1;  
    *p1=*p2;  
    *p2=p;  
}
```

```
main()
```

```
{  int a,b;  
    int *pointer_1,*pointer_2;  
    a=5;  b=9;  
    pointer_1=&a; pointer_2=&b;  
    if(a<b) swap(pointer_1,pointer_2);  
    cout<<a<<b;  
}
```

COPY



例6.5 将数从大到小输出(用指针作函数的参数) 续

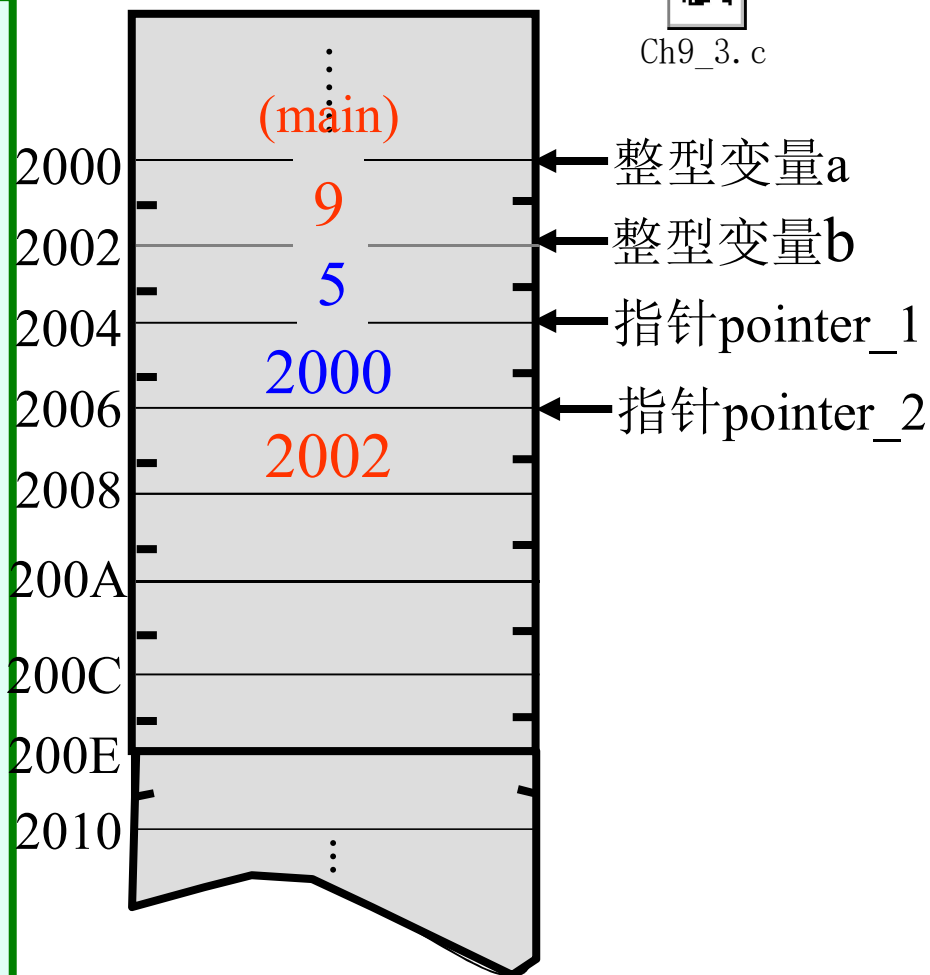


Ch9_3.c

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

main()
{
    int a,b;
    int *pointer_1,*pointer_2;
    a=5; b=9;
    pointer_1=&a; pointer_2=&b;
    if(a<b)swap(pointer_1,pointer_2);
    cout<<a<<b;
}
```

地址值传递



运行结果: 9, 5

例10.5 注意：指针变量要先赋值后，才能进行指针运算！



Ch9_31.c

```
void swap(int *p1, int *p2)
```

```
{  int *p;  
    *p=*p1;  
    *p1=*p2;  
    *p2=*p;  
}
```

加两语句改正

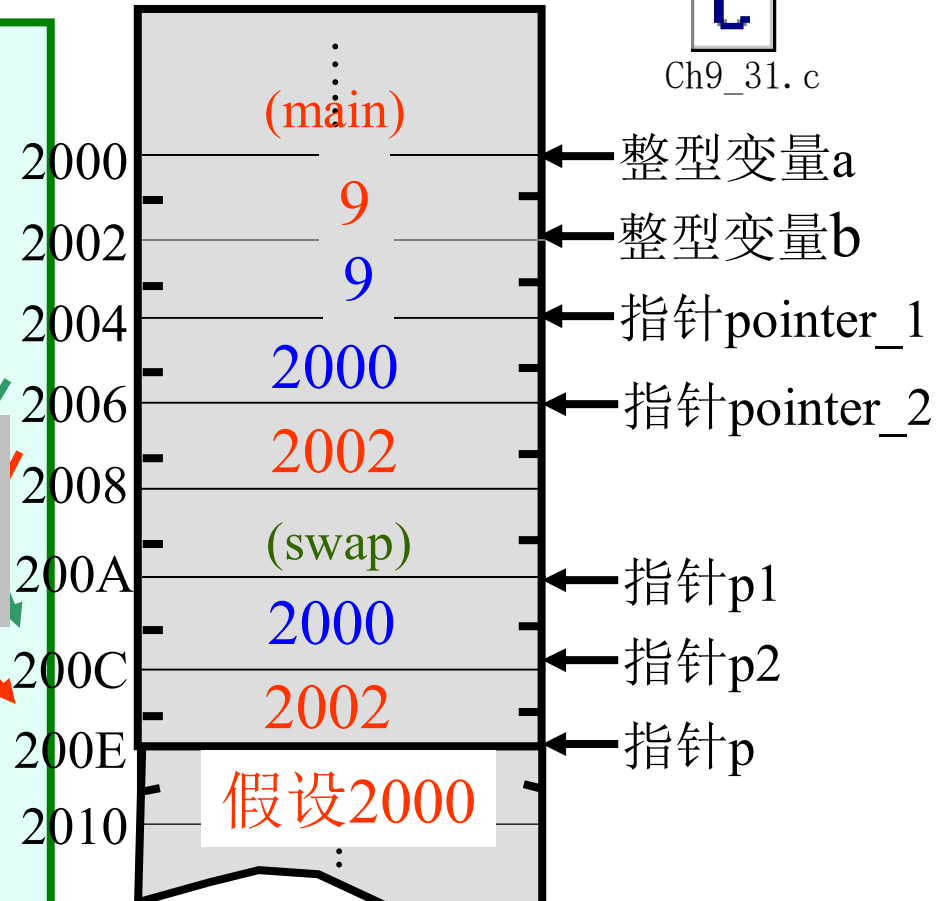
```
int  x;  
int  *p=&x;x;
```

编译警告！
结果有时也正确！
但也有错的时候！

```
main()
```

```
{  int a,b;  
    int *pointer_1,*pointer_2;  
    a=5; b=9;  
    pointer_1=&a; pointer_2=&b;  
    if(a<b) swap(pointer_1,pointer_2);  
    cout<<a<<b;  
}
```

运行结果：9，9



指针变量在使用前
必须赋值！

例6.5 错误程序之二



Ch9_32.c

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

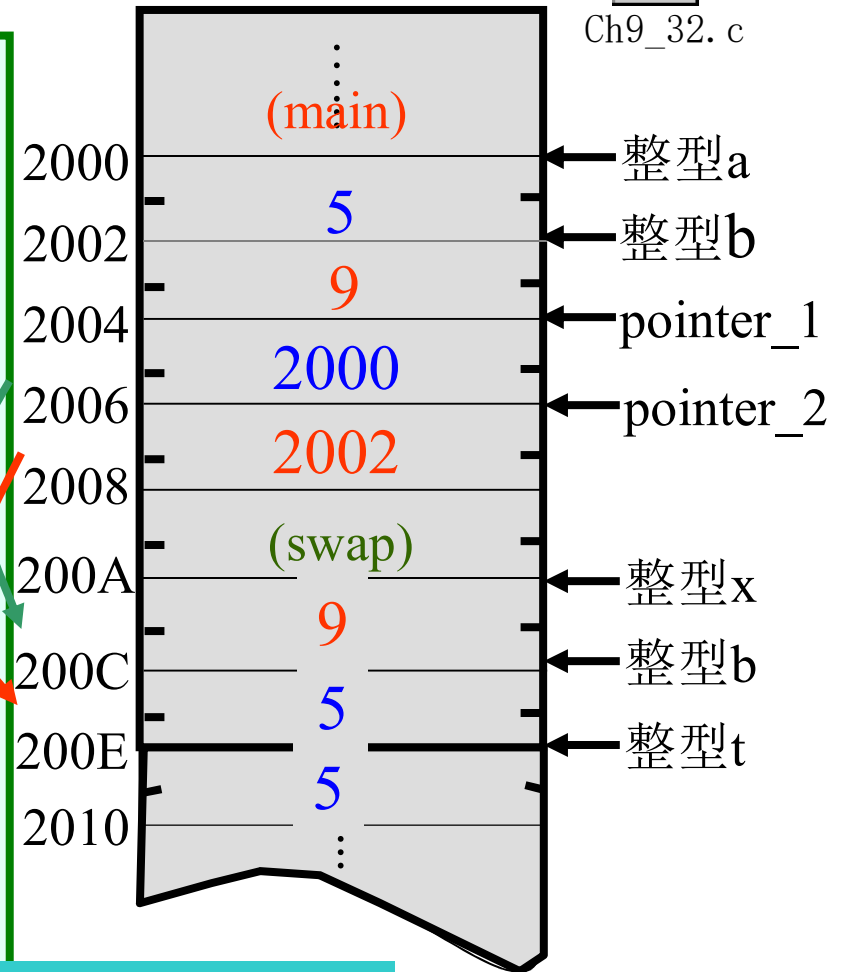
```
{  int t;  
    t=x; x=y; y=t;  
}
```

```
main()
```

```
{  int a,b;  
    int *pointer_1,*pointer_2;  
    cin>>a>>b;  
    pointer_1=&a; pointer_2=&b;  
    if(a<b) swap(*pointer_1,*pointer_2);  
    cout<<a<<b;  
}
```

值传递

COPY



运行结果：5，9
也不能达到预期的结果！

例6.5 错误程序之三

```
swap(int *p1, int *p2)
```

```
{ int *p;
```

```
  p=p1;
```

```
  p1=p2;
```

```
  p2=p;
```

```
}
```

```
main()
```

```
{ int a,b;
```

```
  int *pointer_1,*pointer_2;
```

```
  cin>>a>>b;
```

```
  pointer_1=&a; pointer_2=&b;
```

```
  if(a<b) swap(pointer_1,pointer_2);
```

```
  cout<<a<<b;
```

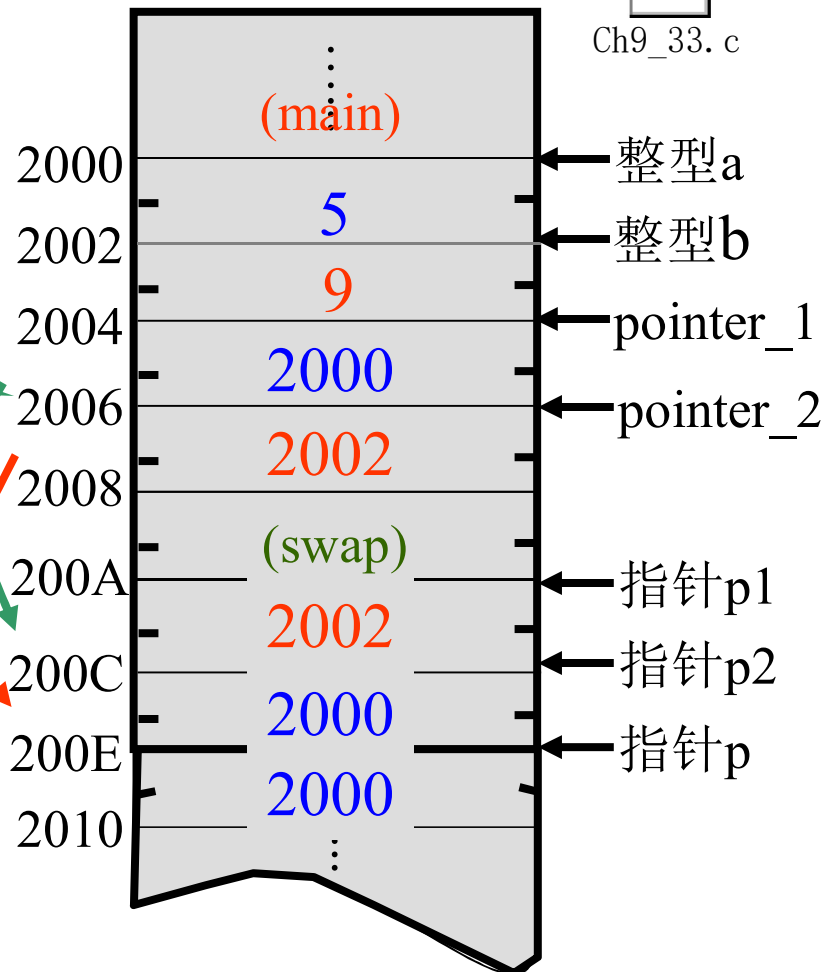
```
}
```

地址传递

COPY



Ch9_33.c



运行结果：5, 9

也不能达到预期的结果！

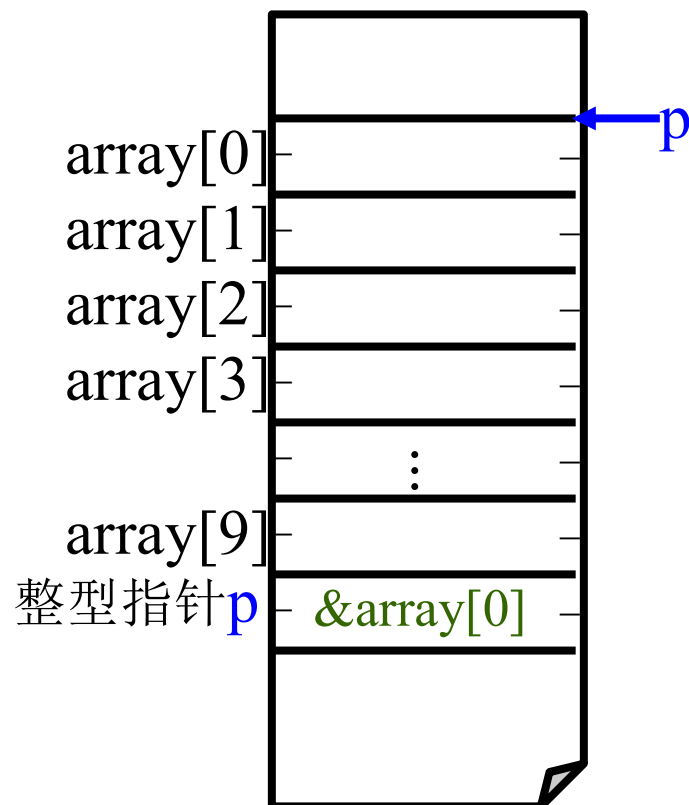
6.3 数组和指针

1. 一维数组和数组元素的地址

一个整数在内存中占两个连续的存储单元，排在前面的那个存储单元的地址就是这个整数的地址；长整数、实数……

数组元素的地址同上。

数组中的若干个数组元素在内存中是依次连续存放的，占一片连续的内存单元，其中排在前面的那个数组元素的地址就是这个数组的地址。



数组名是表示数组首地址的地址常量!

例6.7 数组以及各个数组元素在内存中的地址

```
void main()
{   int i, a[10];
    cout<< "index, Address, size:\n" ;
    for( i = 0; i < 10; i++ )
    cout<< " &a[" <<i<< "], " <<&a[i]<<" " <<
    cout<< "Address of a ="<<a<<endl;
    cout<< "size of a ="<<sizeof(a)<<endl; }
```

在C++语言中，一维数组的任何一个元素的地址，都可以用其数组名加上一个偏移量来表示。

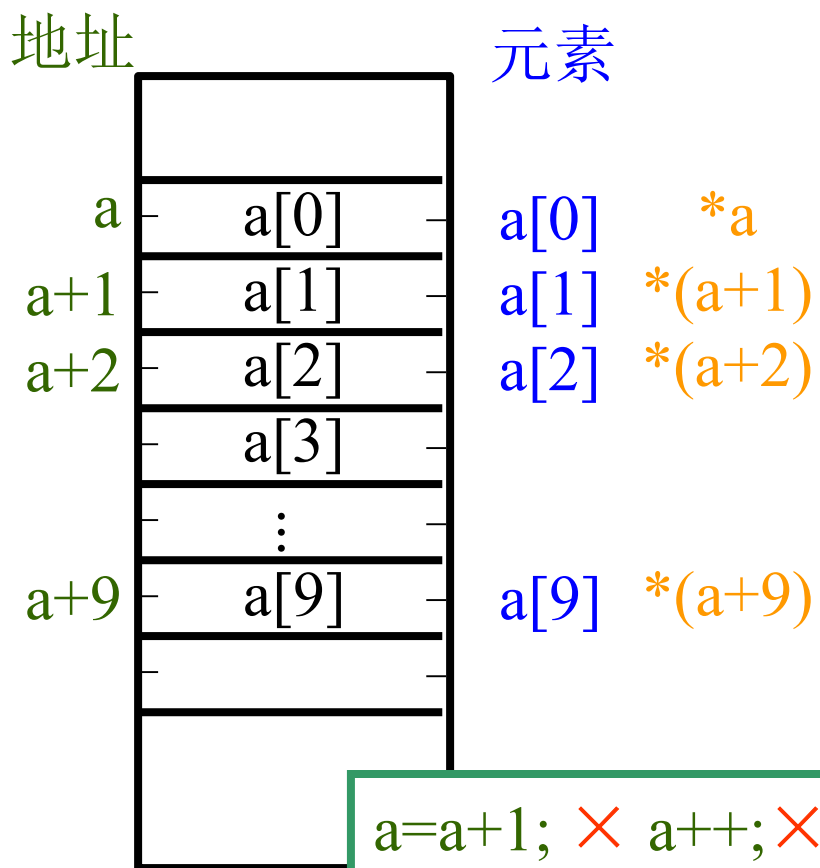
即: $\&a[i] \longleftrightarrow a+i$
 $*\&a[i] \longleftrightarrow a[i] \longleftrightarrow *(a+i)$

程序运行结果如下(VC):

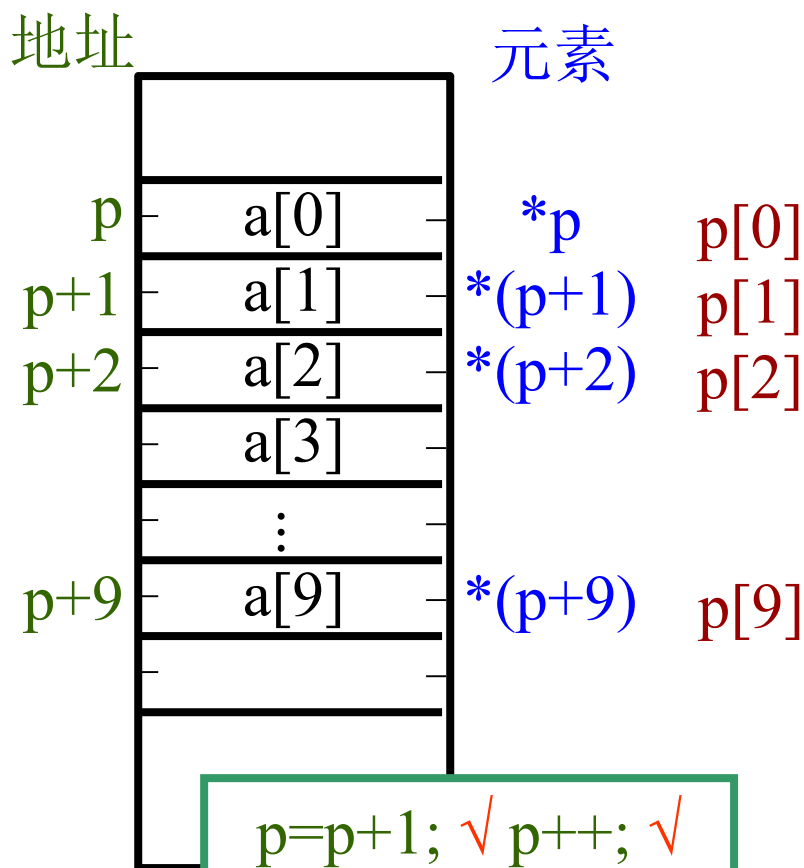
```
index, Address, size:
&a[ 0 ], 0x12ff54, 4
&a[ 1 ], 0x12ff58, 4
&a[ 2 ], 0x12ff5c, 4
&a[ 3 ], 0x12ff60, 4
&a[ 4 ], 0x12ff64, 4
&a[ 5 ], 0x12ff68, 4
&a[ 6 ], 0x12ff6c, 4
&a[ 7 ], 0x12ff70, 4
&a[ 8 ], 0x12ff74, 4
&a[ 9 ], 0x12ff78, 4
Address of a = 0x12ff54
size of a = 40
```

2. 通过指针引用数组元素

```
int a[10], *p=&a[0];
```

$$a[i] \Leftrightarrow p[i] \Leftrightarrow *(p+i) \Leftrightarrow *(a+i)$$


下标法

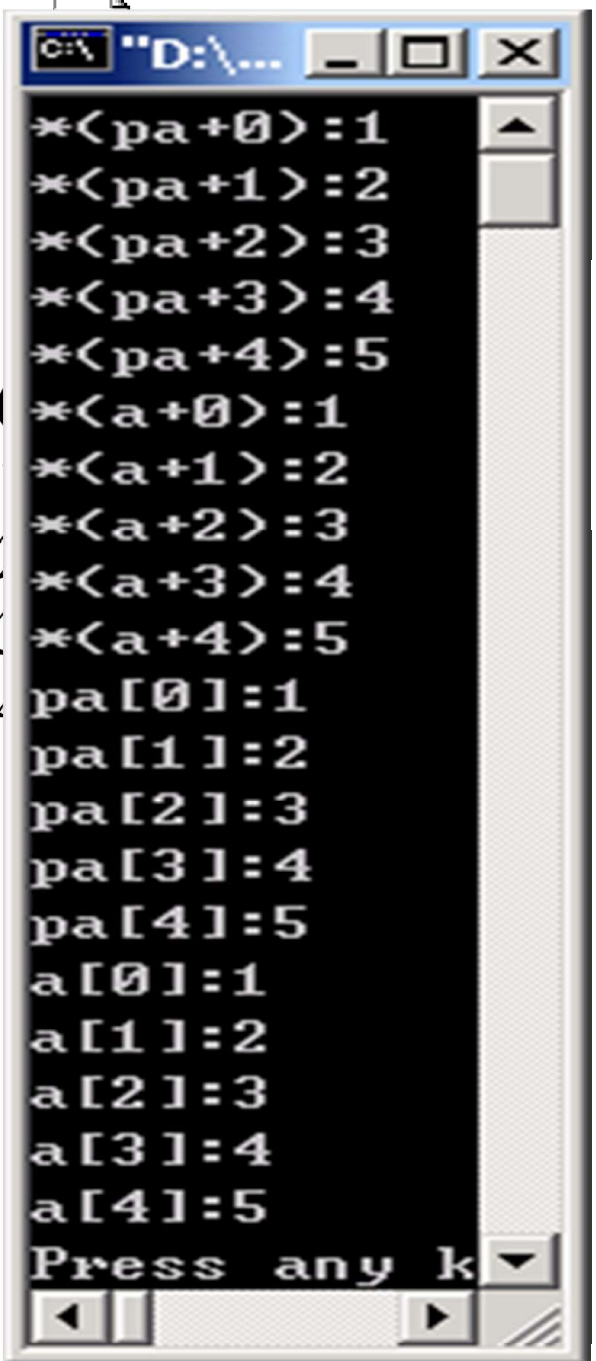


指针法

例 数组元素的引用方法

```
void main()
{int a[5],*pa,i;
for(i=0;i<5;i++)
a[i]=i+1;
pa=a;
for(i=0;i<5;i++)
    cout<<"*(pa+"<<i<<"):"<<*(pa+i)<<endl;
for(i=0;i<5;i++)
    cout<<"*(a+"<<i<<"):"<<*(a+i)<<endl;
for(i=0;i<5;i++)
    cout<<"pa["<<i<<"]:"<<<<pa[i]<<endl;
for(i=0;i<5;i++)
    cout<<"a["<<i<<"]:"<<a[i]<<endl;
}
```

a[0]
a[1]
a[2]
a[3]
a[4]



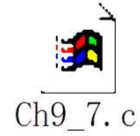
```
*(<pa+0>):1
*(<pa+1>):2
*(<pa+2>):3
*(<pa+3>):4
*(<pa+4>):5
*(<a+0>):1
*(<a+1>):2
*(<a+2>):3
*(<a+3>):4
*(<a+4>):5
pa[0]:1
pa[1]:2
pa[2]:3
pa[3]:4
pa[4]:5
a[0]:1
a[1]:2
a[2]:3
a[3]:4
a[4]:5
Press any key
```


3. 数组名或指针作形参

❖ 数组名作函数参数，实参与形参的对应关系

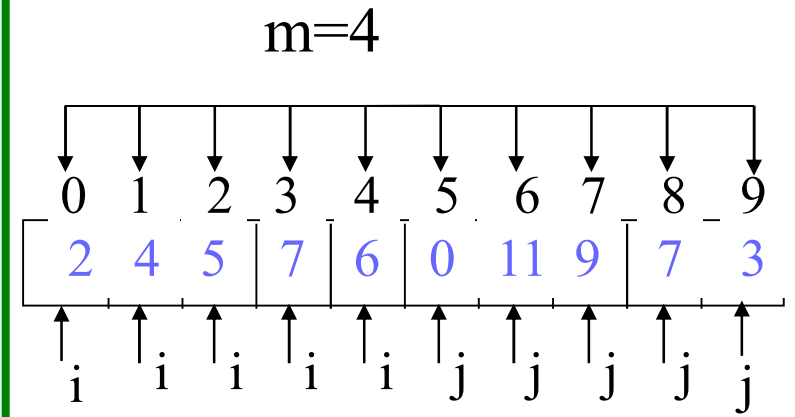
实参	形参
数组名	数组名
数组名	指针变量
指针变量	数组名
指针变量	指针变量

例 将数组a中的n个整数按相反顺序存放



```
void 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;
    }
}

main()
{   int i,a[10]={3,7,9,11,0,6,7,5,4,2};
    inv(a,10);
    cout<<"The array has been reverted:\n";
    for(i=0;i<10;i++)
        cout<<a[i];
    cout<<endl;
}
```



1. 实参与形参均用数组

例 将数组a中的n个整数按相反顺序存放



Ch9_71.c

```
void inv(int *x, int n)
{   int t,*p,*i,*j,m=(n-1)/2;
    i=x; j=x+n-1; p=x+m;
    for(;i<=p;i++,j--)
        { t=*i; *i=*j; *j=t; }
}

main()
{   int i,a[10]={3,7,9,11,0,6,7,5,4,2};
    inv(a,10);
    cout<<"The array has been reverted:\n";
    for(i=0;i<10;i++)
        cout<<a[i];
    cout<<endl;
}
```

2. 实参用数组,形参用指针变量

例 将数组a中的n个整数按相反顺序存放



Ch9_9. c

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

```
{ int t,*i,*j,*p,m=(n-1)/2;
  i=x; j=x+n-1; p=x+m;
  for(;i<=p;i++,j--)
  { t=*i; *i=*j; *j=t; }
}
```

```
main()
```

```
{ int i,a[10],*p=a;
  for(i=0;i<10;i++,p++)
    cin>>*p;
```

```
  p=a; inv(p,10);
```

```
  cout<<"The array has been reverted:\n";
```

```
  for(p=a;p<a+10;p++)
```

```
    cout<<*p;
```

```
}
```

3. 实参与形参均用指针变量

例 将数组a中的n个整数按相反顺序存放



Ch9_72.c

```
void 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;
    }
}

main()
{   int i,a[10],*p=a;
    for(i=0;i<10;i++,p++)
        cin>>*p;
    p=a;   inv(p,10);
    cout<<"The array has been reverted:\n";
    for(p=arr;p<arr+10;p++)
        cout<<*p;
}
```

4. 实参用指针变量,形参用数组

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 $p[i] \Leftrightarrow q[i] \Leftrightarrow *(p+i) \Leftrightarrow *(q+i)$

形参数组实质上是指针变量, 即`int q[]` \Leftrightarrow `int *q`

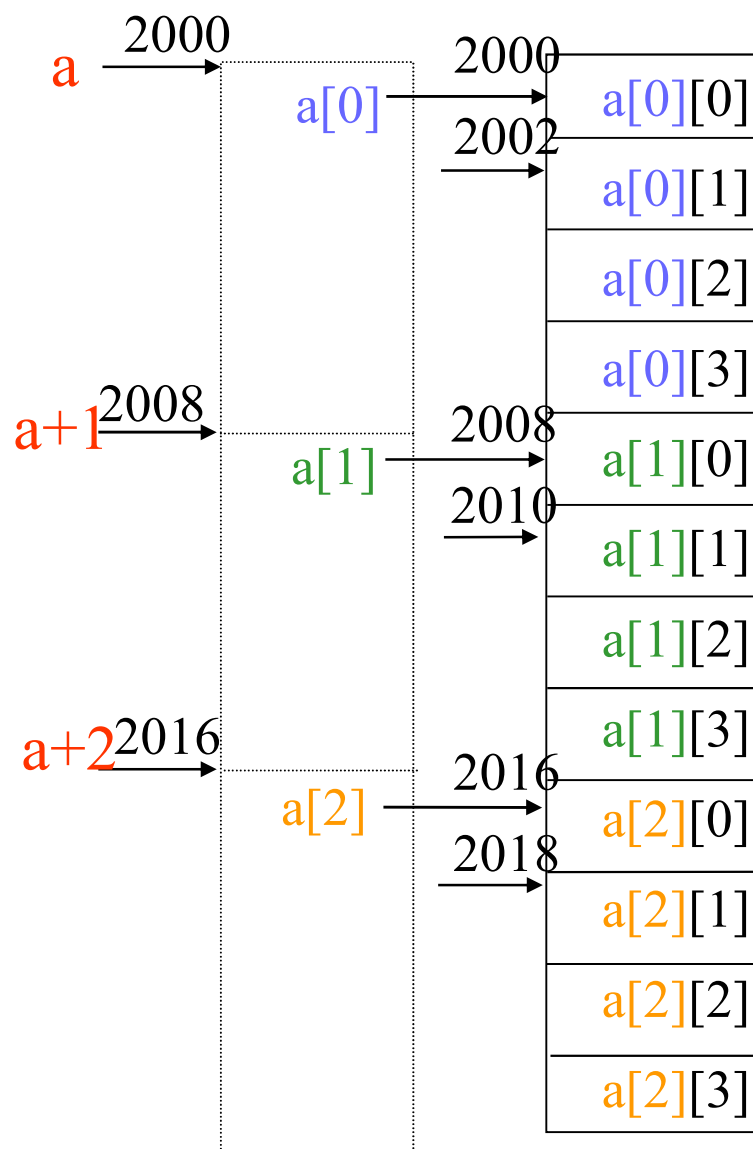
在定义指针变量（不是形参）时, 不能把`int *p` 写成`int p[]`;

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

6.4.5 指针与二维数组

1. 二维数组和数组元素的地址



● 对二维数组 `int a[3][4]`, 有

◆ `a`-----二维数组的首地址, 即第0行的首地址

◆ `a+i`-----第*i*行的首地址

◆ `a[i] ⇔ *(a+i)`-----第*i*行第0列的元素地址

◆ `a[i]+j ⇔ *(a+i)+j` -----第*i*行第*j*列的元素地址

◆ `*(a[i]+j) ⇔ *(*a+i)+j ⇔ a[i][j]`

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

◆ `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

int a[3][4];

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

地址表示:

- (1) $a+1$
- (2) $\&a[1][0]$
- (3) $a[1]$
- (4) $*(a+1)$
- (5) $(\text{int } *) (a+1)$

← 行指针

} 列指针

地址表示:

- (1) $\&a[1][2]$
- (2) $a[1]+2$
- (3) $*(a+1)+2$
- (4) $\&a[0][0]+1*4+2$

二维数组元素表示形式:

- (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 用指针变量指向二维数组的数组元素

```
main()
{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)
    cout<<endl;
    cout<<*p;
  }
}
```

p=*a;



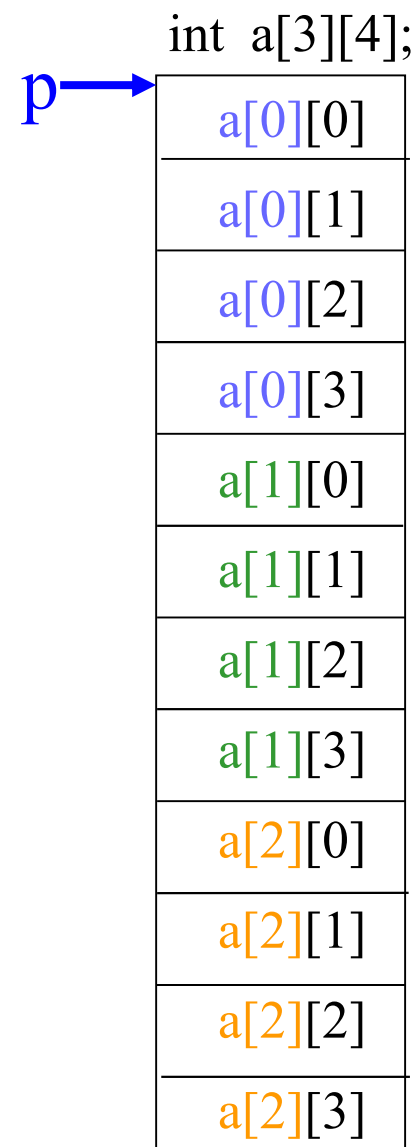
p=&a[0][0];



p=(int *)a;



p=a;



Ch9_12. c

2. 通过建立指针数组和行指针引用二维数组()

◆ 定义形式：数据类型 (*指针名)[一维数组的元素个数];

例 int (*p)[4];

() 不能少
(*p) 说明 p 是一个指针变量!

(*p)[4] 说明 p 的值是某个包含 4 个元素的一维数组的首地址，p 是行指针

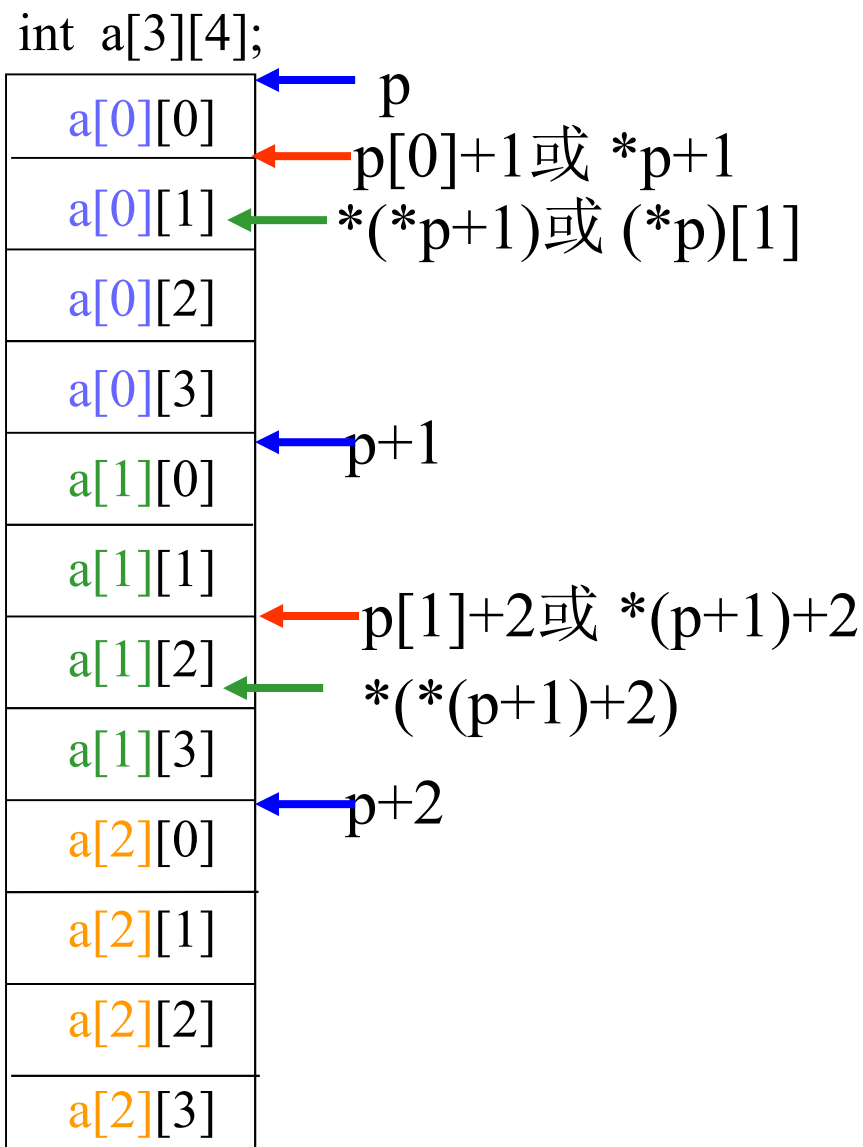
可让 p 指向二维数组某一行

如 int a[3][4], (*p)[4]; p=a;

p 指向的一维数组的元素个数和二维数组列数必须相同

◆ 指针数组：如 int a[3][4], *q[4];

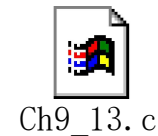
q[0]=&a[0][0]; q[1]=&a[1][0];



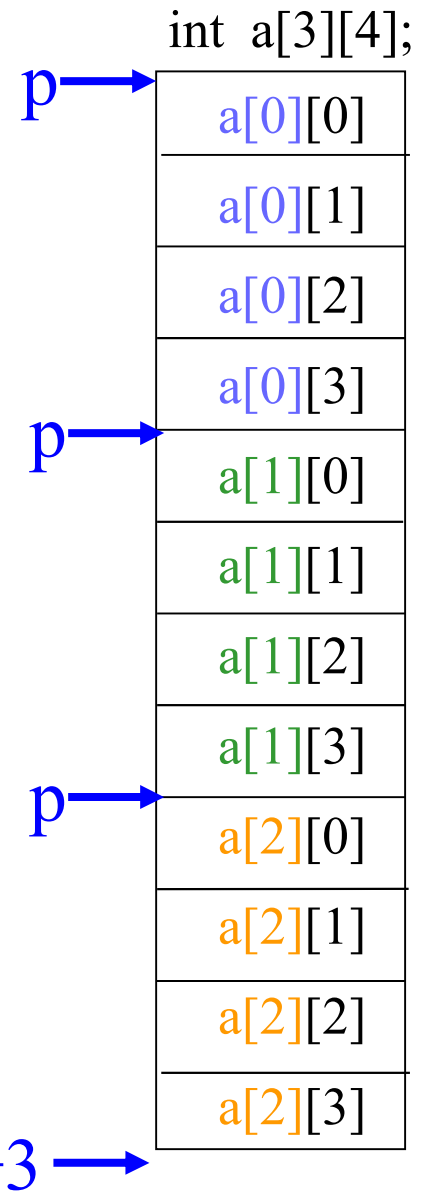
例 指向一维数组的指针变量(行指针)应用

```
main()
{
    static int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
    int i,j,(*p)[4];
    for(p=a,i=0;i<3;i++,p++) ⇔ for(p=a,p<a+3;p++)
        for(j=0;j<4;j++)
            cout<<*(*p+j);    ⇔ p[0][j]
    cout<<endl;
}
```

p=a[0]; ❌
p=*a; ❌
p=&a[0][0]; ❌
p=a; ⇔ p=&a[0]; ✓



Ch9_13.c



3. 二维数组名或行指针作函数的形参

通过指针引用二维数组的几种形式：

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

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

例 3个学生各学4门课，计算总平均分，并输出第n个学生成绩

函数说明

```
main()
{ void average(float *p,int n);
  void 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);
}
```

列指针

行指针

float p[][4]

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

```
void 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;
  cout<<"average="<<aver;
}

void search(float (*p)[4], int n)
{ int i;
  cout<<" No:"<<n;
  for(i=0;i<4;i++)
    cout<<*(*(p+n)+i);
}
```

$*(*(p+n)+i) \Leftrightarrow p[n][i]$



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



Ch9_15.c

```
void search(float (*p)[4], int n)
{
    int i,j,flag;
    for(j=0;j<n;j++)
    {
        flag=0;
        for(i=0;i<4;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

$*(*(p+j)+i) \Leftrightarrow p[j][i]$

```
main()
{
    void search(float (*p)[4], int n);
    float score[3][4]={ {...}, {...}, {...} };
    search(score,3);
}
```

二维数组与指向一维数组的指针变量的关系

如有： `int a[5][10], (*p)[10]; p = a;`

- ☆ 系统给数组a分配2*5*10个字节的内存区。
- ☆ 系统只给变量p分配能保存一个指针值的内存区(2字节);
- ☆ 数组名a的值是一个指向有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个指向一维数组的指针变量,

即: `fun(int x[][10]) ⇔ fun(int (*x)[10])`

在函数fun中两者都可以有`x++;x=x+2;`等操作!

但在变量定义(不是形参)时, 两者不等价;

指针数组与二级指针的关系

`int **p` 与 `int *q[10]`

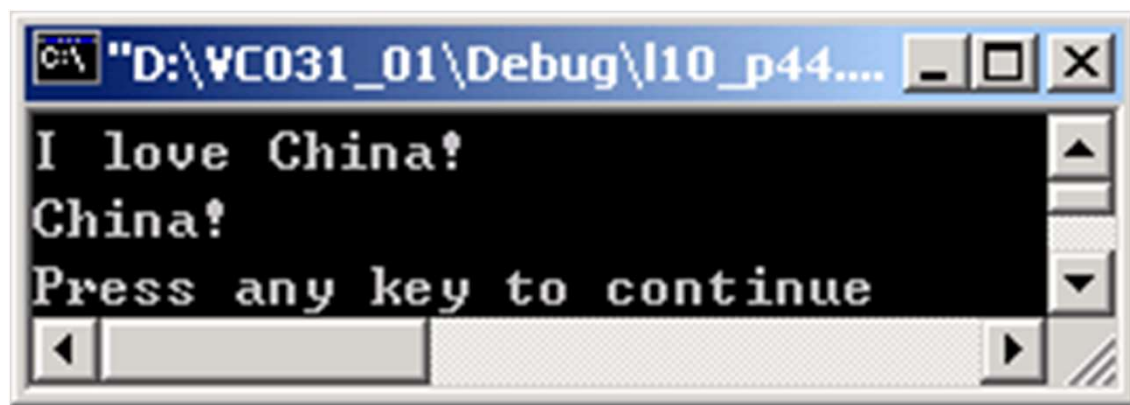
- ★ 系统只给p分配能保存一个指针值的内存区；而给q分配10个内存区，每个内存区均可保存一个指针值；
- ★ 指针数组名是二级指针常量；
- ★ `p=q`；`p+i` 是`q[i]`的地址；
- ★ 指针数组作形参，`int *q[]`与`int **q`完全等价；但作为变量定义两者不同。

6.4 指针与字符串

1. 字符串表示形式

(1) 用字符数组实现

```
例 main()  
{ char string[]="I love China!";  
  cout<<string<<endl;  
  cout<<string+7;  
}
```



string →	I	string[0]
		string[1]
	l	string[2]
	o	string[3]
	v	string[4]
	e	string[5]
		string[6]
	C	string[7]
	h	string[8]
	i	string[9]
	n	string[10]
	a	string[11]
	!	string[12]
	\0	string[13]

(2) 用字符指针实现



Ch9_17.c

字符指针初始化: 把字符串首地址赋给string
相当于以下两个语句:

```
char *string;  
string="I love China!";
```

例 main()

```
{ char *string="I love China!";  
  cout<<string<<endl;  
  string+=7;  
  while(*string)   
  {   putchar(string[0]);  
      string++;  
  }  
}
```

*string!=0

*string

string

string

I
l
o
v
e
C
h
i
n
a
!
\0

2. 用指向字符串的指针作函数参数

例 字符串复制

(1) 用字符数组
作参数

```
void copy_string(char from[],char to[])
{   int i=0;
    while(from[i]!='\0')
    {   to[i]=from[i];
        i++;
    }
    to[i]='\0';
}

main()
{   char a[]="I am a teacher.";
    char b[]="You are a student.";
    cout<<"string_a="<<a<<" string_b="<<b;
    copy_string(a,b);
    cout<<"string_a="<<a<<" string_b="<<b;}
```

(2) 用字符指针变量作参数

例10.19 实现字符串复制

```
void copy_string(char *from, char *to)
{ for(; *from != '\0'; from++, to++)
    *to = *from;
  *to = '\0';
}

void main()
{ char *a = "I am a teacher.123456789";
  char b[80] = "You are a student.";
  cout << "string_a=" << a << endl;
  cout << "string_b=" << b << endl;
  copy_string(a, b);
  cout << "string_a=" << a << endl;
  cout << "string_b=" << b << endl;
}
```

```
string_a=I am a teacher.123456789
string_b=You are a student.

string_a=I am a teacher.123456789
string_b=I am a teacher.123456789
Press any key to continue_
```

注意：数组b要有足够的存储空间！

3. 字符指针变量与字符数组的分别

char *cp; 与 char str[20];

❖ 字符数组str由若干元素组成，每个元素放一个字符；而指针变量cp中只能存放一个地址值。

❖ char str[20]; str= "I love China!"; (×)

char str[20]= "....."; (✓)

char *cp; cp= "I love China!"; (✓)

❖ str是地址常量；cp是地址变量。

cp++; (✓)

str++; (×)

str+1; (✓)

❖ cp接受键入字符串时,必须先开辟存储空间。

例 char str[10];
cin>>str (✓)

而 char *cp;
cin>> cp; (×)

改为: char *cp, str[10];
cp=str;
cin>>cp; (✓)

4. 字符串与数组的关系

- 字符串用一维字符数组存放;
- 一维字符数组中若有一个元素的值为0, 则该数组可当字符串用;
- 字符数组具有一维数组的所有特点;
 - ◆ 数组名是指向数组首地址的地址常量;
 - ◆ 数组元素的引用方法可用指针法和下标法;
 - ◆ 数组名作函数参数是地址传递等;
- 区别
 - ◆ 存储格式: 字符串结束标志;
 - ◆ 赋值方式与初始化;
 - ◆ 输入输出方式: %s %c

```
char str[80];
scanf("%s",str);
printf("%s",str);
gets(str);
puts(str);
```

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};	(×)

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

例 对字符串排序（简单选择排序）

```
main()
{ void sort(char *name[],int n), print(char *name[],int n);
  char *name[]={"Follow me","BASIC",
    "Great Wall","FORTRAN","Computer "};
  int n=5;
  sort(name,n);
  print(name,n);
}
void 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;}
  }
}
```

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



Ch9_28.c

例 用二级指针处理字符串



Ch9_000.c

```
#define NULL 0
void main()
{
    char **p;
    char *name[]={"hello","good", "world","bye",""};
    p=name+1;
    printf("%o : %s  ", *p,*p);
    p+=2;
    while(**p!=NULL)
        printf("%s\n",*p++); ⇔ *(p++)右结合
}
```

用*p可输出地址(%o或%x),
也可用它输出字符串(%s)

运行结果:
644 : good bye

6.7 指针与函数

★ **函数指针**：函数被存放在内存中一片连续的存储单元内，其中排在**最前面**的那个**存储单元的地址**就是这个函数的地址，也叫**函数指针**，用**函数名**表示，它是一个地址**常量**。

★ 指向函数的指针变量

❖ 定义形式：**数据类型** **(*)指针变量名**;

如 `int (*p)();`

❖ 函数指针变量 **(*)** 不能省

❖ 函数调用形式：`c = max(a, b);` 与 `c = (*p)(a, b);`

❖ 对函数指针变量 `p` 指向的函数必须有**函数说明**

1. 用指向函数的指针变量调用函数



Ch9_23.c

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

```
main()
{ int max(int ,int), (*p)();
  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);
}
```

2. 用指向函数的指针变量作函数参数



Ch9_24. c

例 用函数指针变量作参数，求最大值、最小值和两数之和

```
void main()
{ int a,b,max(int,int),
  min(int,int),add(int,int);
  void process(int,int,int (*fun)());
  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 result;
  result=(*fun)(x,y);
  printf("%d\n",result);
}
```

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

3. 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)

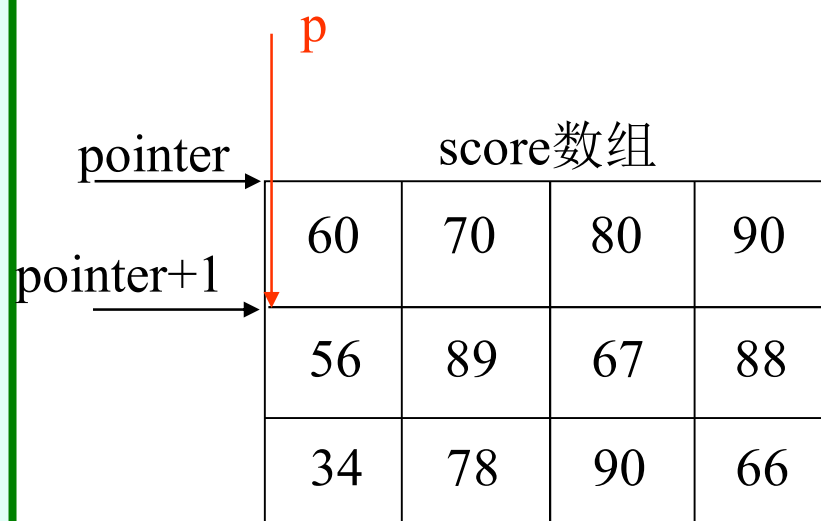
例 用指针函数实现: 有若干学生成绩, 要求输入学生序号后, 能输出其全部成绩。



Ch9_26.c

```
main()
{ float score[][4]={ {60,70,80,90},
                      {56,89,67,88},{34,78,90,66}};
  float *search(float (*pointer)[4],int n), *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("%5.2ft",*(p+i));
}

float *search(float (*pointer)[4], int n)
{ float *pt;
  pt=*(pointer+n);
  return(pt);
}
```



6.6 有关指针操作的小结

1. 指针变量是把其它变量的地址作为内容的变量。指针变量的内容可以是0、NULL和一个确定的地址。
2. 地址运算符(&) 返回其操作数的地址。
地址运算符的操作数必须是一个变量(或数组元素)。
3. 指针运算符(*) 又称为“间接引用运算符”，它表示从相应的存储单元中获取某种类型的数据值。
4. 指针±整数
5. 指针1—指针2
6. 指针的关系运算，如：指针1<指针2
7. 在调用带有参数的函数时，如果调用函数要求被调用函数修改参数的值，应该把参数的地址传递给被调用函数，被调用函数用间接引用运算符(*)修改调用函数中的参数的值。

指针的数据类型

定义	含义
<code>int i;</code>	定义整型变量i
<code>int *p;</code>	p为指向整型数据的指针变量
<code>int a[n];</code>	定义含n个元素的整型数组a
<code>int *p[n];</code>	n个指向整型数据的指针变量组成的指针数组p
<code>int (*p)[n];</code>	p为指向含n个元素的一维整型数组的指针变量
<code>int f();</code>	f为返回整型数的函数
<code>int *p();</code>	p为返回指针的函数，该指针指向一个整型数据
<code>int (*p)();</code>	p为指向函数的指针变量，该函数返回整型数
<code>int **p;</code>	p为指针变量，它指向一个指向整型数据的指针变量

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);`



指向函数的指针，函数返回`int`型指针

(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

