

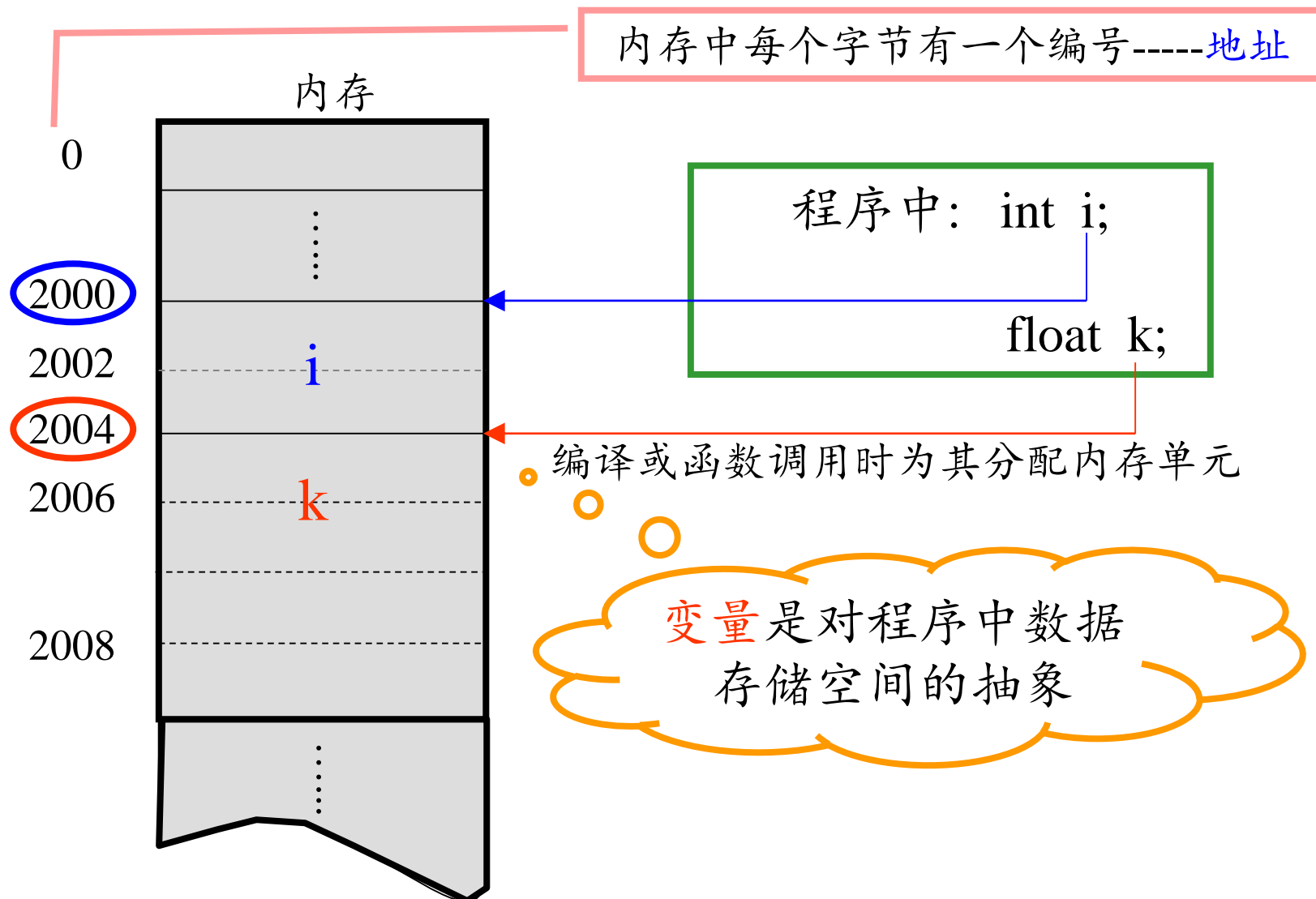
第六章 指 针

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
- 得到多于一个的函数返回值
-

§ 6.1 指针的概念

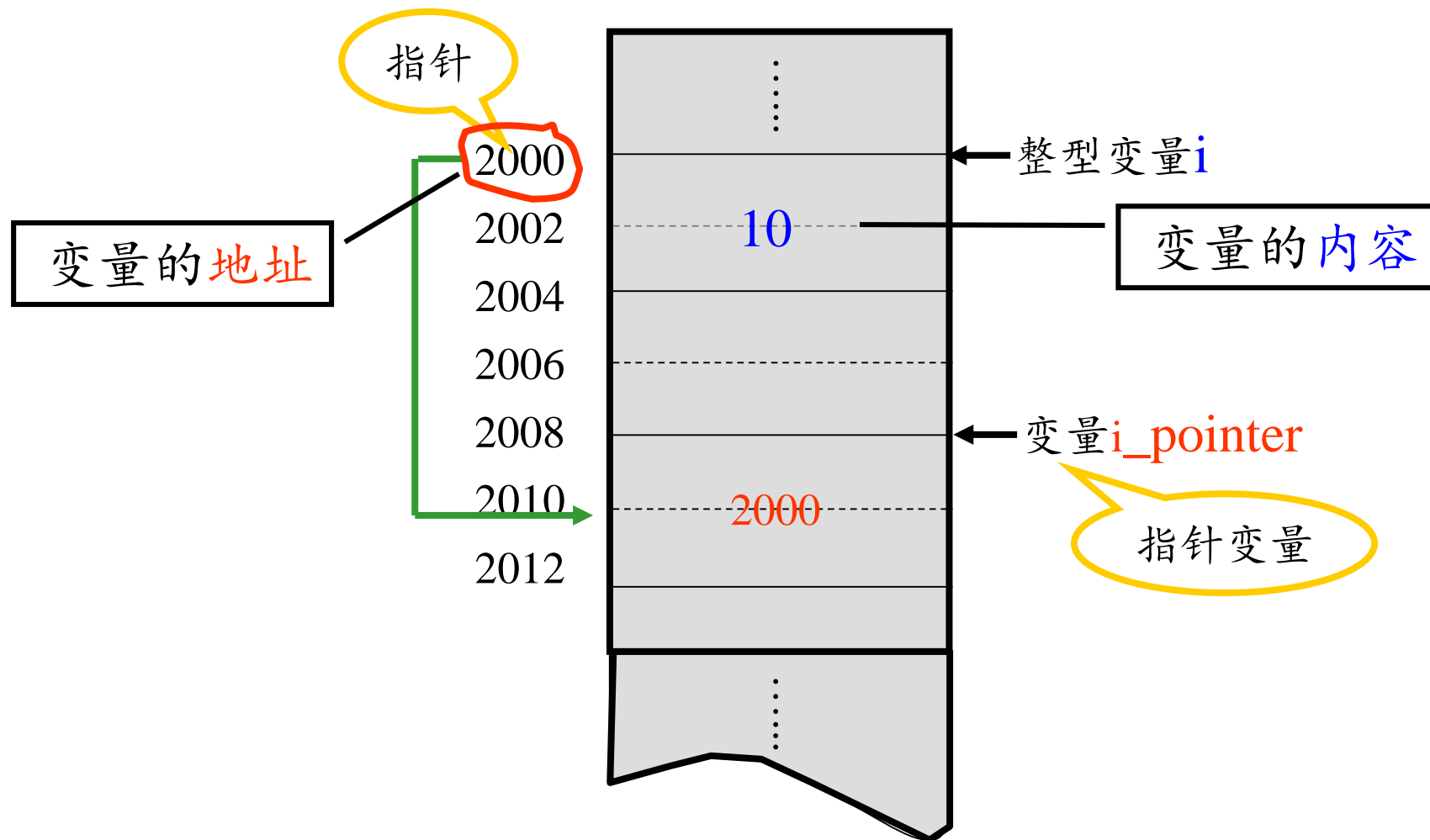
★ 变量与地址



★ 指针与指针变量

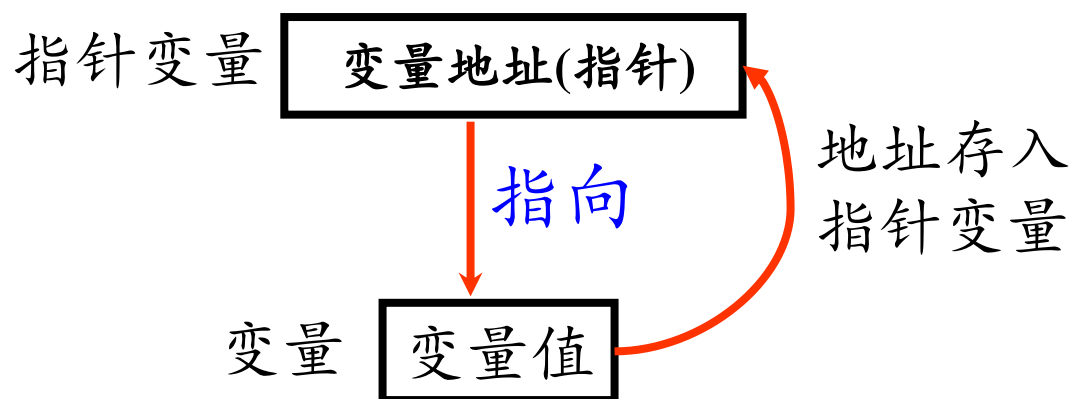
❖ 指针：一个变量的地址

❖ 指针变量：专门存放变量地址的变量叫指针变量



★ 指针与指针变量

- ❖ 指针：一个变量的地址
- ❖ 指针变量：专门存放变量地址的变量叫指针变量



★ 运算符: & 与 *

❖ 含义

含义: 取变量的地址

单目运算符

优先级: 3

结合性: 自右向左

含义: 取指针所指向变量的内容

单目运算符

优先级: 3

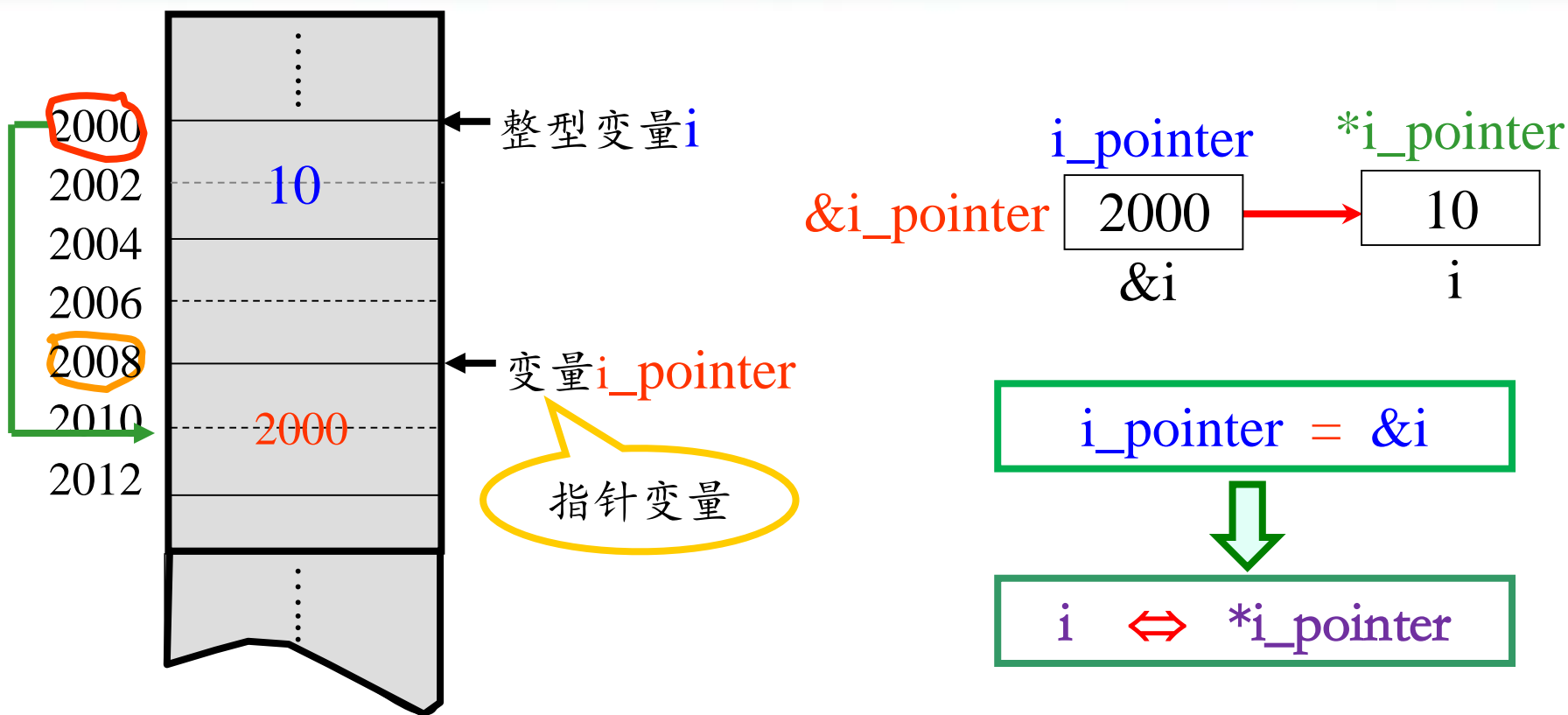
结合性: 自右向左

❖ 两者关系: 互为逆运算

例: `int k;`

`&k`: 变量k的地址

`*&k`: 等价于 `k`

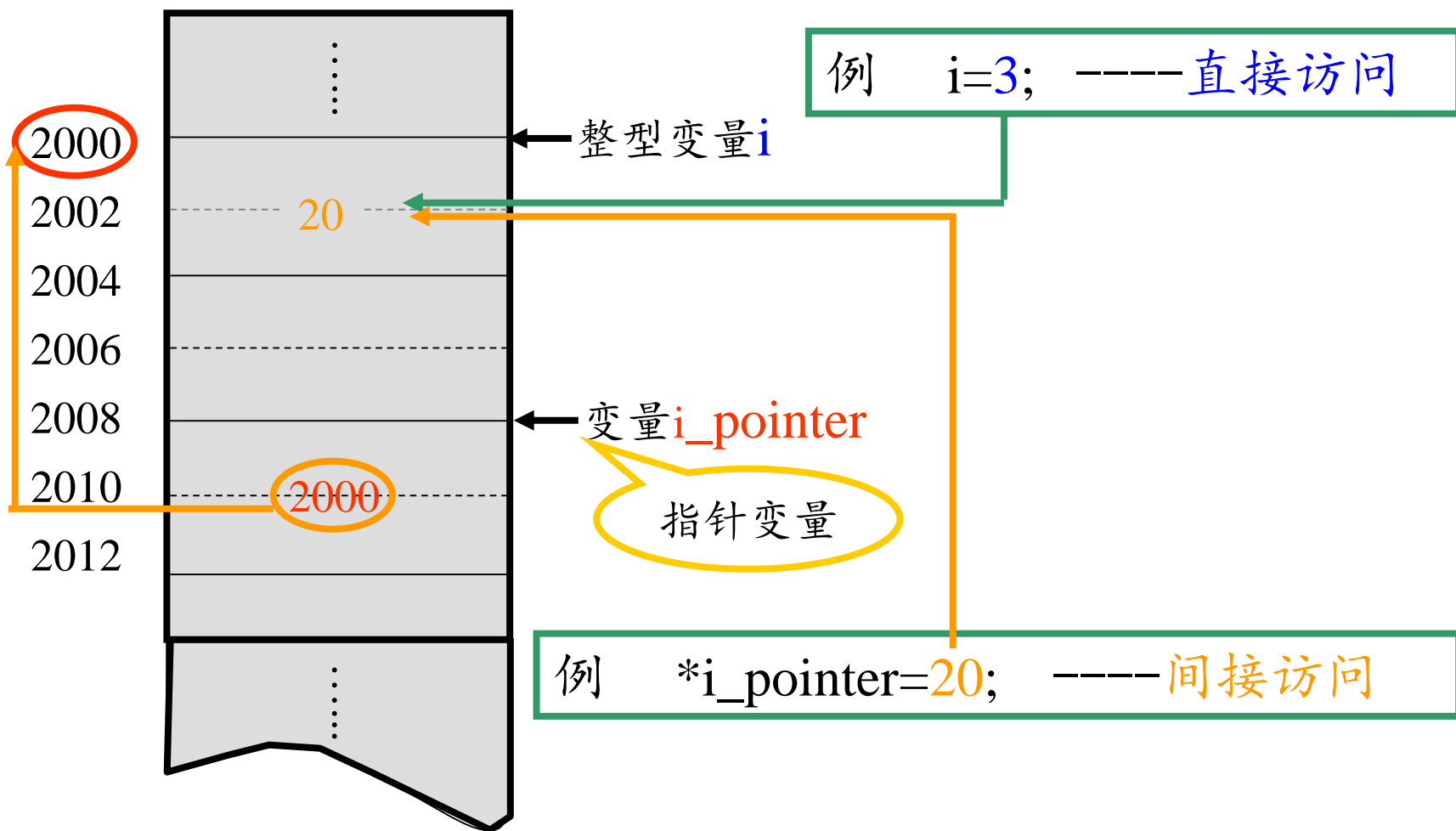


$i_pointer$ ——指针变量，它的内容是地址量
 $*i_pointer$ ——指针的**目标变量**，它的内容是数据
 $\&i_pointer$ ——指针变量占用内存的地址

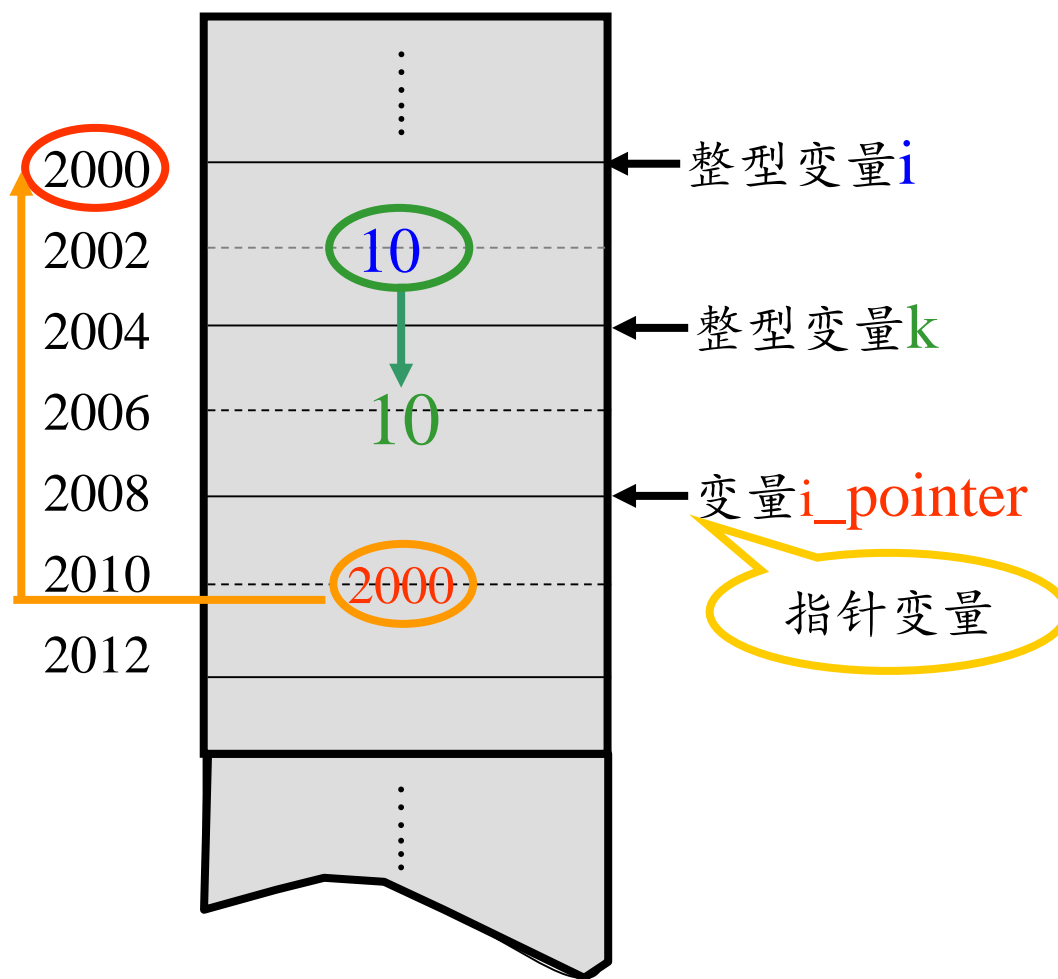
★ 直接访问与间接访问

❖ 直接访问：按变量存取变量值

❖ 间接访问：通过存放变量地址的变量去访问变量



例	<code>k=i;</code>	-- 直接访问
	<code>k=*i_pointer;</code>	-- 间接访问



§ 6.2 指针变量

★ 指针变量的定义

❖ 一般形式： [存储类型] 数据类型 *指针名；

指针变量本身的
存储类型

表示定义
指针变量

合法标
识符

例 `int *p1, *p2;`

指针的目标变量
的数据类型

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, 变量值不确定, 应用前必须先赋值

★ 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;

赋给指针变量，
不是赋给目标变量

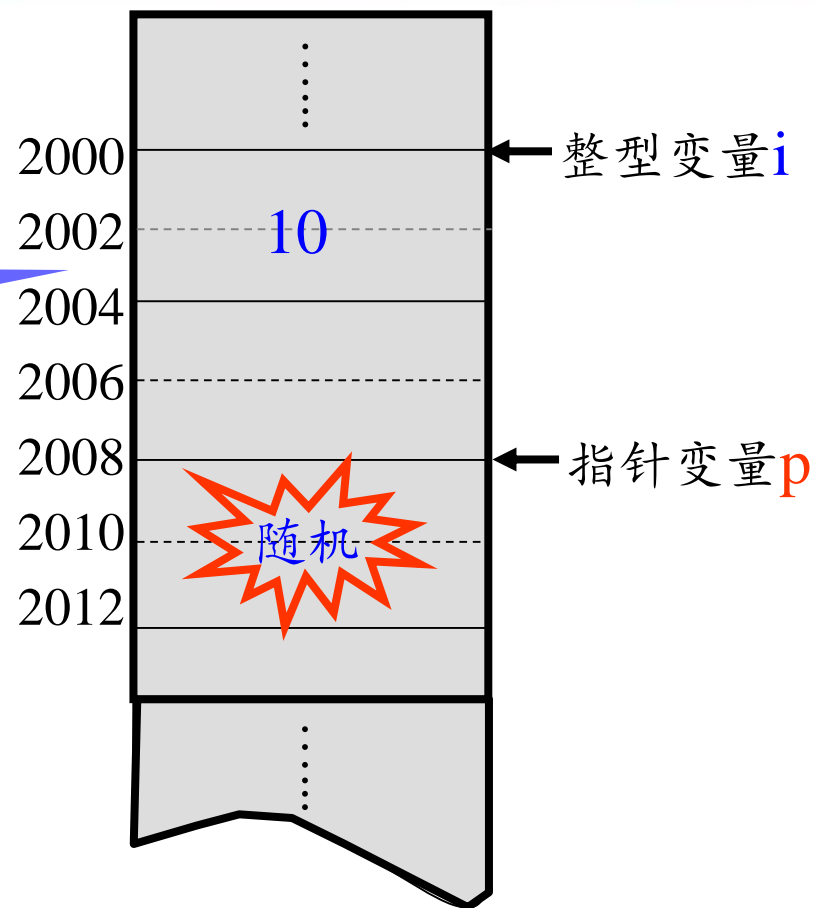
例 int i;
 int *p=&i;
 int *q=p;

变量必须已说明过
类型应一致

用已初始化指针变
量作初值

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

危险!

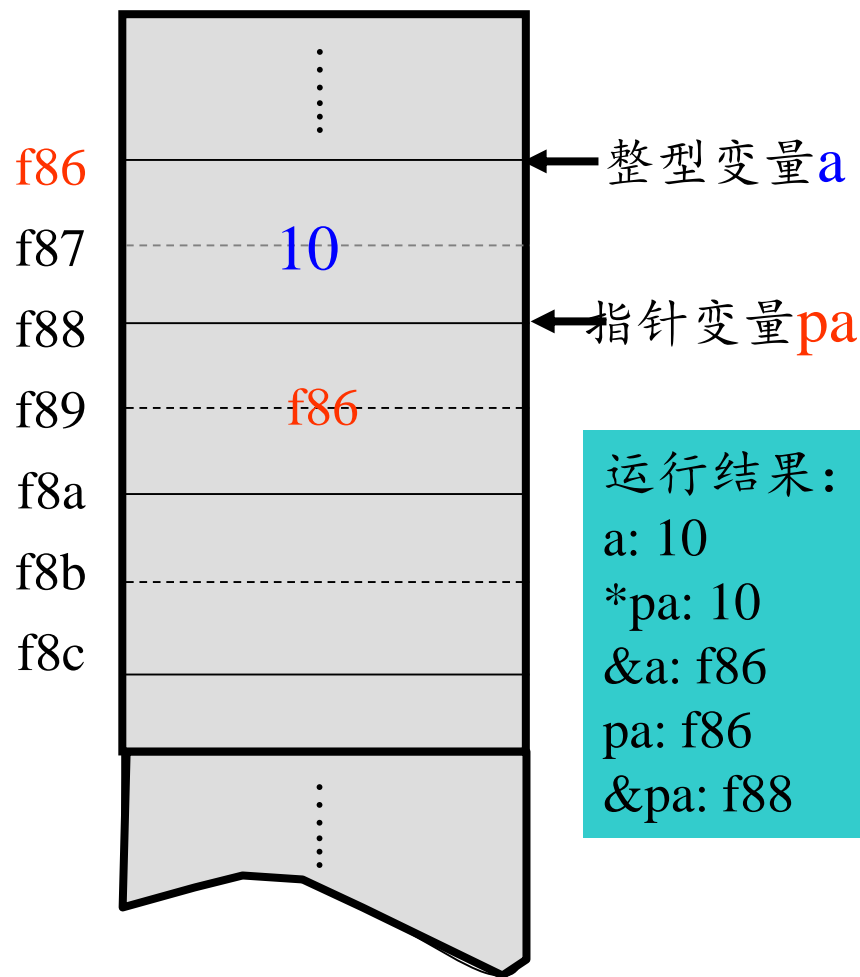


指针变量必须先赋值, 再使用

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

例 指针的概念

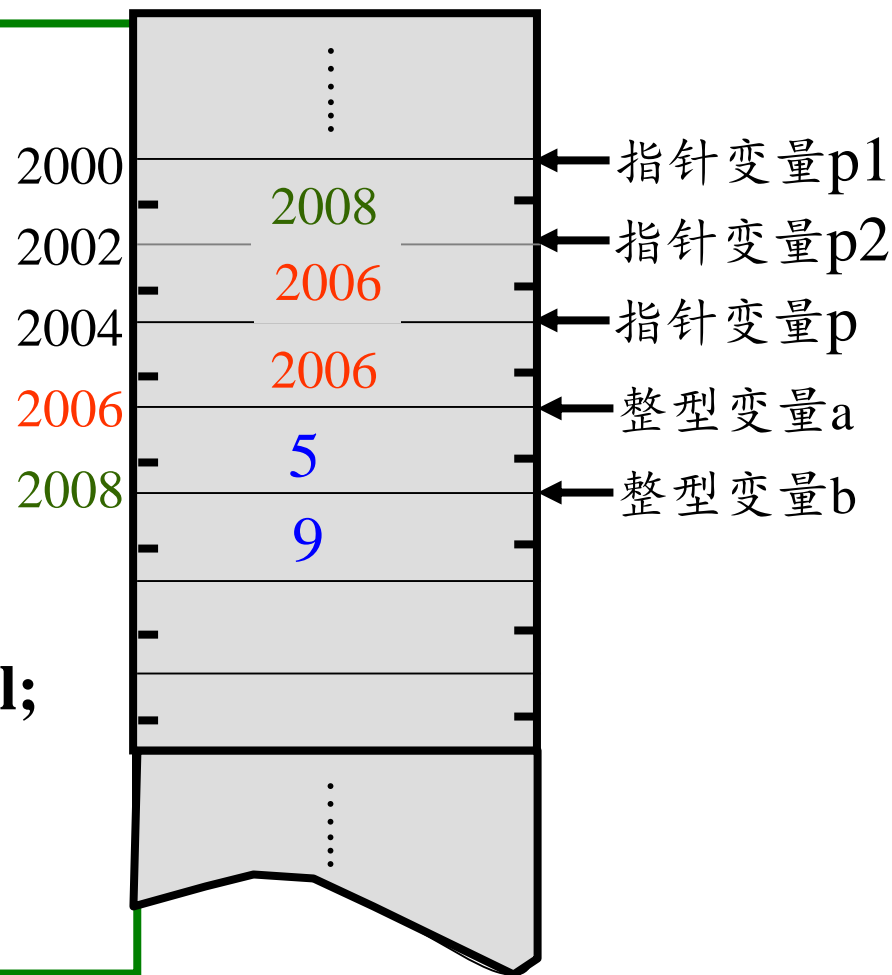
```
#include <iostream>
using namespace std;
void main()
{   int a;
    int *pa=&a;
    a=10;
    cout<<"a:"<<a;
    cout<<"*pa:"<<*pa;
    cout<<"&a:"<<&a;
    cout<<"pa:"<<pa;
    cout<<"&pa:"<<&pa;
}
```



例 输入两个数，并使其从大到小输出

```
#include <iostream>
using namespace std;
void main()
{ int *p1,*p2,*p,a,b;
  cin>>a>>b;
  p1=&a; p2=&b;
  if(a<b)
  { p=p1; p1=p2; p2=p;}
  cout<<"a="<<a<<" ,b="<<b<<endl;
  cout<<"max="<<*p1<<endl;
  cout<<"min="<<*p2<<endl;
}
```

运行结果： a=5,b=9
max=9
min=5

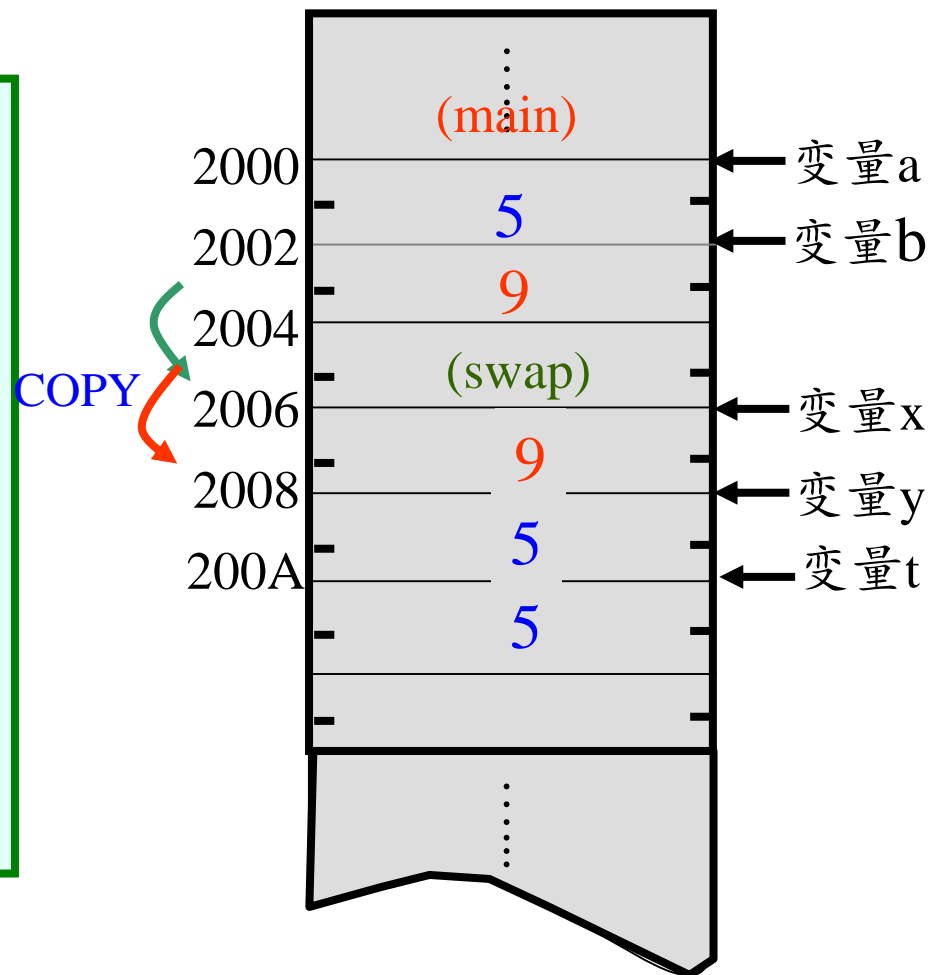


★ 指针变量作为函数参数——地址传递

特点：共享内存，“双向”传递

例 将数从大到小输出

```
#include <iostream>
using namespace std;
void swap(int x,int y)
{ int t; t=x; x=y; y=t; }
void main()
{ int a,b;
  cin>>a>>b;
  if(a<b) swap(a,b);
  cout<<a<<","<<b<<endl;
}
```



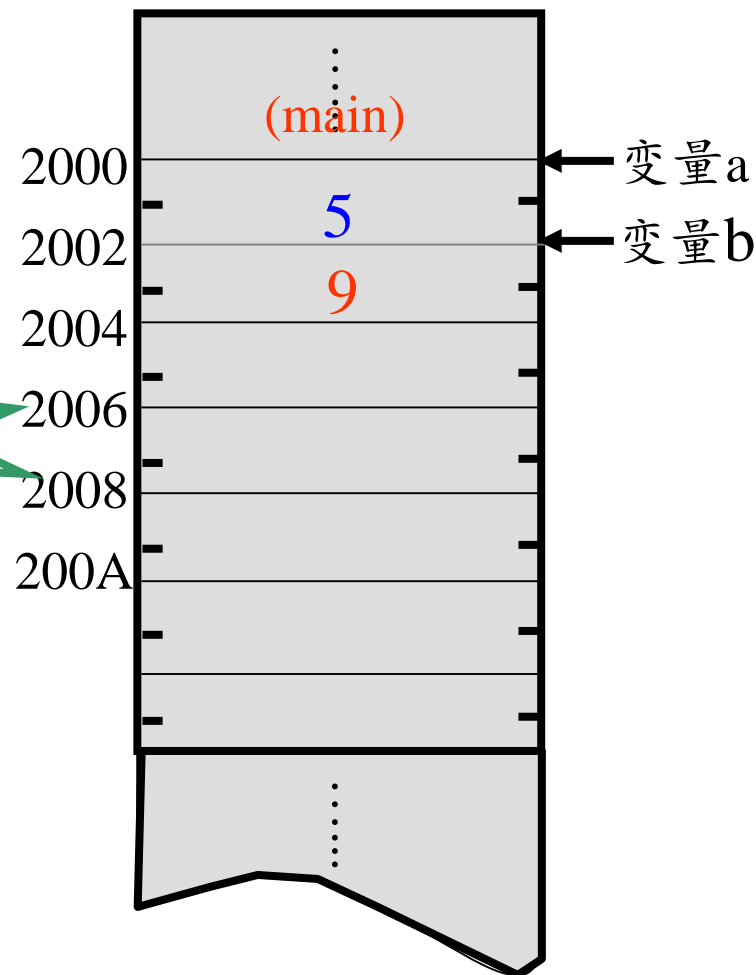
★ 指针变量作为函数参数——地址传递

特点：共享内存，“双向”传递

例 将数从大到小输出

```
#include <iostream>
using namespace std;
void swap(int x,int y)
{ int t; t=x; x=y; y=t; }
void main()
{ int a,b;
  cin>>a>>b;
  if(a<b) swap(a,b);
  cout<<a<<","<<b<<endl;
}
```

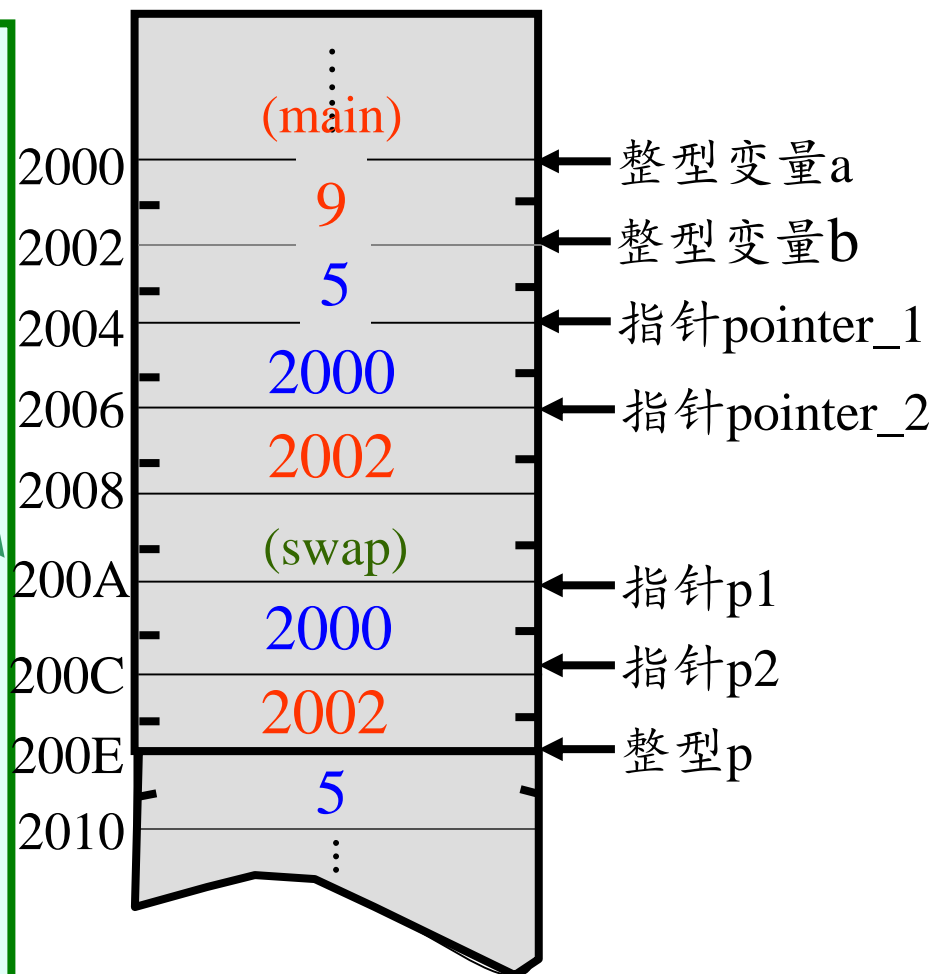
运行结果：5, 9



例 将数从大到小输出

```
#include <iostream>
void swap(int *p1, int *p2)
{ int p; p=*p1; *p1=*p2; *p2=p; }
void 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<<endl;
}
```

COPY

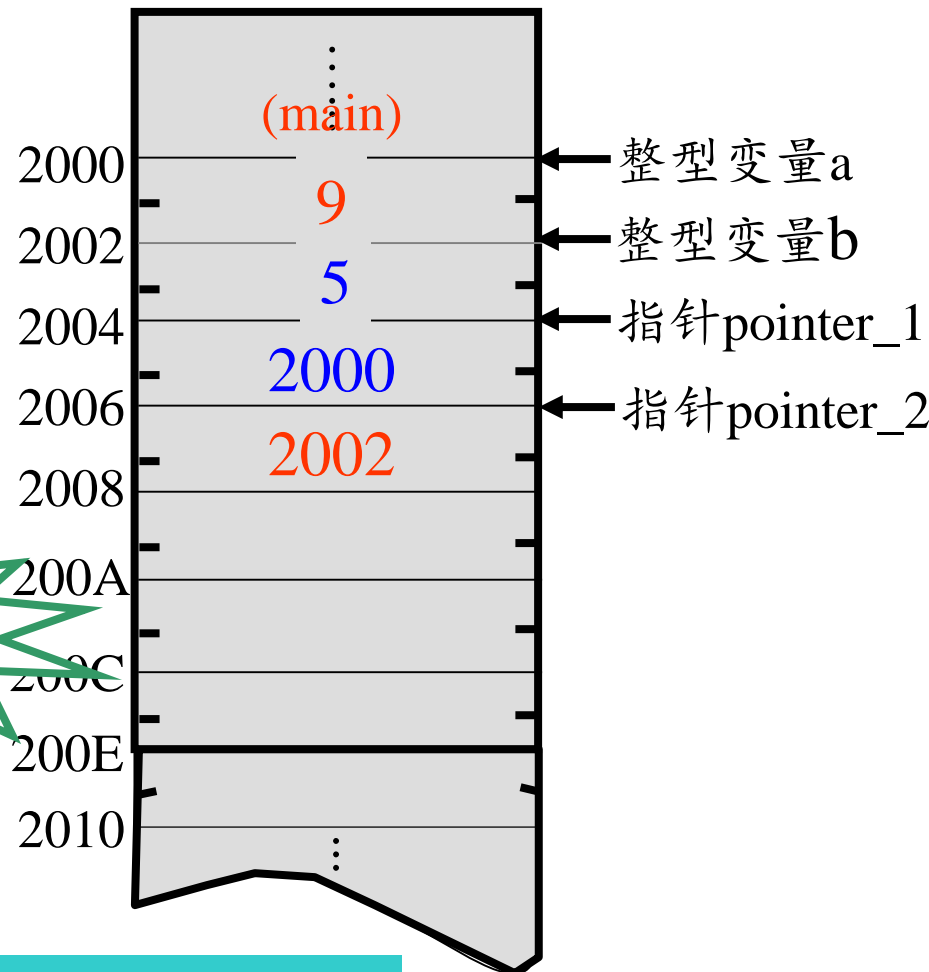


例 将数从大到小输出

共享内存,“双向”传递

```
#include <iostream>
void swap(int *p1, int *p2)
{ int p; p=*p1; *p1=*p2; *p2=p; }
void 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<<endl;
}
```

地址传递



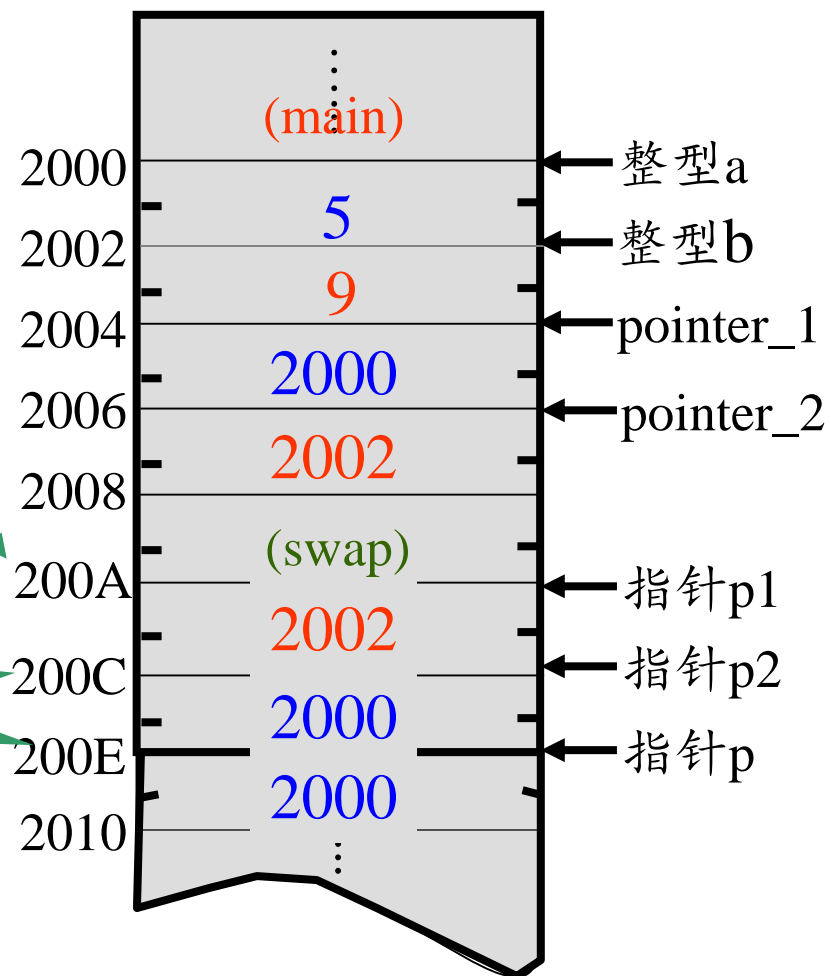
运行结果: 9, 5

例 将数从大到小输出

```
#include <iostream>
void swap(int *p1, int *p2)
{ int *p; p=p1; p1=p2; p2=p;}
void 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;
}
```

COPY

地址传递



运行结果：5， 9

★ 零指针与空类型指针

❖ 零指针：(空指针)

- 定义：指针变量值为零

- 表示： `int *p=0;`

```
#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

`p` 指向地址为 0 的单元，
系统保证该单元不作它用
表示指针变量值没有意义

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

★ 零指针与空类型指针

❖ 零指针：(空指针)

- 定义：指针变量值为零

- 表示： `int * p=0;`

❖ void * 类型指针

- 表示： `void *p;`

- 使用时要进行强制类型转换

表示不指定p是指向哪一种类型数据的指针变量

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

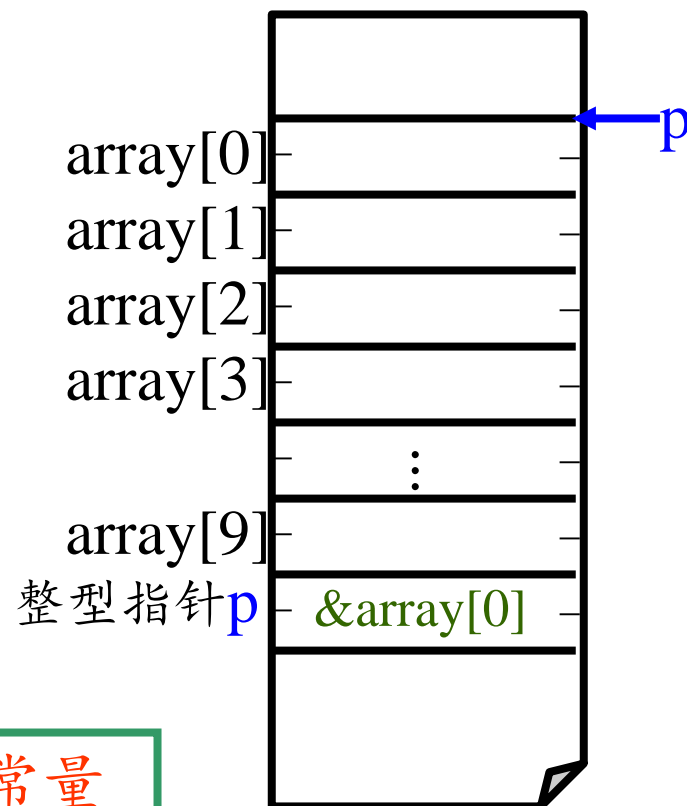
§ 6.3 指针与数组

★ 指向数组元素的指针变量

```
例  int array[10];  
    int *p;  
    p=&array[0]; //⇔ p=array;
```

```
或  int *p=&array[0];  
或  int *p=array;
```

数组名是表示数组**首地址**的**地址常量**



★ 指针的运算

❖ 指针变量的赋值运算

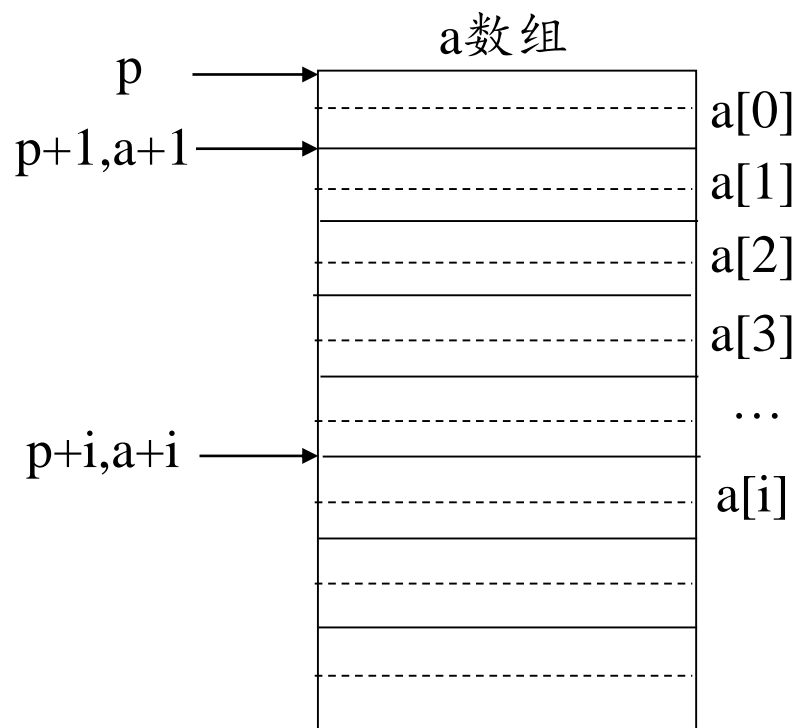
- $p = \&a;$; (将变量a地址 \Rightarrow p)
- $p = \text{array};$ (将数组array首地址 \Rightarrow p)
- $p = \&\text{array}[i];$ (将数组元素地址 \Rightarrow p)
- $p1 = p2;$ (指针变量p2值 \Rightarrow p1)
- 不能把一个整数 \Rightarrow p,也不能把p的值 \Rightarrow 整型变量

如 `int i, *p;`
`p=1000;` (×)
`i=p;` (×)

分别都有Warning

❖ 指针的算术运算:

- $p \pm i$ 值为 $p \pm i \times d$ (i 为整型数, d 为 p 指向的变量所占字节数)

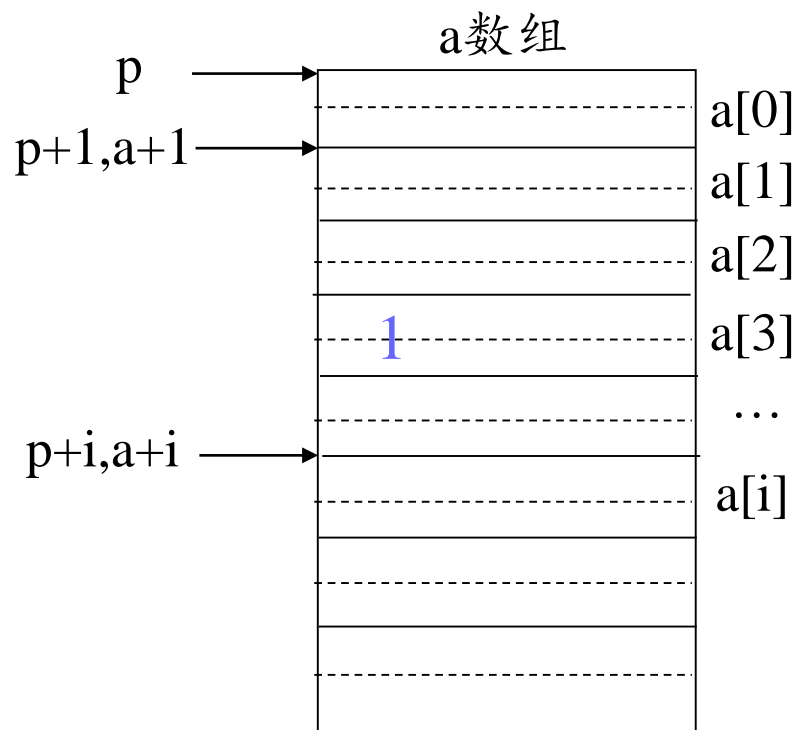


- $p++$, $p--$, $p+i$, $p-i$, $p+=i$, $p-=i$ 等
- 若 $p1$ 与 $p2$ 指向同一数组,
 $p1-p2$ =两指针间元素个数 值为 $(p1-p2)/d$
- $p1+p2$ 无意义

例 p 指向 `int` 型数组, 且 $p=\&a[0]$;
则 $p+1$ 指向 $a[1]$

例 `int a[10], *p=&a[2];`
`p++;`
`*p=1;`

例 `int a[10];`
`int *p1=&a[2], *p2=&a[5];`
则: $p2-p1=3$;



❖ 指针变量的关系运算

- 若 $p1$ 和 $p2$ 指向同一数组，则

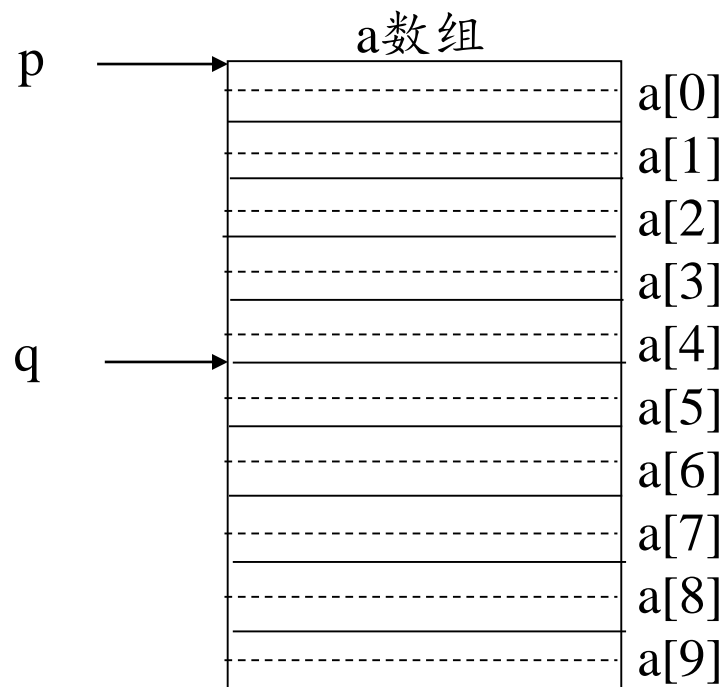
- ◆ $p1 < p2$ 表示 $p1$ 指的元素在前

- ◆ $p1 > p2$ 表示 $p1$ 指的元素在后

- ◆ $p1 == p2$ 表示 $p1$ 与 $p2$ 指向同一元素

- 若 $p1$ 与 $p2$ 不指向同一数组，比较无意义

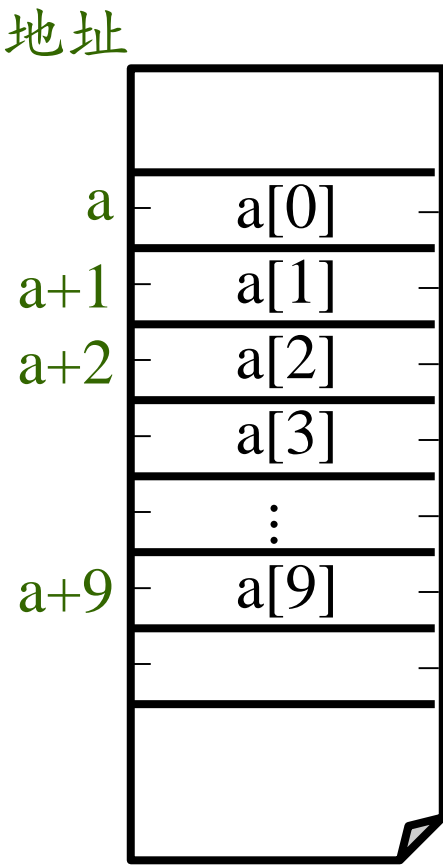
- $p == \text{NULL}$ 或 $p != \text{NULL}$



★数组元素表示方法

`[]` 变址运算符
`a[i]` \Leftrightarrow `*(a+i)`

`int *p=a;`



元素

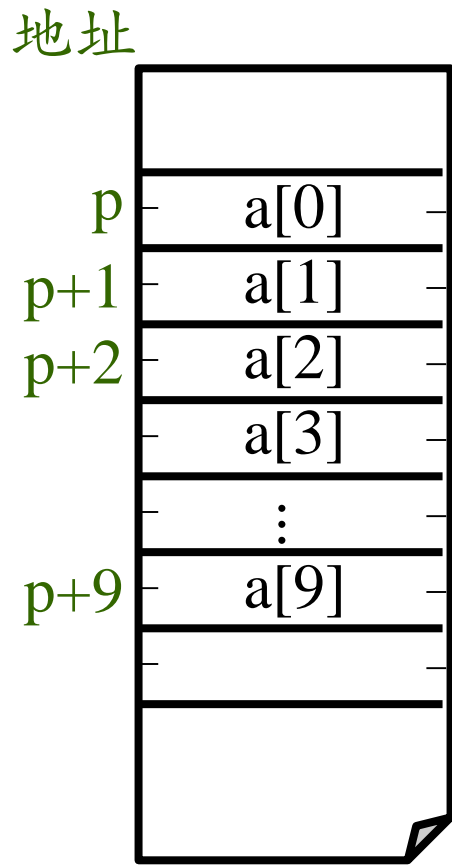
a[0] *a

a[1] *(a+1)

a[2] *(a+2)

⋮

a[9] *(a+9)



元素

*p p[0]

*(p+1) p[1]

*(p+2) p[2]

⋮

*(p+9) p[9]

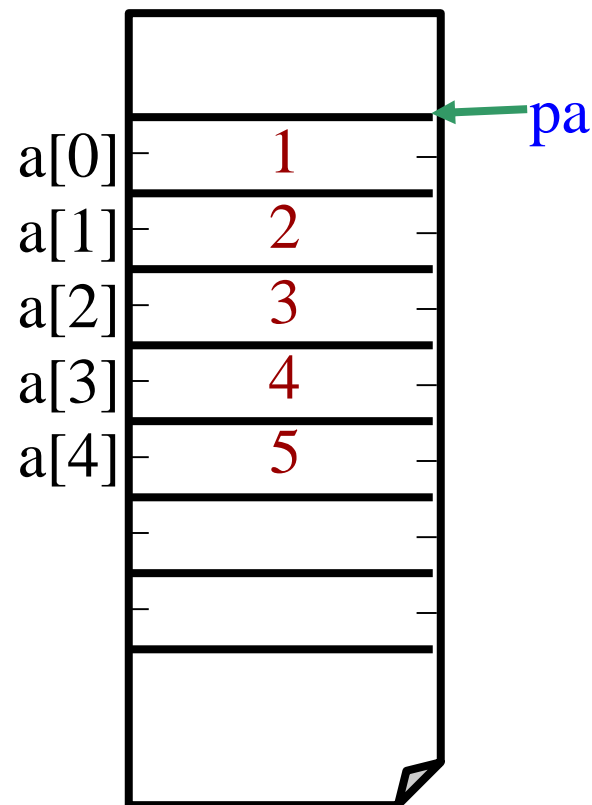
下标法

指针法

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

例 数组元素的引用方法

```
#include <iostream>
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<<i<<": ",*(pa+i)<<endl;
    for(i=0;i<5;i++)
        cout<<i<<": ",*(a+i)<<endl;
    for(i=0;i<5;i++)
        cout<<i<<": ", pa[i]<<endl;
    for(i=0;i<5;i++)
        cout<<i<<": ", a[i]<<endl;
}
```



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

数组元素地址的正确表示:

(A) `&(a+1)` (B) `a++` (C) `&p` (D) `&p[i]`

数组名是地址常量

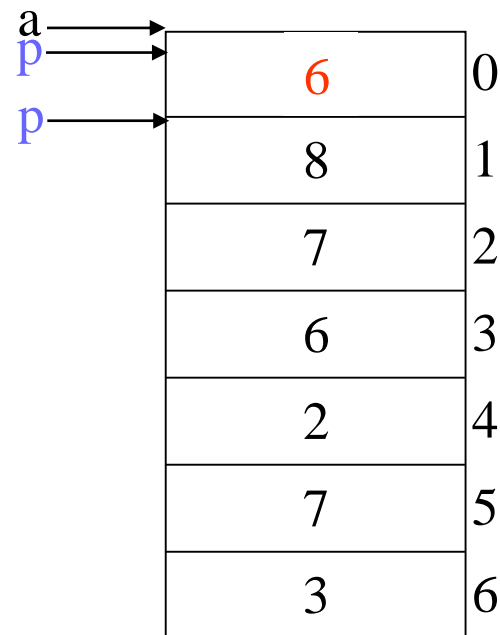
`p++,p--` (✓)

`a++,a--` (✗)

`a+1, *(a+2)` (✓)

例 注意指针变量的运算

```
例 void main()
{   int a[]={5,8,7,6,2,7,3};
    int y, *p=&a[1];
    y=(*--p)++;
    cout<<y<<endl;
    cout<<a[0]<<endl;
}
```



输出： 5
6

★数组名作函数参数

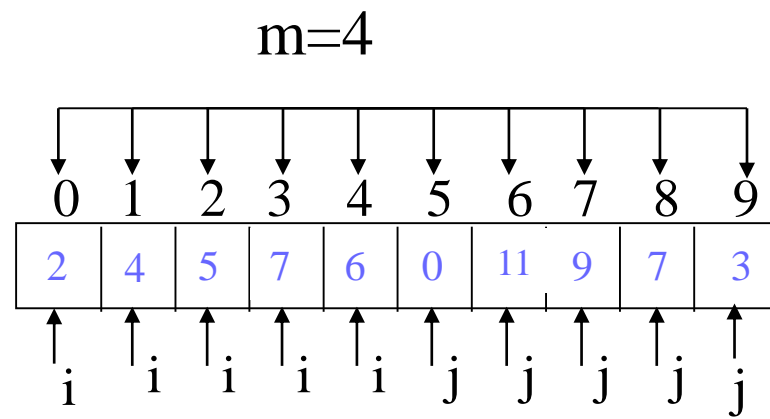
- ❖数组名作函数参数，是地址传递
- ❖数组名作函数参数，实参与形参的对应关系

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; }
}

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



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

```
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; }
}
```

```
void main()
```

```
{  int i,a[10]={3,7,9,11,0,6,7,5,4,2};
```

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

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

```
    for(i=0;i<10;i++)
```

```
        cout<<a[i] <<" ";
```

```
    cout<<endl;
```

```
}
```

		a数组		
x	i	3	2	a[0]
	i	7	4	a[1]
	i	9	5	a[2]
	i	11	7	a[3]
p=x+m	i	0	6	a[4]
	j	6	0	a[5]
	j	7	11	a[6]
	j	5	9	a[7]
	j	4	7	a[8]
	j	2	3	a[9]

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

```
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; }  
}
```

```
void 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:";  
    for(p=a;p<a+10;p++)  
        cout<<*p<<" ";  
}
```

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

```
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; }
}

void 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:";
    for(p=arr;p<arr+10;p++)
        cout<<*p<<" ";
}
```

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 $\text{int } q[] \Leftrightarrow \text{int } *q$
- 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 `4*10` 字节的内存区

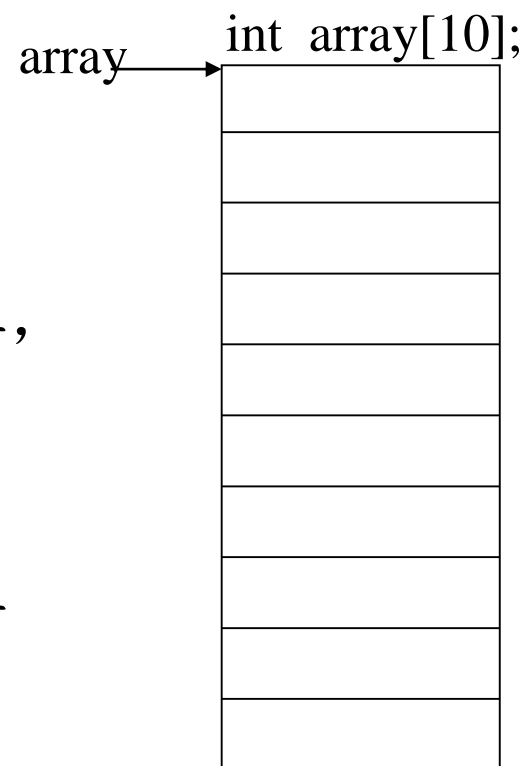
讲解作业P188: 第一题

★ 指针与二维数组

❖ 二维数组的地址

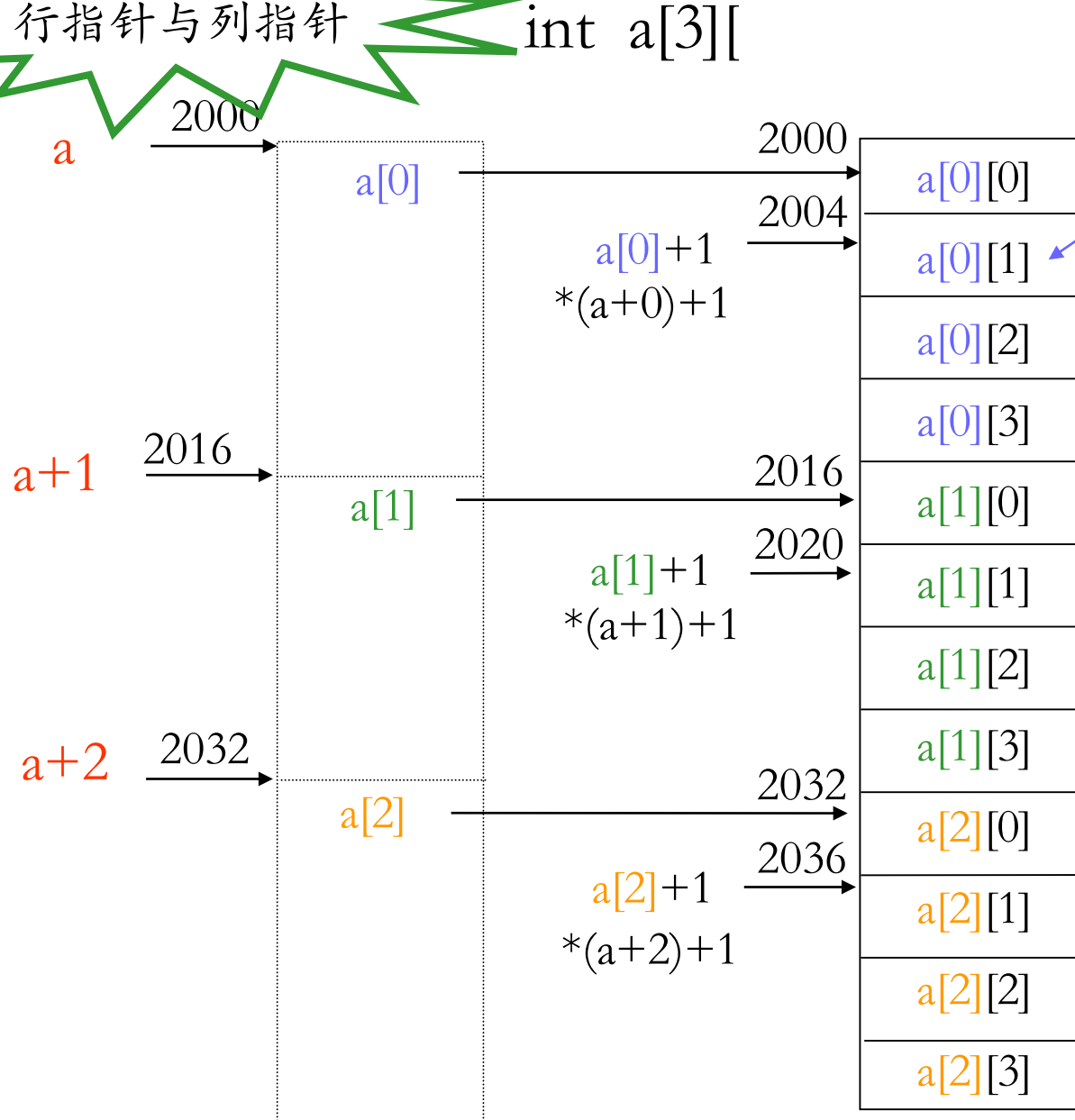
对于一维数组：

- (1) 数组名`array`表示数组的首地址，
即`array[0]`的地址；
- (2) 数组名`array`是地址常量
- (3) `array+i`是元素`array[i]`的地址
- (4) `array[i] ⇔ *(array+i)`



行指针与列指针

int a[3][

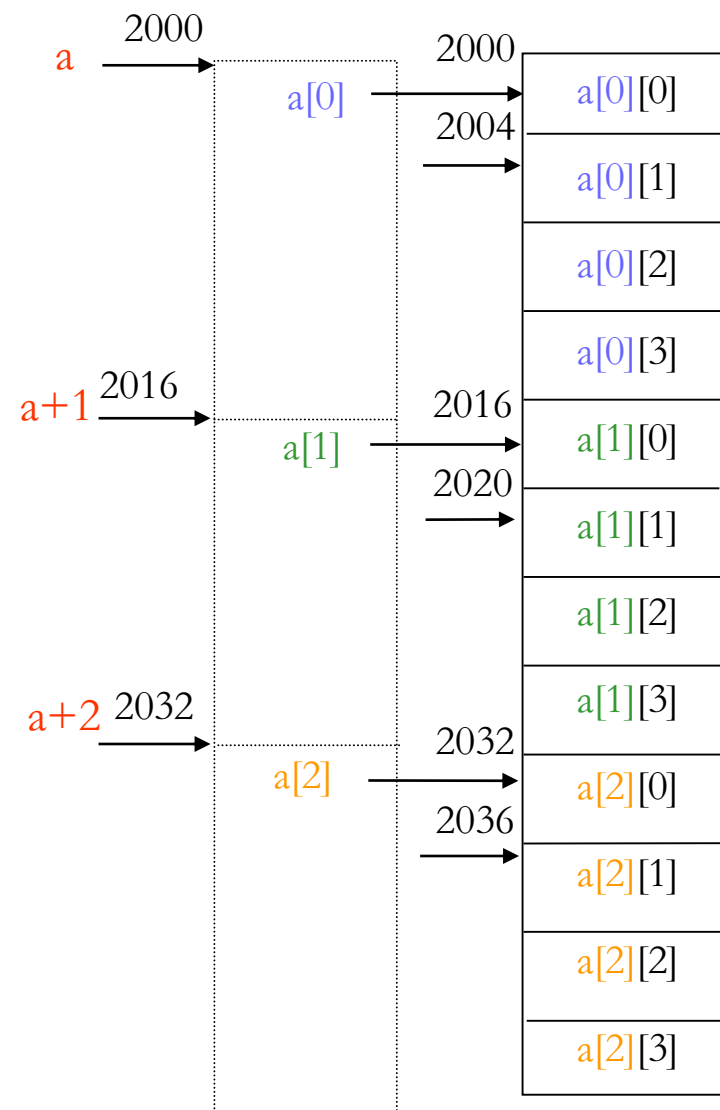
 $*(a[0]+1)$ $*(*(a+0)+1)$

对于二维数组:

(1) `a`是数组名,
包含三个元素
`a[0]`,`a[1]`,`a[2]`

(2) 每个元素`a[i]`
又是一个一维
数组, 包含4个
元素

```
int a[3][4];
```



● 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

◆ $a[i] \Leftrightarrow *(a+i)$ ----- 第*i*行第0列的元素地址

◆ $a[i]+j \Leftrightarrow *(a+i)+j$ ----- 第*i*行第*j*列的元素地址

◆ $*(a[i]+j) \Leftrightarrow *((a+i)+j) \Leftrightarrow a[i][j]$

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

◆ $a+i \Leftrightarrow \&a[i]$, 表示第*i*行首地址, 指向行

◆ $a[i] \Leftrightarrow *(a+i) \Leftrightarrow \&a[i][0]$, 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) (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)

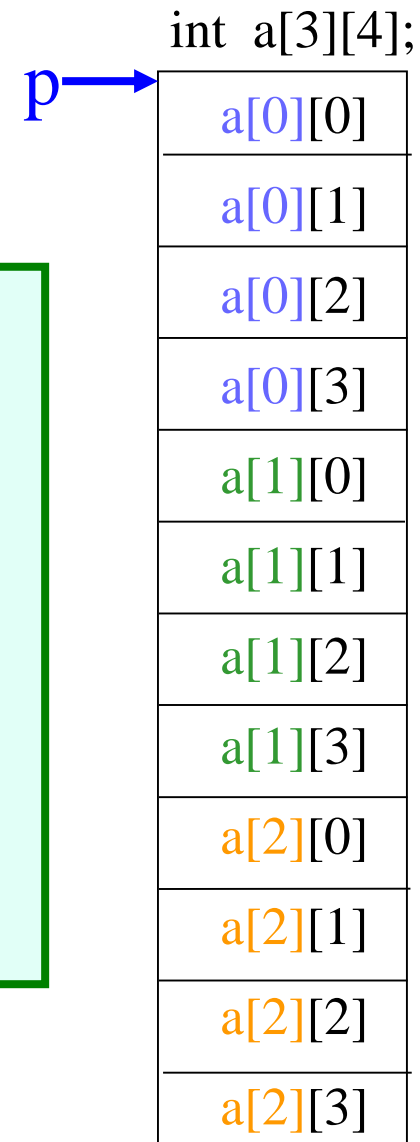
❖ 二维数组的指针变量

● 指向二维数组元素的指针变量

例 指向二维数组元素的指针变量

```
void 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[0])%4==0) cout<<"\n";
    cout<<*p<<" ";
  }
}
```

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



● 指向一维数组的指针变量

() 不能少, `int (*p)[4]` 与 `int *p[4]` 不同

◆ 定义形式: **数据类型 (*指针名)[一维数组维数];**

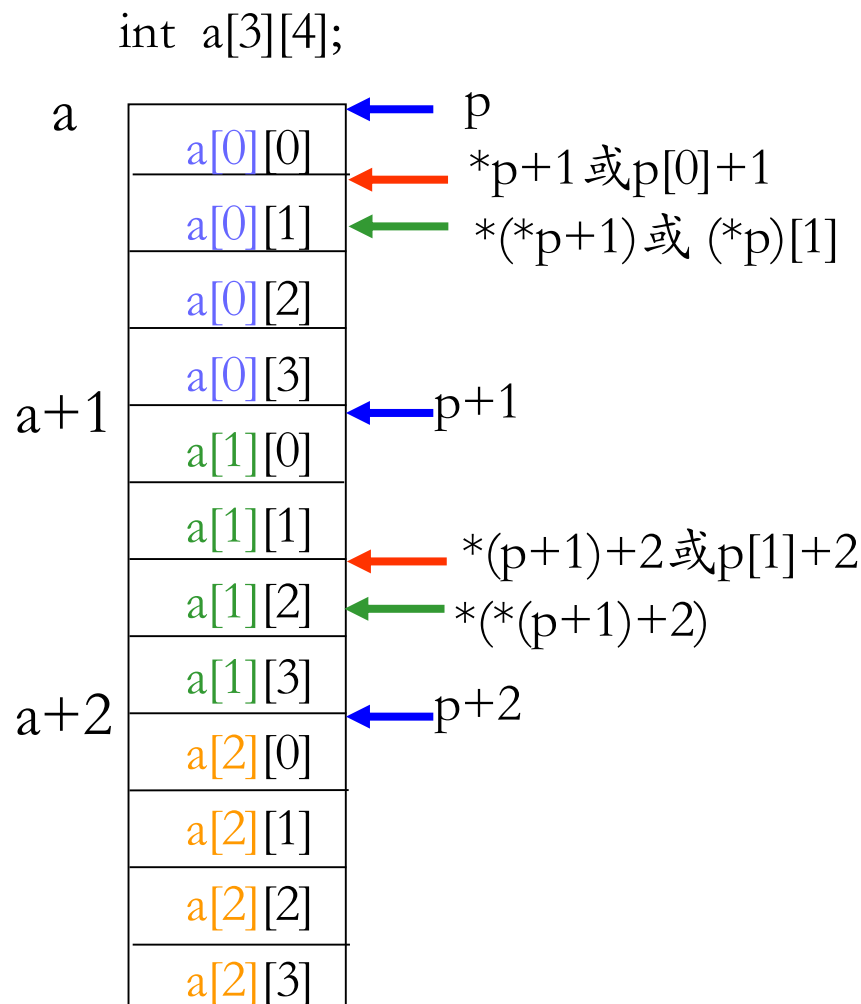
例 `int (*p)[4];`

p 的值是一维数组的首地址, p 是行指针。 `p=a;`

◆ 可让 p 指向二维数组某一行

如 `int a[3][4], (*p)[4]=a;`

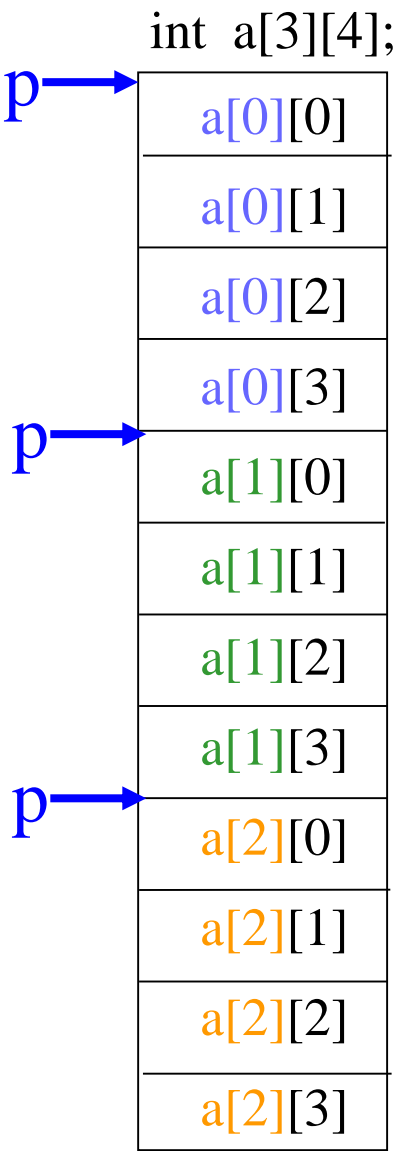
一维数组指针变量维数和二维数组列数必须相同



例 一维数组指针变量举例

```
void main()
{
    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(j=0;j<4;j++)
            cout<<*(p+j)<<" ";
    cout<<endl;
}
```

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



❖ 二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];`

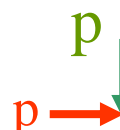
实参	形参
数组名a	数组名 <code>int x[][4]</code>
数组名a	指针变量 <code>int (*q)[4]</code>
指针变量p1	数组名 <code>int x[][4]</code>
指针变量p1	指针变量 <code>int (*q)[4]</code>
指针变量p2	指针变量 <code>int *q</code>

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

```
void 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



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<<aver;
}

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

float p[][4]

$\Leftrightarrow p[n][i]$

§ 6.4 指针与字符串

★ 字符串表示形式

❖ 用字符数组实现

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

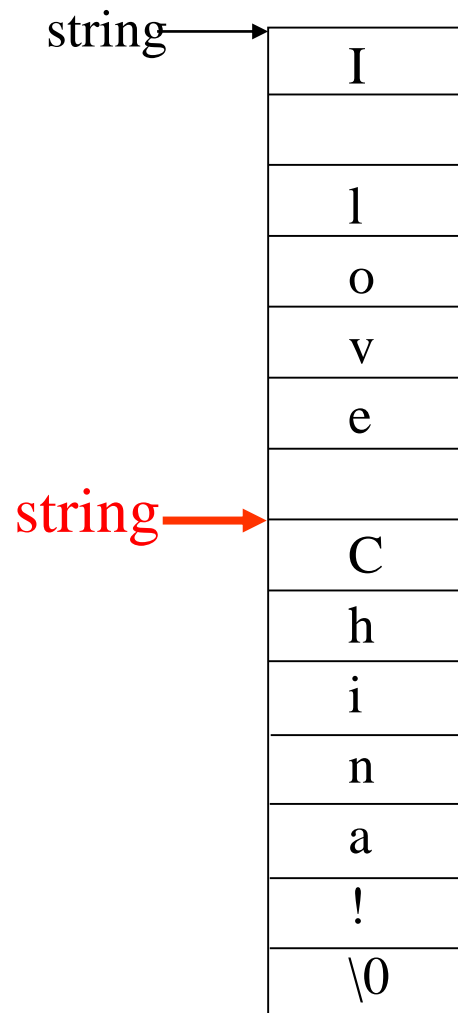
运行结果：
I love China!
China!

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]

❖ 用字符指针实现

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

运行结果：
I love China!
China!



★ 字符指针变量与字符数组

char *cp; 与 char str[20];

❖ 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

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

char *cp=str; cp="I love China!"; (✓)

❖ str是地址常量；cp是地址变量

❖ 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; (✓)

§ 6.5 指针与函数

★函数指针：函数在编译时被分配的入口地址,用函数名表示

★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)();`

()不能省
`int (*p)()` 与 `int *p()`不同

函数返回值
的数据类型

max →

指令1

指令2

⋮

❖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); ⇔ c=p(a,b);`

例 用函数指针变量调用函数，比较两个数大小

```
void main()
{  int max(int, int);
   int a,b,c;
   cin>>a>>b;
   c=max(a,b);
   cout<<a<<b<<c<<endl;
}

int max(int x,int y)
{  int z;
   if(x>y) z=x;
   else   z=y;
   return(z);
}
```

```
void main()
{  int max(int,int), (*p)(int, int);
   int a,b,c;
   p=max;
   cin>>a>>b;
   c=p(a,b);
   cout<<a<<b<<c<<endl;
}

int max(int x,int y)
{  int z;
   if(x>y) z=x;
   else   z=y;
   return(z);
}
```

★ 用函数指针变量作函数参数

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 x,int y);
  int min(int x,int y),add(int x,int y);
  void process(int x,int y,int (*fun)());
  cin>>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);
  cout<<result;
}
```

```
int max(int x,int y)
{ cout<<"max=";
  return(x>y?x:y);
}
```

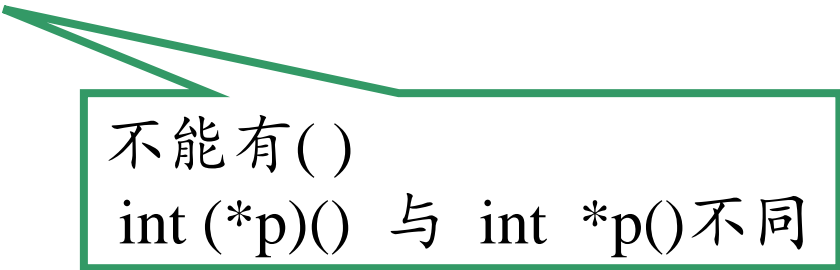
```
int min(int x,int y)
{ cout<<"min=";
  return(x<y?x:y);
}
```

```
int add(int x,int y)
{ cout<<"sum=";
  return(x+y);
}
```

§ 6.6 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);



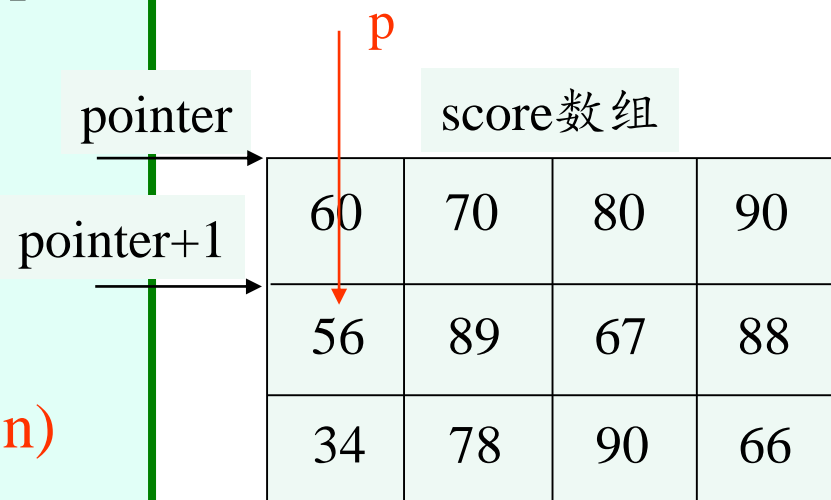
不能有()
int (*p)() 与 int *p()不同

例 `int *f(int x, int y)`

例 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。

```
void 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;   cin>>m;
  p=search(score,m);
  for(i=0;i<4;i++)
      cout<<*(p+i)<<'\t';
}

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



§ 6.7 指针数组和多级指针

用于处理二维数组或多个字符串

★ 指针数组

❖ 定义：数组中的元素为指针变量

指针本身的存储类型

❖ 定义形式：[存储类型] 数据类型 *数组名[数组长度说明];

例 `int *p[4];`

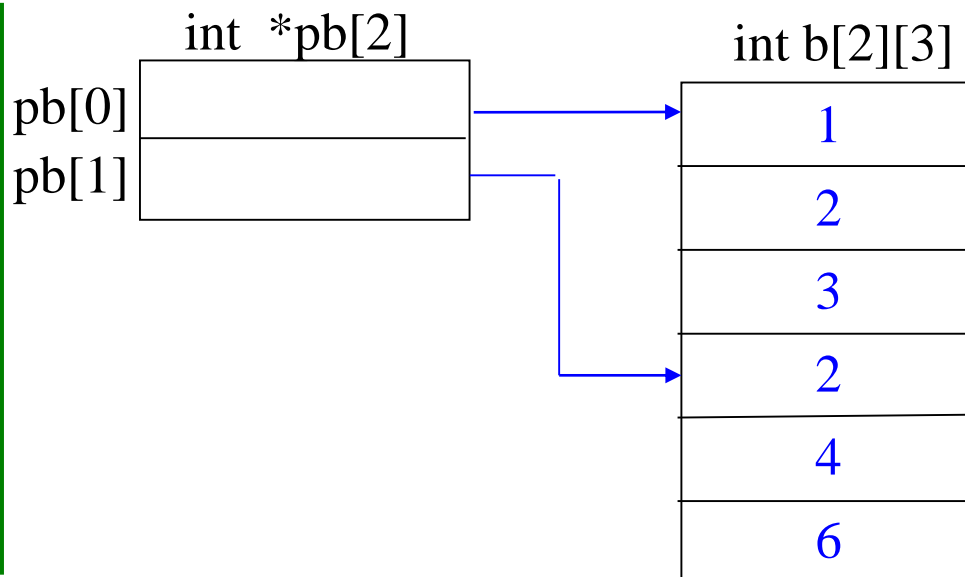
指针所指向变量的数据类型

区分 `int *p[4]` 与 `int (*p)[4]`

❖ 指针数组赋值与初始化

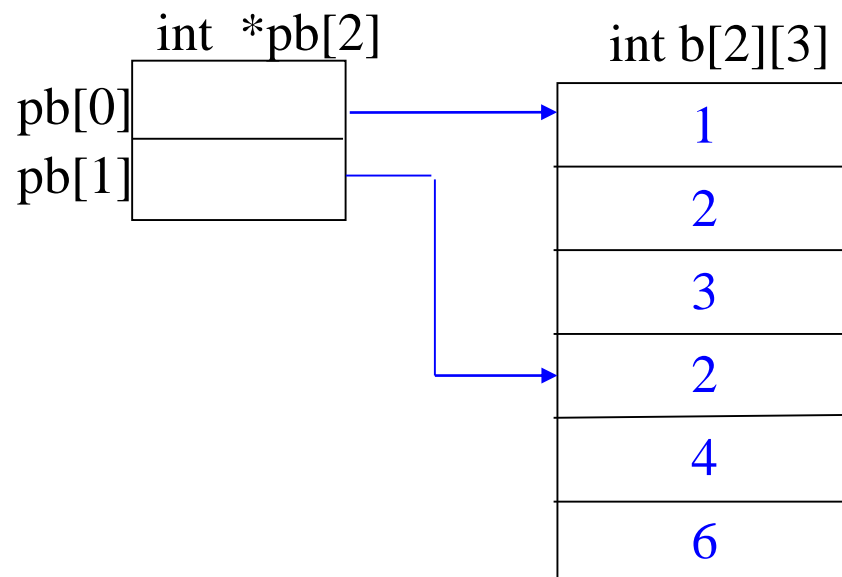
赋值:

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



初始化:

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



❖ 指针数组赋值与初始化

或：

```
void main()
```

```
{ char *p[4];
```

```
p[0]= "Fortran";
```

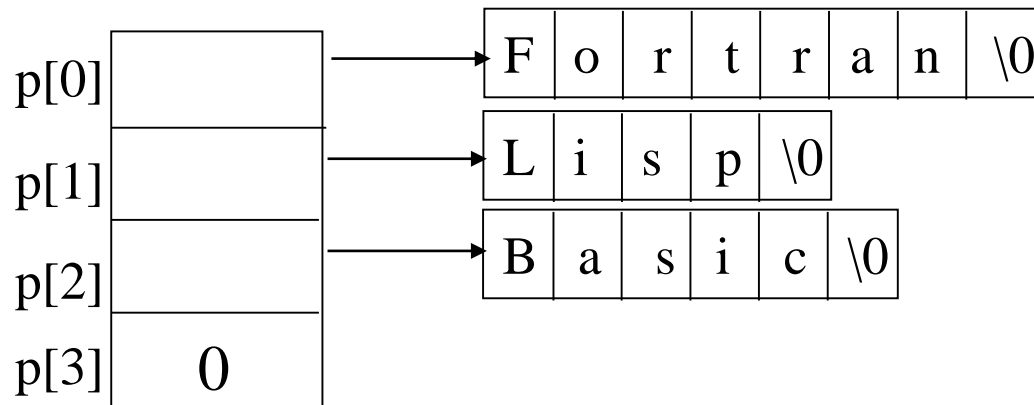
```
p[1]= "Lisp";
```

```
p[2]= "Basic";
```

```
p[3]=NULL;
```

• • • • •

}



❖ 指针数组赋值与初始化

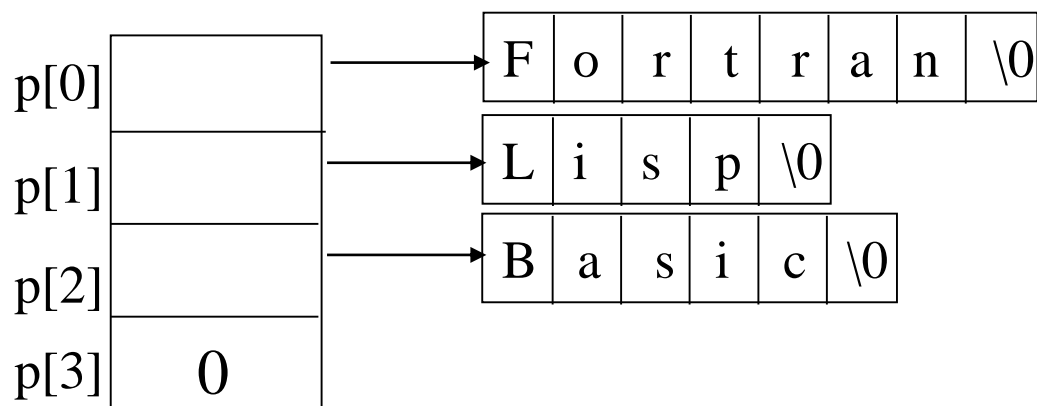
初始化:

```
void main()
```

```
{ char *p[]={ "Fortran", "Lisp", "Basic",NULL};
```

```
.....
```

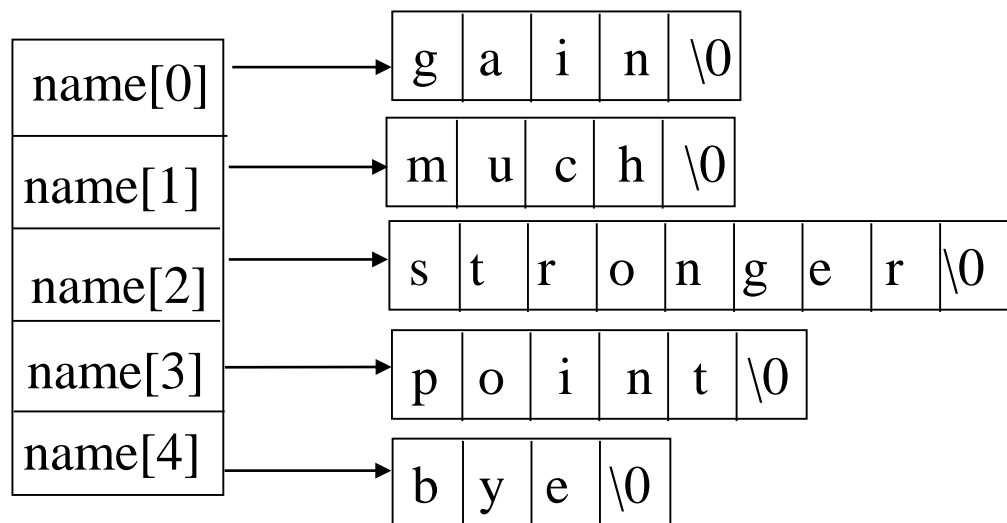
```
}
```



❖ 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”};
```

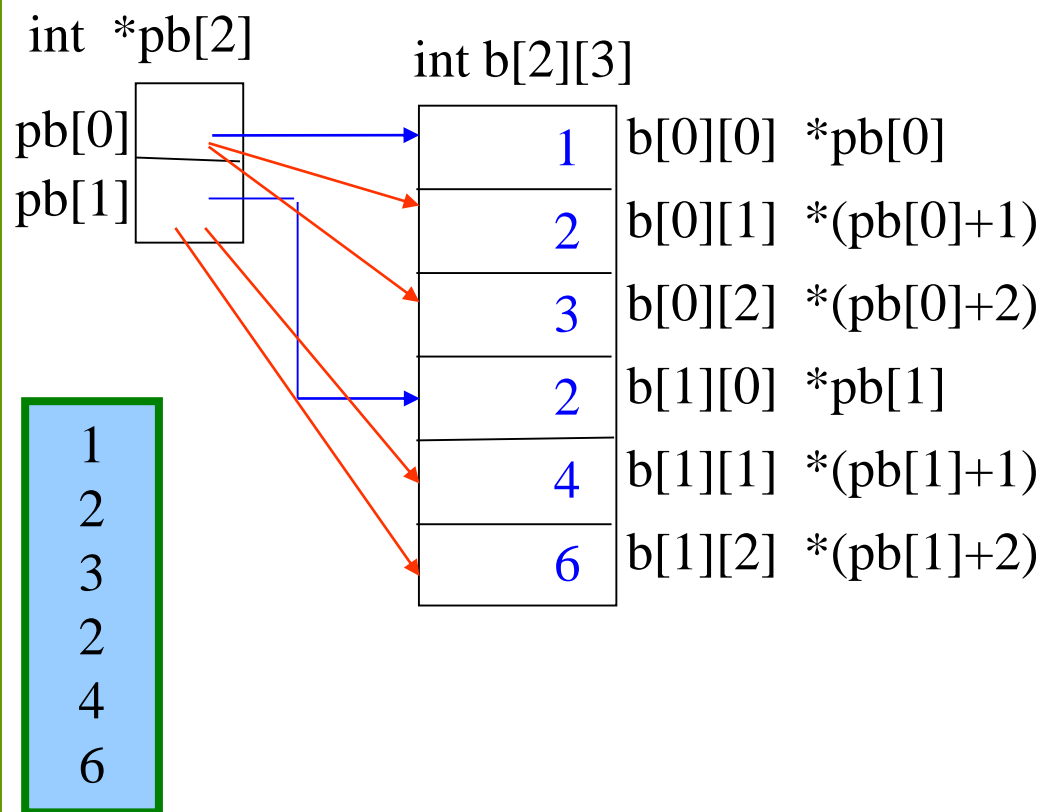
二维数组存储空间固定

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+数组维数

例 用指针数组处理二维数组

```
void 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]++)  
            cout<<*pb[i]<<endl;  
}
```

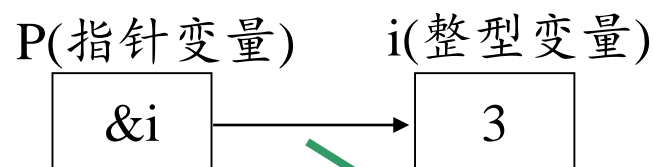


★ 多级指针

❖ 定义: 指向指针的指针

❖ 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;
```

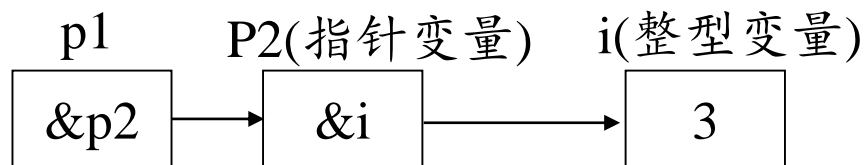


一级指针

单级间接寻址

❖ 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;
```



二级指针

一级指针

目标变量

二级间接寻址

指针本身的存储类型

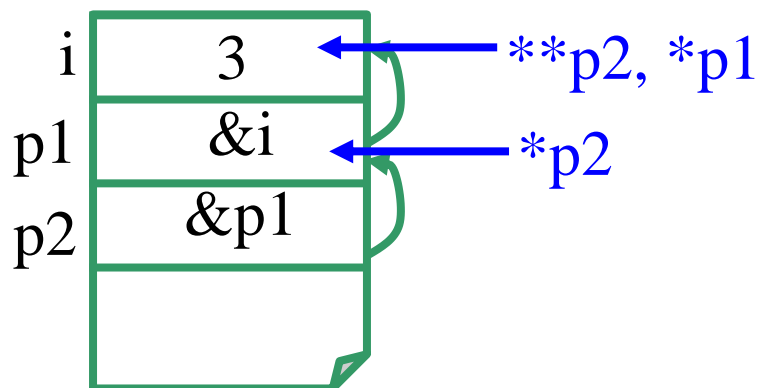
最终目标变量的数据类型

● 定义形式: [存储类型] 数据类型 **指针名;

如 char **p;

*p是p间接指向对象的地址
**p是p间接指向对象的值

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



❖ 多级指针

例 三级指针 int ***p;

四级指针 char ****p;

指针的数据类型

定义	含义
<code>int i;</code>	定义整型变量 <code>i</code>
<code>int *p;</code>	<code>p</code> 为指向整型数据的指针变量
<code>int a[n];</code>	定义含 <code>n</code> 个元素的整型数组 <code>a</code>
<code>int *p[n];</code>	<code>n</code> 个指向整型数据的指针变量组成的指针数组 <code>p</code>
<code>int (*p)[n];</code>	<code>p</code> 为指向含 <code>n</code> 个元素的一维整型数组的指针变量
<code>int f();</code>	<code>f</code> 为返回整型数的函数
<code>int *p();</code>	<code>p</code> 为返回指针的函数，该指针指向一个整型数据
<code>int (*p)();</code>	<code>p</code> 为指向函数的指针变量，该函数返回整型数
<code>int **p;</code>	<code>p</code> 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) <code>int *p[3];</code> | ← | 指针数组 |
| (2) <code>int (*p)[3];</code> | ← | 指向一维数组的指针 |
| (3) <code>int *p(int);</code> | ← | 返回指针的函数 |
| (4) <code>int (*p)(int);</code> | ← | 指向函数的指针，函数返回int型变量 |
| (5) <code>int *(*p)(int);</code> | ← | 指向函数的指针，函数返回int型指针 |
| (6) <code>int (*p[3])(int);</code> | ← | 函数指针数组，函数返回int型变量 |
| (7) <code>int *(*p[3])(int);</code> | ← | 函数指针数组，函数返回int型指针 |

§ 6.11 引用

★ 引用就是一个变量或对象的别名。

➤ 引用的声明需要用到引用运算符“&”，一般形式如下：

类型 &变量 = 变量;

例如： `int a = 10; int &ra = a;`

➤ 注意引用的声明方法是：先写上目标对象的数据类型，然后跟引用运算符“&”，接着写引用的名字。

➤ 声明引用时，必须给引用赋初值。

```
void main()    变量a和引用b共用同一内存空间.  
{  int   a=10;  
    int   &b=a;    //声明b是对整数a的引用  
    a=a*a;    //a的值变化了, b的值也一起变化  
    cout<<a<<setw ( 6 )<<b<< endl;  
    b=b/5;    //b的值变化了, a的值也一起变化  
    cout<<b<< setw ( 6 )<<a<< endl;  
}
```

运行结果为:

100	100
20	20

● 引用特点：

- (1) 可以引用任何合法变量名；
- (2) 引用不是变量，是声明，必须初始化；
- (3) 声明引用时，目标的存储状态不会改变。所以，引用只有声明，没有定义。
- (4) 引用仅在声明时带有“&”，以后就像普通变量一样使用，不需再带“&”；
- (5) 引用最好用做函数参数。

引用作为函数参数

函数参数传递的方法：

1. 将变量名作为实参和形参。这时传给形参的是变量的值，传递是单向的。

输出结果： 3, 5

失败！

```
#include <iostream>
void swap(int x, int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
void main()
{
    int a=3,b=5;
    if(a<b) swap(a,b);
    cout<<a<<","<<b<<endl;
}
```

引用作为函数参数

函数参数传递的方法：

2. 传递变量的指针。形参是指针变量，实参是一个变量的地址，调用函数时，形参(指针变量)指向实参变量单元。

输出结果： 5, 3

共享内存，“双向”传递

```
#include <iostream>
void swap(int *p1, int *p2)
{   int p;
    p=*p1;
    *p1=*p2;
    *p2=p;
}
void main()
{   int a=3,b=5;
    int *p1=&a,*p2=&b;
    if(a<b) swap(p1,p2);
    cout<<a<<","<<b<<endl;
}
```

引用作为函数参数

函数参数传递的方法：

3. 使用引用，
即传送变量的别名。

输出结果： 5, 3

```
#include <iostream>
void swap(int &x, int &y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
void main()
{
    int a=3,b=5;
    if(a<b) swap(a,b);
    cout<<a<<","<<b<<endl;
}
```

例 对3个变量按由小到大的顺序排序。

```
#include <iostream>

void change (int &x, int &y)
//使x和y互换
{ int t; t=x; x=y; y=t; }

void sort(int &i, int &j, int &k)
//对i,j,k 3个数排序
{
    if (i>j) change (i,j); //使i<=j
    if (i>k) change (i,k); //使i<=k
    if (j>k) change (j,k); //使j<=k
}
```

```
void main()
{ int a,b,c;
  int a1,b1,c1;
  cout<<" Enter 3 integers:" ;
  cin>>a>>b>>c; //输入a,b,c
  a1=a; b1=b; c1=c;
  sort(a1, b1, c1);
  cout<<" Sorted: " <<a1<<" "
    <<b1<<" " <<c1<<endl;
}
```

作业：

P₁₈₈

1, 3, 8

提示：要求使用指针或引用