

《面向对象程序设计》之——

指 针

连 盛

shenglian@fzu.edu.cn

计算机与大数据学院

福州大学

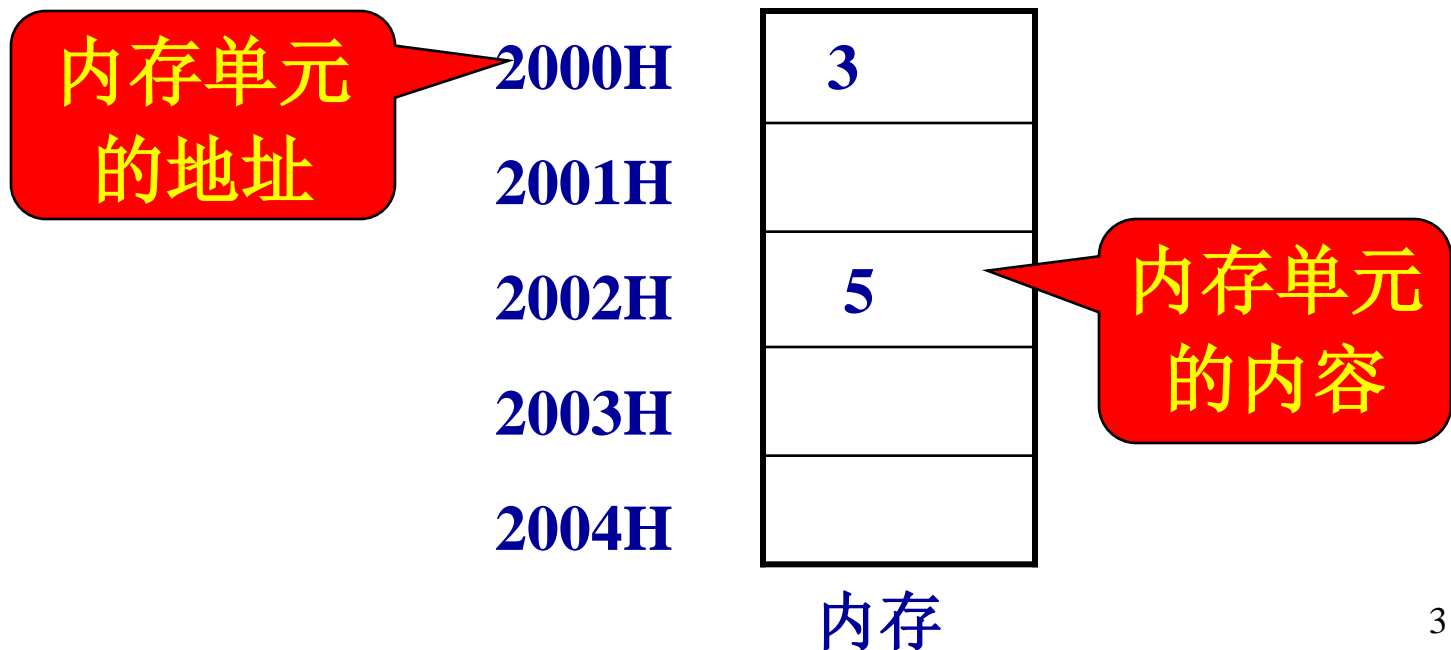
本节提纲

- 指针的概念
- 指针变量，指针函数
- 数组、字符串、函数与指针
- 引用与指针

指针的概念

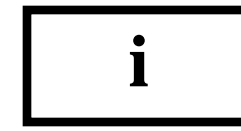
数据在内存中是如何存取的？

系统根据程序中定义变量的类型，给变量分配一定的长度空间。字符型占1个字节，整型数占4个字节.....。内存区的每个字节都有编号，称之为**地址**。



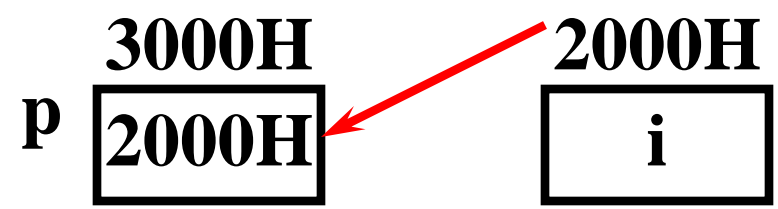
1、直接访问

按变量地址存取变量的值。**cin>>i**；实际上放到定义 **i** 单元的**地址**中。



2、间接访问

将变量的地址存放在另一个单元**p**中，通过 **p** 取出变量的地址，再针对变量操作。



一个变量的地址称为该变量的指针。

如果在程序中定义了一个变量或数组，那么，这个变量或数组的地址（指针）也就确定为一个**常量**。

变量的**指针**和指向变量的**指针变量**

变量的指针就是**变量的地址**，当变量定义后，其指针（地址）是一常量。

```
int i;           &i : 2000H           2000H
                                     ┌───┴───┐
                                     │   i   │
```

可以**定义一个变量**专门用来**存放**另一变量的**地址**，这种变量我们称之为**指针变量**。在编译时同样分配一定字节的存储单元，未赋初值时，该存储单元内的值是随机的。

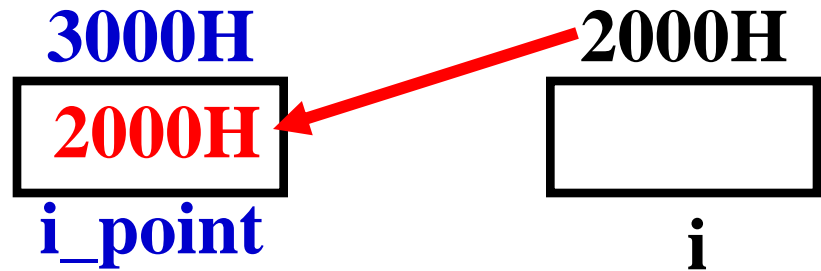
指针变量定义的一般形式为：

```
类型标识符      *变量名      指针类型      变量名
int              *i_point;
```

指针**变量**同样也可以赋值：

```
int i, *i_point;
```

```
i_point=&i;
```



也可以在定义**指针变量**时赋初值：

```
int i;
```

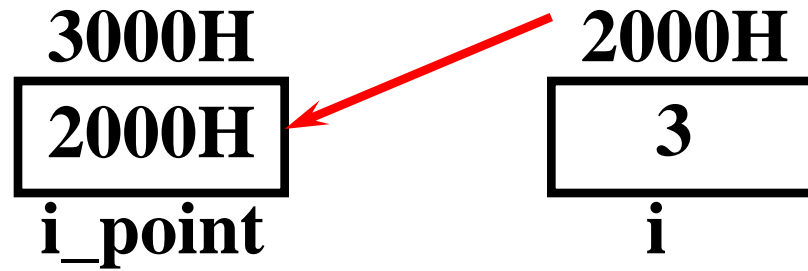
```
int *i_point=&i;
```



一个指针变量只能指向同一类型的变量。即整型指针变量只能放整型数据的地址，而不能放其它类型数据的地址。

* 在**定义语句**中只表示变量的类型是指针，没有任何计算意义。


* 在语句中表示“指向”。&表示“地址”。



int i; **表示类型**

int *i_point=&i;

表示指向 *i_point=3;



指针变量的引用

指针变量只能存放地址，不要将非地址数据赋给指针变量。

```

int *p, i;    p=100;    p=&i;

void main(void)
{
    int a=10, b=100;

    int *p1, *p2;
    p1=&a; p2=&b;

    cout<<a<<'\t'<<b<<endl;

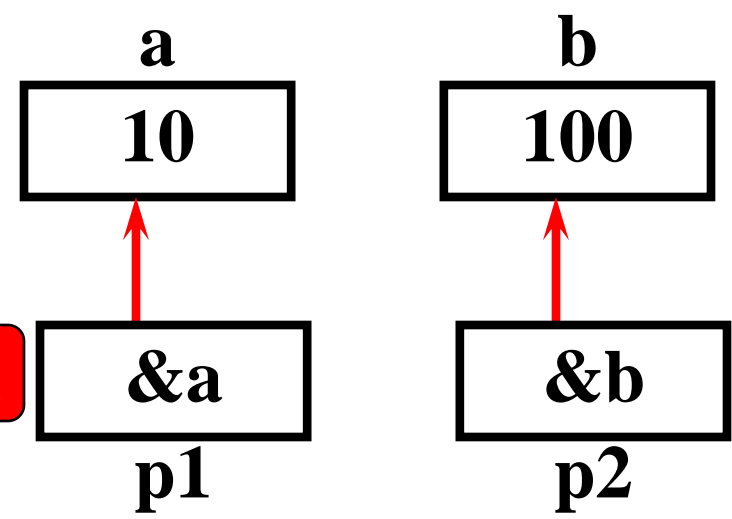
    cout<<*p1<<'\t'<<*p2<<endl;
}
    
```

非法

指针变量赋值

表示指向

指针变量引用



10 100

10 100


```
void main(void)
```

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

```
int *p1, *p2;
```

指针变量赋值

```
p1=&a; p2=&b;
```

```
*p1=10; *p2=100;
```

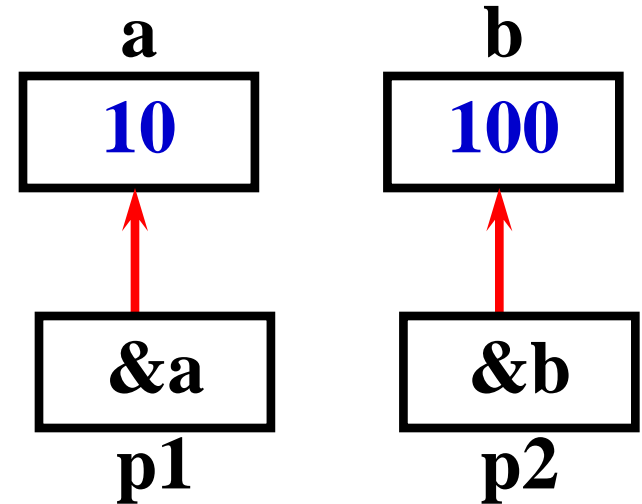
通过指针对
变量赋值

```
cout<<a<<'\t'<<b<<endl;
```

```
cout<<*p1<<'\t'<<*p2<<endl;
```

```
}
```

指针变量引用



```
void main(void)
```

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

```
int *p1, *p2;
```

```
*p1=10; *p2=100;
```

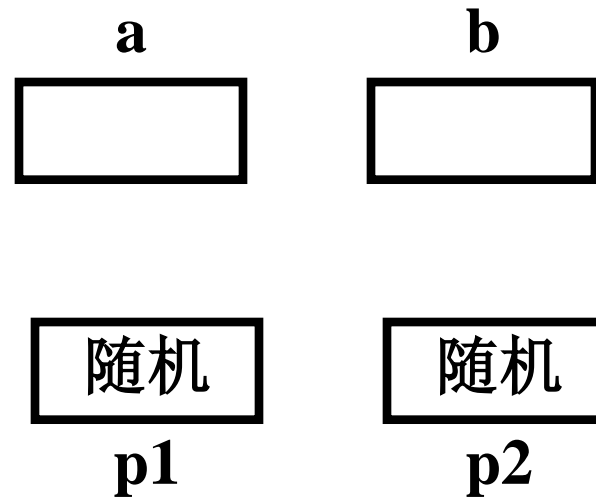
```
cout<<a<<'\t'<<b<<endl;
```

```
cout<<*p1<<'\t'<<*p2<<endl;
```

```
}
```

但指针变量未赋值，即
指针指向未知地址

通过指针对
变量赋值



绝对不能对未赋值的指针变量作“指向”运算。

```
int i, *p1;
```

```
p1=&i;
```

用指针变量前，必须
对指针变量赋值

输入a, b两个整数，按大小输出这两个数。

```
void main(void)
```

```
{ int *p1, *p2, *p, a, b;
```

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

```
p1=&a; p2=&b;
```

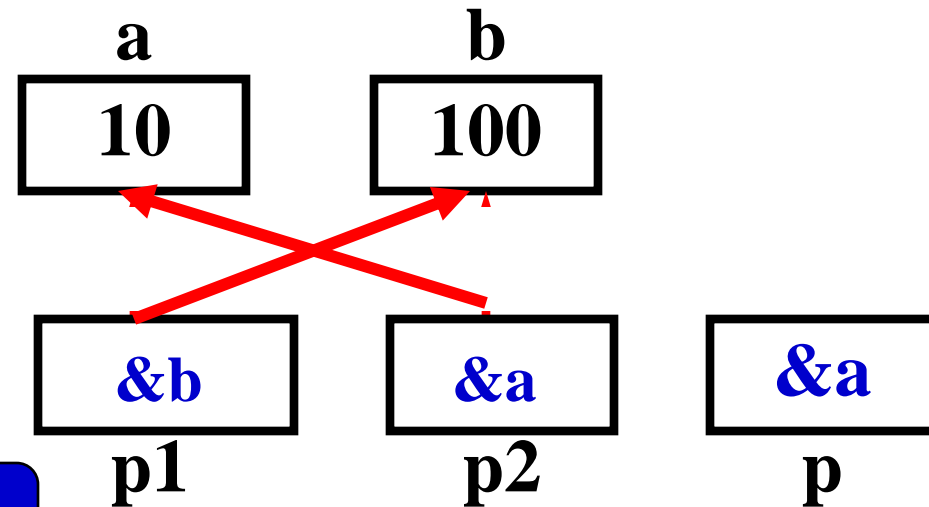
```
if (a<b)
```

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

```
cout<<a<<'\t'<<b<<endl;
```

```
cout<<*p1<<'\t'<<*p2<<endl;
```

```
}
```



10 100

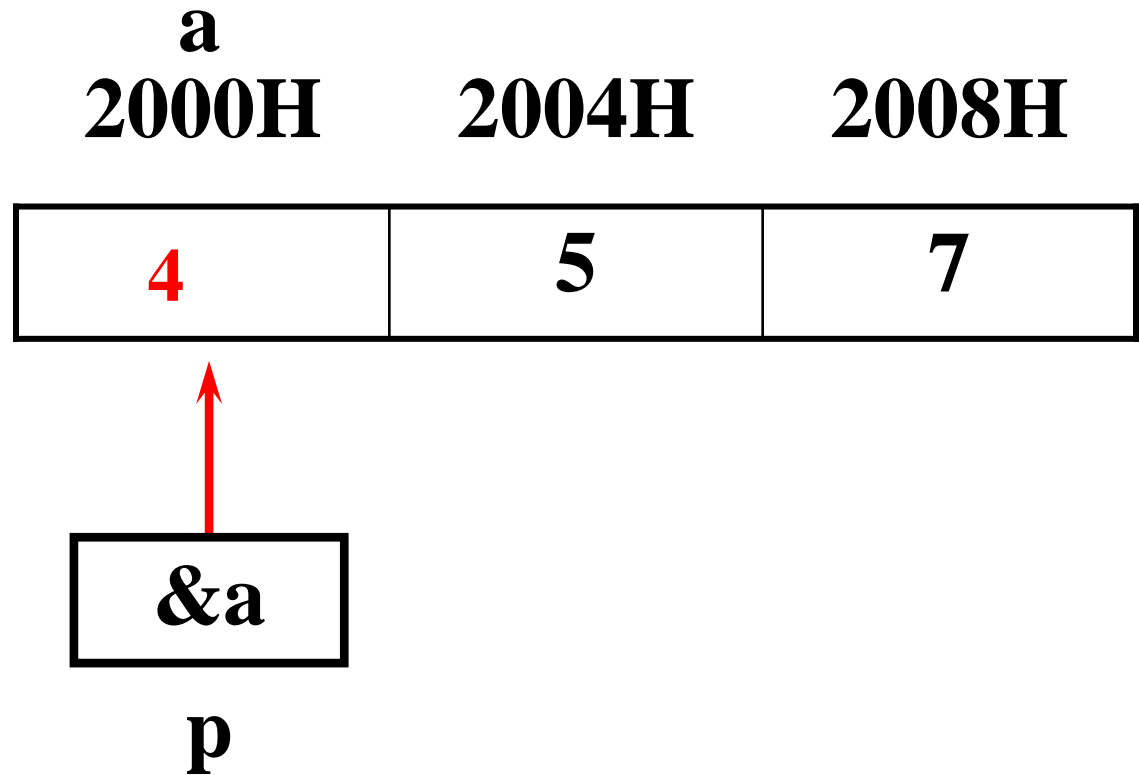
100 10

虽然变量不变，但指向变量的指针发生变化

++, --, * 优先级相同，都是右结合性。

int a=3, *p;

p=&a;

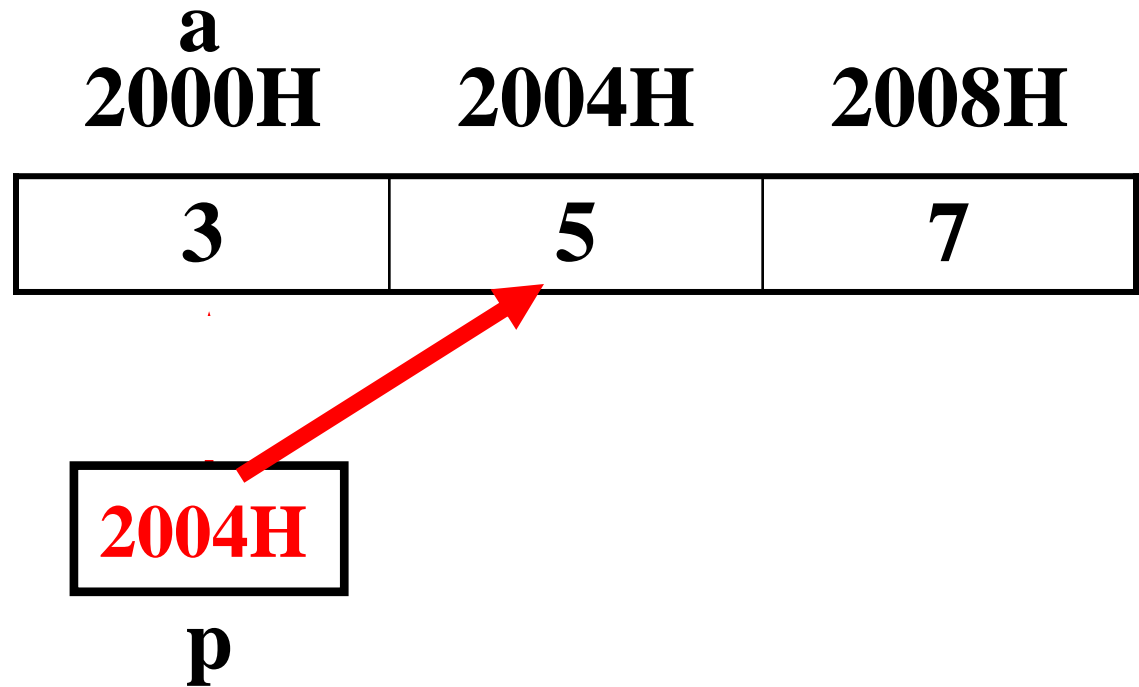


(*p)++; 相当于**a++**。表达式为**3**, **a=4**

++, --, * 优先级相同，都是右结合性。

int a=3, *p;

p=&a;

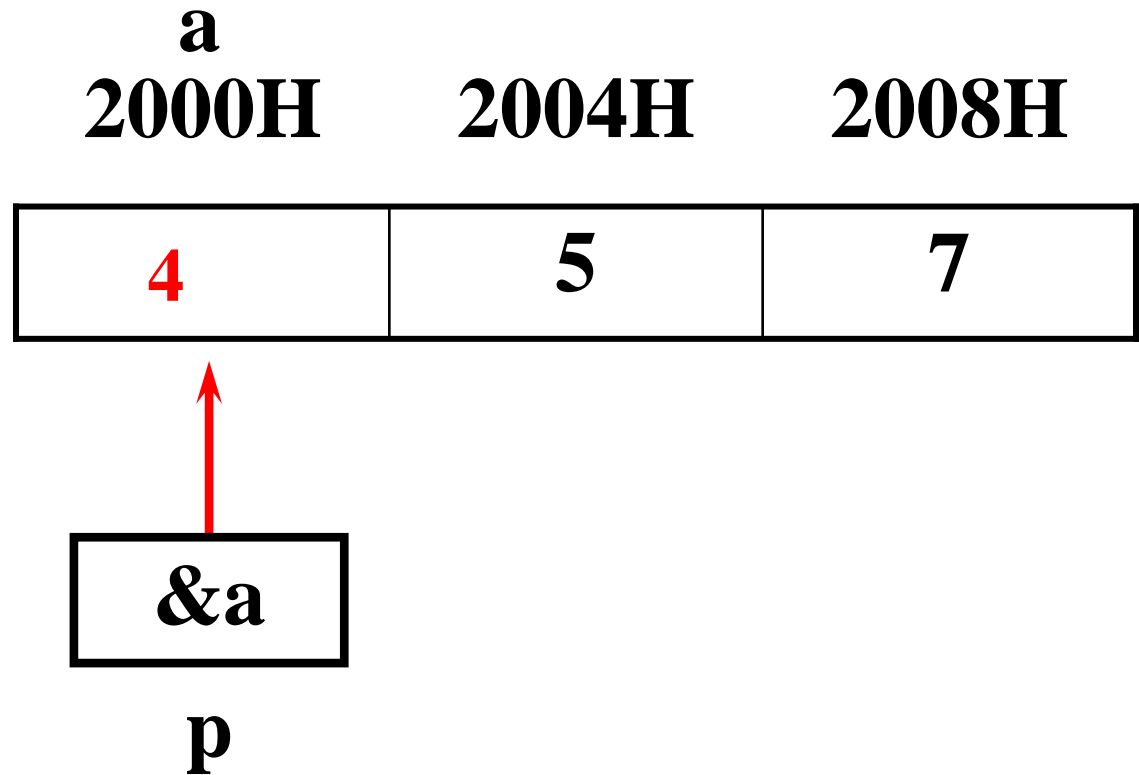


p++;** ***(p++)**首先p**，然后**p=p+1**,指针指向下一个int单元 表达式为**3, p=2004H**。

++, --, * 优先级相同，都是右结合性。

int a=3, *p;

p=&a;



++*p

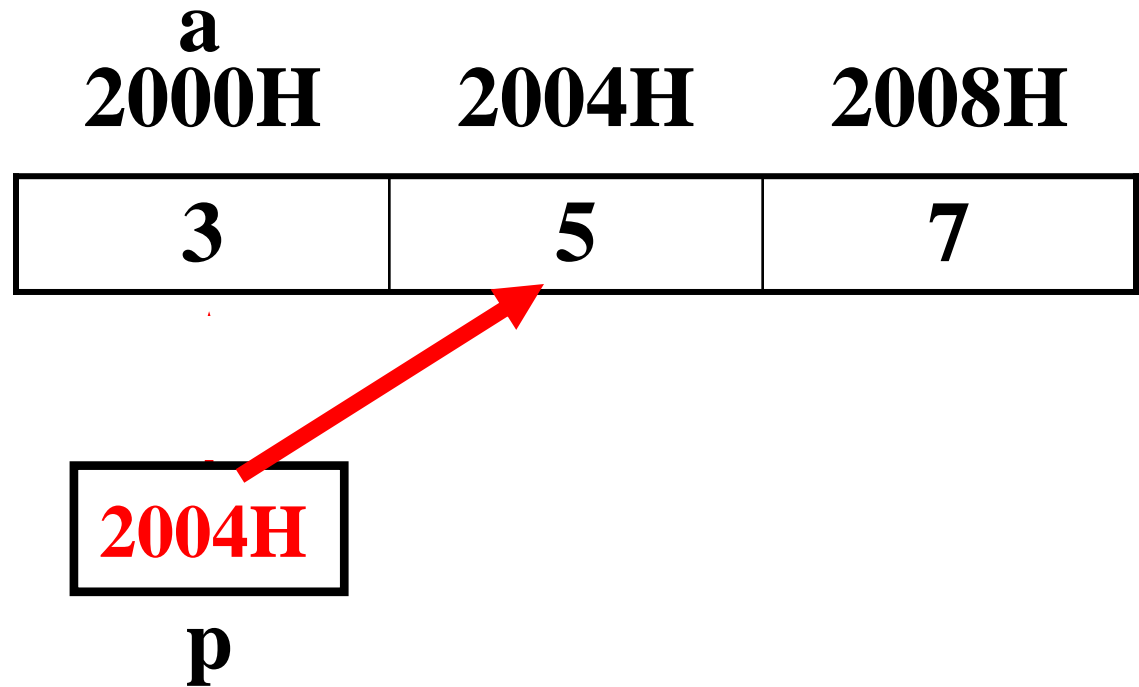
++(*p)

***p=*p+1 a=4**

++, --, * 优先级相同，都是右结合性。

int a=3, *p;

p=&a;



++p** ***(++p)**, 首先: **p=p+1**, 然后取p**。即取**p**所指的下一个**int**单元的内容。

表达式为5 **p=2004H**

指针变量作为函数参数：

函数的参数可以是指针类型，它的作用是将一个变量的地址传送到另一个函数中。

指针变量作为函数参数与变量本身作函数参数不同，变量作函数参数传递的是具体值，而指针作函数参数传递的是内存的地址。

输入a, b两个整数，按大小输出这两个数。

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

```
{ int t;
```

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

```
}
```

```
void main(void)
```

```
{ int *point1, *point2, a,b;
```

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

```
  point1=&a; point2=&b;
```

```
  if (a<b)
```

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

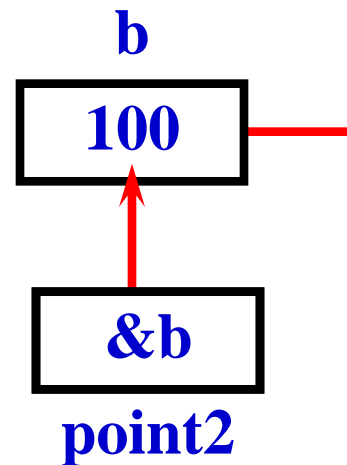
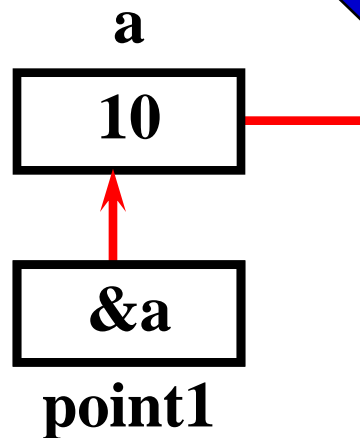
```
  cout<<"a="<<a<<" ,b="<<b<<"\n";
```

```
  cout<<*point1<<"\t"<<*point2;
```

```
}
```

值传递

x=100



y=10

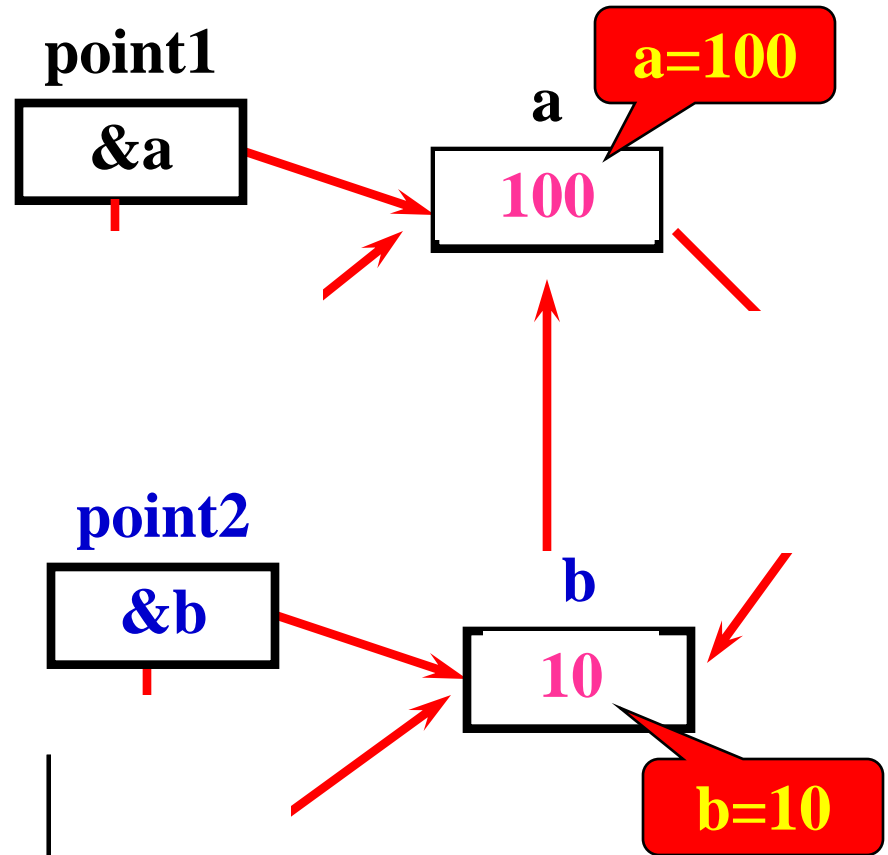
输出: a=10,b=100

10,100

输入a, b两个整数，按大小输出这两个数。

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

```
void main(void)
{ int *point1, *point2, a,b;
  cin>>a>>b;
  point1=&a; point2=&b;
  if (a<b)
    swap (point1, point2);
  cout<<"a="<<a<<","<<b<<endl;
  cout<<*point1<<","<<*point2<<endl;
}
```



输出: a=100,b=10

100,10

用指针变量作函数参数，在被调函数的执行过程中，应使指针变量所指向的参数值发生变化，这样，函数在调用结束后，其变化值才能保留回主调函数。

函数调用不能改变实参指针变量的值，但可以改变实参指针变量所指向变量的值。

用指针变量作函数参数，可以得到多个变化了的值。

数组的**指针**和指向数组的**指针变量**

数组与变量一样，在内存中占据单元，有地址，一样可以用指针来表示。C++规定：**数组名就是数组的起始地址**；又规定：**数组的指针就是数组的起始地址**。数组元素的指针就是数组元素的地址。

一、指向数组元素的指针变量的定义与赋值

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

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

数组第一个元素的地址

```
p=a;
```

直接用数组名赋值

&a[0]

p

2000H

2004H

2008H

200CH

2010H

2014H

2018H

201CH

2020H

2024H

a

a[0]

a[1]

a[2]

a[3]

a[4]

a[5]

a[6]

a[7]

a[8]

a[9]

p是变量，a为常量。

若数组元素为int型，则指向其的指针变量也应定义为int型。

```
int a[10];
```

这两种情况均为赋初值

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

二、通过指针引用数组元素

```
int a[10];
```

```
int *p=a;
```

为指针变量赋初值

```
*p=1;
```

通过指针变量为数组元素赋值

```
a[0]=1;
```

C++规定，**p+1**指向数组的下一个元素，而不是下一个字节。

```
*(p+1)=2;
```

```
a[1]=2;
```

指针变量也重新赋值

```
*++p=2;
```

```
p=p+1; *p=2; p=2004H
```

a		
2000H	1	a[0]
2004H	2	a[1]
2008H		a[2]
200CH		a[3]
2010H		a[4]
2014H		a[5]
2018H		a[6]
201CH		a[7]
2020H		a[8]
2024H		a[9]

$*(p+1)=2;$ $a[1]=2;$

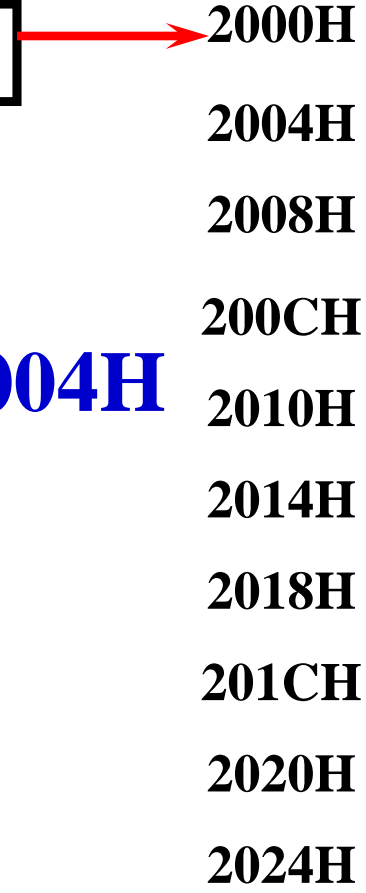
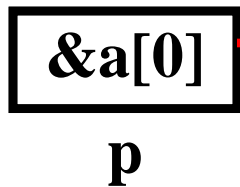
$*(a+1)=2;$

$*(a+1)$ 与 $a[1]$ 等同。

$*++p=2;$ $p=p+1;$ $*p=2;$ $p=2004H$

$*++a=2;$ 错误

a 为常量，不可赋值。



a	
1	$a[0]$
2	$a[1]$
	$a[2]$
	$a[3]$
	$a[4]$
	$a[5]$
	$a[6]$
	$a[7]$
	$a[8]$
	$a[9]$

$p+i$ 或 $a+i$ 均表示 $a[i]$ 的地址 $\&a[i]$

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

用指向数组的指针变量输出数组的全部元素



```
void main(void)
```

```
{ int a[10], i;
```

```
int *p;
```

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

```
cin>>a[i];
```

输入数组元素

指针变量赋初值

指向下一元素

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

```
cout<<*p<<'\t';
```

输出指针指向的数据

```
}
```

```
void main(void)
```

```
{ int a[10], i;
```

```
int *p=a;
```

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

```
cin>>a[i];
```

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

```
cout<<*p++<<'\t';
```

```
*p, p=p+1
```

输出数据后指针加1

```
}
```



```
void main(void)
```

```
{  int  x[ ]={1,2,3};
```

```
    int  s, i, *p;
```

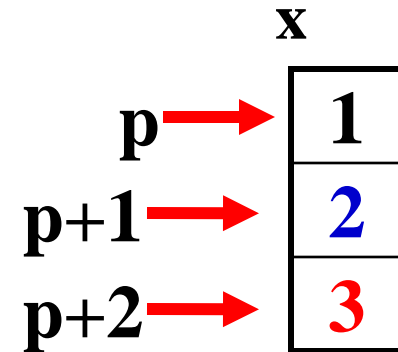
```
    s=1; p=x;
```

```
    for (i=0; i<3; i++)
```

```
        s*=* (p+i);
```

```
        cout<<s<<endl;
```

```
}
```



i=0 $s = s * (* (p+0)) = s * 1 = 1$

i=1 $s = s * (* (p+1)) = s * 2 = 2$

i=2 $s = s * (* (p+2)) = s * 3 = 6$



```
static int a[] = {1, 3, 5, 7, 11, 13};
```

```
main( )
```

```
{ int *p;
```

```
  p = a + 3;
```

```
  cout << *p << '\t' << (*p++) << endl;
```

```
  cout << *(p-2) << '\t' << *(a+4) << endl;
```

```
}
```

p

&a[3]



1	a[0]
3	a[1]
5	a[2]
7	a[3]
11	a[4]
13	a[5]

7 7
5 11

三、数组名作函数参数

数组名可以作函数的实参和形参，传递的是**数组的地址**。这样，实参、形参共同指向同一段内存单元，内存单元中的数据发生变化，这种变化会反应到主调函数内。

在函数调用时，**形参数组并没有另外开辟新的存储单元**，而是以实参数组的首地址作为形参数组的首地址。**这样形参数组的元素值发生了变化也就使实参数组的元素值发生了变化。**

1、形参实参都用数组名

```
void main(void)
```

```
{ int array[10];
```

```
.....
```

```
f(array, 10);
```

```
....
```

```
}
```

形参数组,必须进行类型说明

```
f(int arr[ ], int n)
```

```
{
```

```
.....
```

```
}
```

实参数组

用数组名作形参，因为接收的是地址，所以可以不指定具体的元素个数。

指向同一
存储区间

array, arr		arr[0]
2000H		array[0]
2004H		array[1]
2008H		array[2]
200CH		array[3]
2010H		array[4]
2014H		array[5]
201CH		array[6]
2020H		array[7]
2024H		array[8]
2028H		array[9]

2、实参用数组名，形参用指针变量

```
void main(void)
```

```
{ int a [10];
```

```
.....
```

```
f(a, 10);
```

```
.....
```

```
}
```

```
f(int *x, int n )
```

```
{
```

形参指针

```
.....
```

```
}
```

实参数组

3、形参实参都用指针变量

```
void main(void)
```

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

```
  p=a;
```

```
  ....
```

```
  f(p, 10);
```

```
  ....
```

```
}
```

形参指针

```
f(int *x, int n )
```

```
{
```

```
  ....
```

```
}
```

实参指针

实参指针变量调用前必须赋值

4、实参为指针变量，形参为数组名

```
void main(void)
```

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

```
    p=a;
```

```
    .....
```

```
    f(p, 10);
```

```
    .....
```

```
}
```

形参数组

```
f(int x[ ], int n )
```

```
{
```

```
    .....
```

```
}
```

实参指针

将数组中的n个数按相反顺序存放。

```
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(void)
{ int i, a[10]={3,7,9,11,0,6,7,5,4,2};
  inv(a,10);
  for (i=0; i<10; i++)
    cout<<a[i]<<'\t';
}
```

x与a数组指向同一段内存

	x, a	
x[0]	3	a[0]
x[1]	7	a[1]
x[2]	9	a[2]
x[3]	11	a[3]
x[4]	0	a[4]
x[5]	6	a[5]
x[6]	7	a[6]
x[7]	5	a[7]
x[8]	4	a[8]
x[9]	2	a[9]

```
void inv(int *x, int n)
{
    int *p, t, *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(void)
{
    int i, a[10]={3,7,9,11,0,6,7,5,4,2};
    inv(a,10);
    for (i=0;i<10; i++)
        cout<<a[i]<<'\t';
}
```

用指针变量来
接受地址

x, a		
i →	x[0]	3
	x[1]	7
	x[2]	9
	x[3]	11
p →	x[4]	0
	x[5]	6
	x[6]	7
	x[7]	5
	x[8]	4
j →	x[9]	2

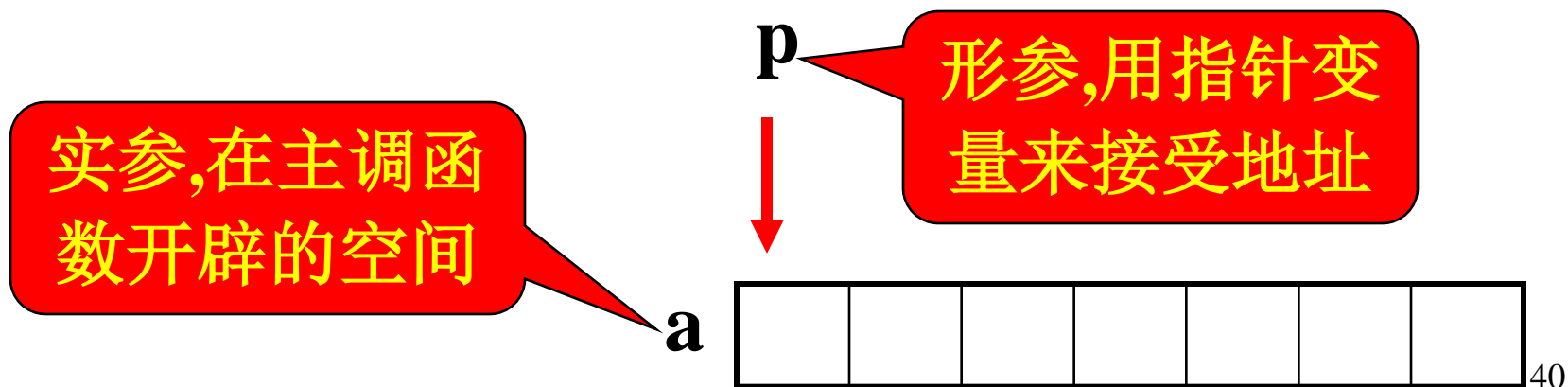
数组名作函数参数

数组名可以作函数的实参和形参，传递的是**数组的地址**。这样，实参、形参共同指向同一段内存单元，内存单元中的数据发生变化，这种变化会反应到主调函数内。

在函数调用时，**形参数组并没有另外开辟新的存储单元**，而是以实参数组的首地址作为形参数组的首地址。**这样形参数组的元素值发生了变化也就使实参数组的元素值发生了变化。**

既然数组做形参没有开辟新的内存单元，接受的只是实参数组的首地址，那么，这个首地址也可以在被调函数中用一个**指针变量**来接受，通过在被调函数中对这个**指针变量的指向**进行操作而使实参数组发生变化。

实际上在被调函数中只开辟了p的空间，里面放的是a的值。



四、指向多维数组的指针和指针变量

用指针变量也可以指向多维数组，表示的同样也是多维数组的首地址。

```
int a[3][4]; //首地址为2000H
```

2000H		2008H		2010H	2014H		201cH	2020H		2028H	202cH
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]

a		2000H	2004H	2008H	200CH
2000H	a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
2010H	a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2020H	a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

可以将a数组看作一个一维数组，这个一维数组的每个元素又是一个具有4个int型数据的一维数组，这样，**我们就可以利用一维数组的概念来标记一些写法。**

	a	2000H	2004H	2008H	200CH
2000H	a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
2010H	a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2020H	a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

a[0]=*(a+0) a+0为a[0]的地址&a[0]，其值为2000H。

a[1]=*(a+1) a+1为a[1]的地址&a[1]，其值为2010H。

a[2]=*(a+2) a+2为a[2]的地址&a[2]，其值为2020H。

a[0]为一维数组名，其数组有四个int型的元素：

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

同样，**a[0]**代表一维数组的首地址，所以，a[0]为&a[0][0]。

	a	2000H	2004H	2008H	200CH
2000H	a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
2010H	a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2020H	a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

把a[0]看成
一维数组b

a[0]代表一维数组的首地址,也就是一维数组名,a[0]为
&a[0][0]。

a[0]为&a[0][0] a[0][0]=*(a[0]+0) b[0] = *(b+0)

a[0]+1为&a[0][1] a[0][1]=*(a[0]+1) b[1] = *(b+1)

a[0]+2为&a[0][2] a[0][2]=*(a[0]+2) b[2] = *(b+2)

a[0]+3为&a[0][3] a[0][3]=*(a[0]+3) b[3] = *(b+3)

行 列

a[1]+2为&a[1][2] a[1][2]=*(a[1]+2)

a[i][j]=*(a[i]+j)

a		2000H	2004H	2008H	200CH
2000H	a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
2010H	a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2020H	a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

行

列

$a[1]+2$ 为 $\&a[1][2]$ $a[1][2]=*(a[1]+2)$ $a[i][j]=*(a[i]+j)$

a 为二维数组名， $a+1$ 为 $a[1]$ 的地址，也就是数组第一行的地址，所以 a 为行指针。

$a[1]$ 为一维数组名， $a[1]+1$ 为 $a[1][1]$ 的地址，也就是数组第一行第一列的地址，所以 $a[1]$ 为列指针。

a		2000H	2004H	2008H	200CH
2000H	a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
2010H	a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2020H	a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

可以看到： **a**, **a+0** , ***(a+0)**, **a[0]**, **&a[0][0]**表示的都是**2000H**，即二维数组的首地址。

实际上， **a[0]**, **a[1]**, **a[2]**并不是**实际的元素**，它们在内存并不占具体的存储单元，只是为了我们表示方便起见而设计出的一种表示方式。

a为行指针，加1移动一行。

***a或a[0]为列指针，加1移动一列。**

	a	2000H	2004H	2008H	200CH
2000H	a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
2010H	a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2020H	a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

$$\mathbf{a[1]=*(a+1) \quad *(a+1)+2=\&a[1][2]}$$

$$\mathbf{*(*(a+1)+2)=a[1][2]}$$

$$\mathbf{**(\mathbf{a+1})=*(a[1])=*(*(a+1)+0)=a[1][0]}$$

$$\mathbf{(*(a+1))[1]=*(*(a+1)+1)=a[1][1]}$$

$$\mathbf{*(a+1)[1]=*(((a+1)[1])=*((*(a+1)+1))=**(\mathbf{a+2})=a[2][0]}$$

注意二维数组的各种表示法，**a**为常量。

多维数组的指针作函数参数

主要注意的是函数的**实参**究竟是行指针还是列指针，从而决定函数**形参**的类型。要求**实参、形参一一对应，类型一致**。

字符串的表示形式

string

数组首地址

1、用字符数组实现

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

```
void main(void )
```

```
{ char string[ ]="I love China";
```

cout<<string;

}

string为数组名，代表数组的首地址，是常量。

string

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

char string[20];

string="I love China";

错误！常量不能赋值

strcpy(string, "I love China");

正确赋值形式

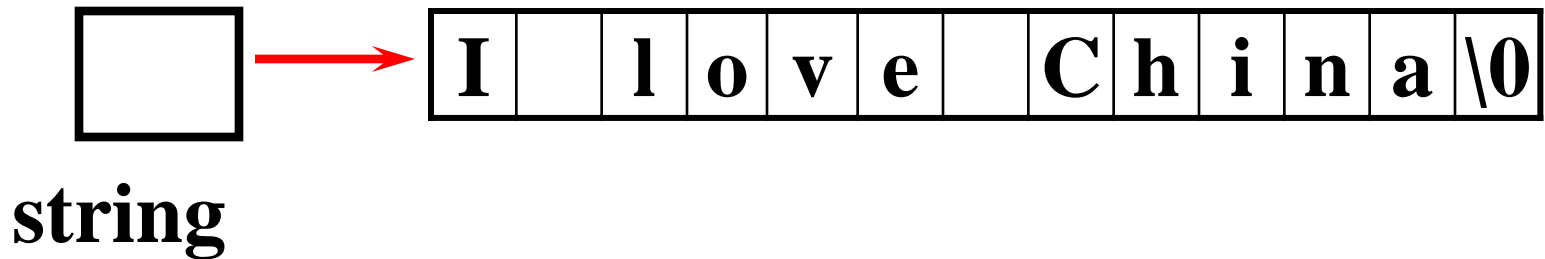
cin.getline(string); //从键盘输入

2、用字符**指针**表示字符串

```
void main(void)
{ char *string="I love China";
  cout<<string;
}
```

指针变量

字符串常量



将内存中字符串常量的首地址赋给一个指针变量

- 指针的概念
- 指针变量，指针函数
- 指针作为函数形参、实参
- 数组、字符串等

字符串**指针**作函数参数

将一个字符串从一个函数传递到另一个函数，可以用地址传递的办法。即用**字符数组名作参数或用指向字符串的指针变量作参数**。在被调函数中可以改变原字符串的内容。

将字符串a复制到字符串b。

```
void main(void)
```

```
{ char a[ ]="I am a teacher";
```

```
char b[ ]="You are a student";
```

```
copy_string(a , b);
```

from与a一个地址, to与b一个地址

```
cout<<a<<endl;
```

```
cout<<b<<endl;
```

```
}
```

```
copy_string( char from[],char to[])
```

```
{ int i;
```

```
for (i=0; from[i]!='\0'; i++)
```

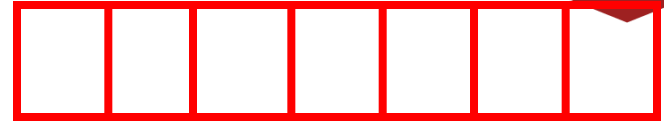
```
to[i]=from[i];
```

```
to[i]='\0';
```

```
}
```

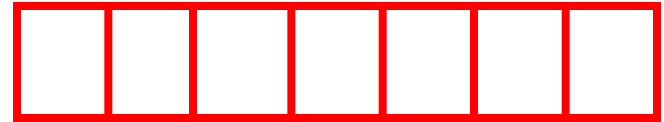
from

a



to

b



将字符串a复制到字符串b。

`copy_string(char *from, char *to)`



```
{ for ( ; *from!='\0'; )
```

```
for( ; *from++=*to++; ) ;
```

```
*to++=*from++;
```

```
*to='\0';
```

```
void main(void) }
```

```
{ char a[ ]={"I am a teacher"};
```

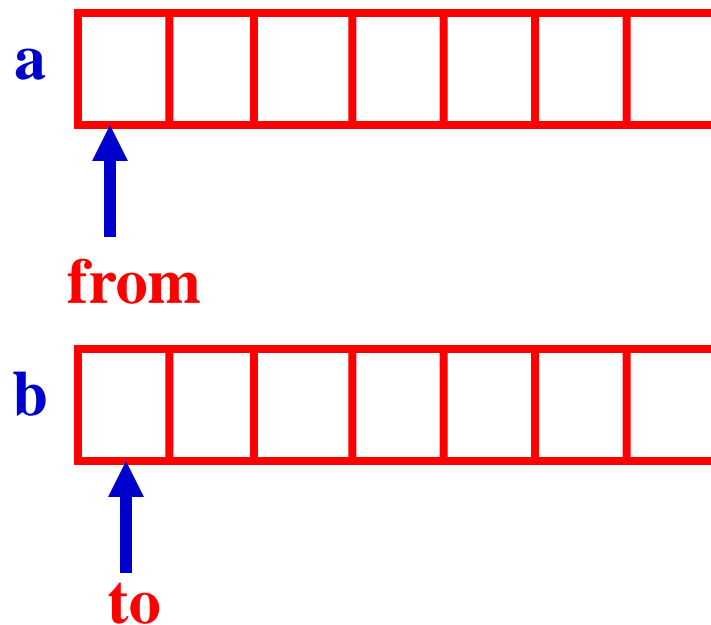
```
char b[ ]={"You are a student"};
```

```
copy_string(a,b);
```

```
cout<<a<<endl;
```

```
cout<<b<<endl;
```

```
}
```



也可以用字符指针来接受数组名

字符指针变量与字符数组

字符**数组**和字符**指针变量**都可以实现字符串的存储和运算，区别在于：

字符数组名是常量，定义时必须指明占用的空间大小。

字符指针变量是变量，里面存储的是字符型地址，可以整体赋值，**但字符串必须以 ‘\0’ 结尾。**

```
void fun(char *s)
```

```
{ int i, j;
```

```
for( i=j=0; s[i]!='\0'; i++)
```

```
    if(s[i]!='c')
```

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

```
    s[j] = '\0';
```

必须以\0结束

```
return;
```

```
}
```

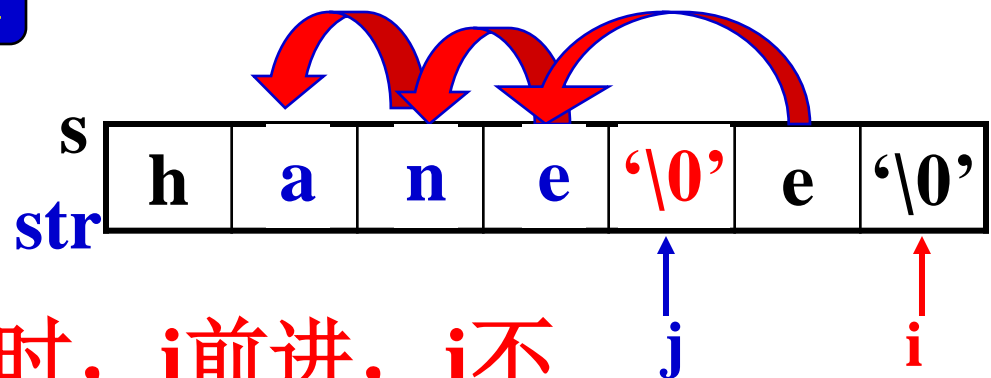
```
void main(void)
```

```
{ char str[80] = "hcance";
```

```
  fun(str);
```

```
  cout << str << endl;
```

```
}
```



当s[i]等于字符 'c'时，i前进，j不

动
输出： hane

```
void swap(char *s1, char *s2)
```

```
{ char t;
```

```
    t=*s1; *s1=*s2; *s2=t;
```

```
}
```

AD

BC

```
void main(void)
```

BB

```
{ char *s1="BD", *s2="BC", *s3="AB";
```

```
    if(strcmp(s1,s2)>0) swap(s1,s2);
```

```
    if(strcmp(s2,s3)>0) swap(s2,s3);
```

```
    if(strcmp(s1,s2)>0) swap(s1,s2);
```

```
    cout<<s1<<endl<<s2<<endl<<s3<<endl;
```

```
}
```

函数的指针和指向函数的指针变量

可以用指针变量指向变量、字符串、数组，也可以指向一个函数。

一个存放地址的指针变量空间可以存放数据的地址（整型、字符型），也可以存放数组、字符串的地址，还可以存放函数的地址。

函数在编译时被分配给一个入口地址。这个入口地址就称为函数的地址，也是函数的指针。像数组一样，C++语言规定，函数名就代表函数的入口地址

专门存放函数地址的**指针变量**称为**指向函数的指针变量**。

函数类型 (*指针变量名)(参数类型);

同时该函数具有两个整型形参

空间的内容只能放函数的地址

`int (*p)(int, int);`

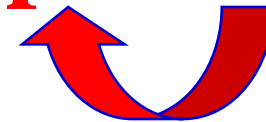
且该函数的返回值为整型数

p



直接用函数名为指针变量赋值。

`p=max;`



`int max (int x, int y)`

`{ return x>y?x:y;`

`}`

这时，指针变量p中放的是max函数在内存中的入口地址。

函数名**max**代表函数在内存中的入口地址，**是一个常量**，不可被赋值。

而指向函数的指针变量**p**可以先后指向不同的**同种类型**的函数。但不可作加减运算。

定义
`int (*p)(int , int);`

赋值
`p=max;`
`p=min;`

如何用指向函数的指针变量调用函数？

```
int max(int x, int y)
{ return x>y?x:y; }
```

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

给指针变量赋值

一般的调用方法

c=(*p)(a,b)



```
int max(int x, int y)
{ return x>y?x:y; }

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

定义指向函数的指针变量

通过指针变量调用

实际上就是用p替换函数名

返回指针值的函数

被调函数返回的不是一个数据，而是一个地址。所以函数的类型为指针类型。

类型标识符 * 函数名(参数表)

指出返回是什么类型的地址

int *max(x, y)

```
int *max (int *x, int *y)
```

```
{ int *pt;
```

```
if (*x>*y)
```

```
pt=x;
```

```
else pt=y;
```

```
return pt;
```

```
}  
返回类型是指针
```

```
void main(void)
```

```
{ int a, b , *p;
```

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

```
p=max(&a,&b);
```

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

```
}
```

用指针类型接收



输出： 4

该指针所指向的空间是在主调函数中开辟的。

```
char *strc(char *str1, char *str2)
```

```
{ char *p;
```

```
for(p=str1;*p!='\0'; p++);
```

p指向str1的最后

```
do { *p++=*str2++; } while (*str2!='\0');
```

```
*p='\0';
```

最后 '\0'结束

str2向p赋值

```
return (str1);
```

```
}
```

```
void main(void)
```

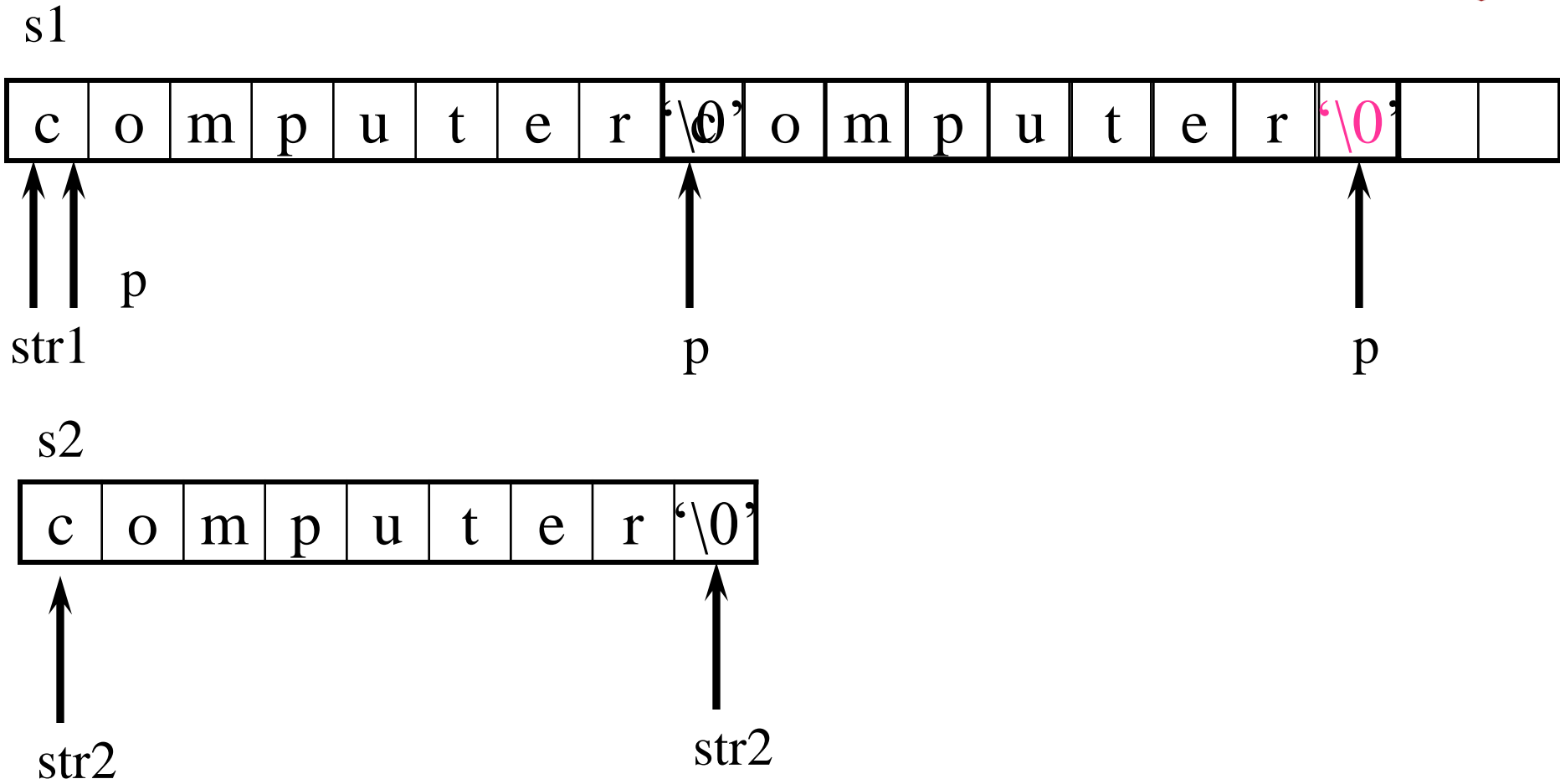
```
{char s1[80]="computer", s2[ ]="language", *pt;
```

```
pt=strc(s1, s2);
```

```
cout<<pt<<endl;
```

computerlanguage

```
}
```



function

strcpy

<cstring>

```
char * strcpy ( char * destination, const char * source );
```

Copy string

Copies the C string pointed by **source** into the array pointed by **destination**, including the terminating null character (and stopping at that point).

To avoid overflows, the size of the array pointed by **destination** shall be long enough to contain the same C string as **source** (including the terminating null character), and should not overlap in memory with **source**.

Parameters

destination

Pointer to the destination array where the content is to be copied.

source

C string to be copied.

Return Value

destination is returned.

<https://cplusplus.com/reference/cstring/strcpy/>

strcmp

```
int strcmp ( const char * str1, const char * str2 );
```

Compare two strings

Compares the C string **str1** to the C string **str2**.

This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.

This function performs a binary comparison of the characters. For a function that takes into account locale-specific rules, see [strcoll](#).

Parameters

str1

C string to be compared.

str2

C string to be compared.

Return Value

Returns an integral value indicating the relationship between the strings:

return value	indicates
<0	the first character that does not match has a lower value in ptr1 than in ptr2
0	the contents of both strings are equal
>0	the first character that does not match has a greater value in ptr1 than in ptr2

指针数组和指向指针的指针

指针数组的概念

一个数组，其元素均为指针类型的数据，称为指针数组。也就是说，指针数组中的每一个元素都是指针变量，可以放地址。

类型标识 *数组名[数组长度说明]

`int *p[4];`

p为数组名，内有四个元素，每个元素可以放一个int型数据的地址

	p
p[0]	地址
p[1]	地址
p[2]	地址
p[3]	地址

`int (*p)[4];`

p为指向有四个int型元素的一维数组的行指针

```
void main(void)
```

```
{    float a[]={100,200,300,400,500};
```

```
    float *p[]={&a[0],&a[1],&a[2],&a[3],&a[4]};
```

```
    int i;
```

```
    for(i=0;i<5;i++)
```

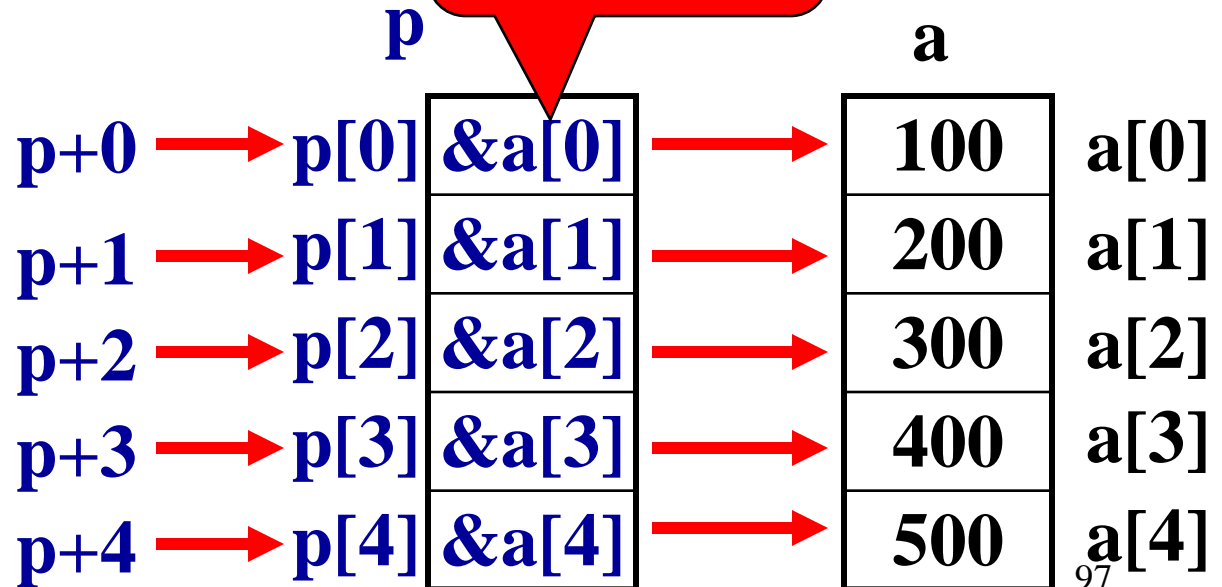
```
        cout<<*p[i]<<'\t';
```

```
    cout<<endl;
```

```
}
```

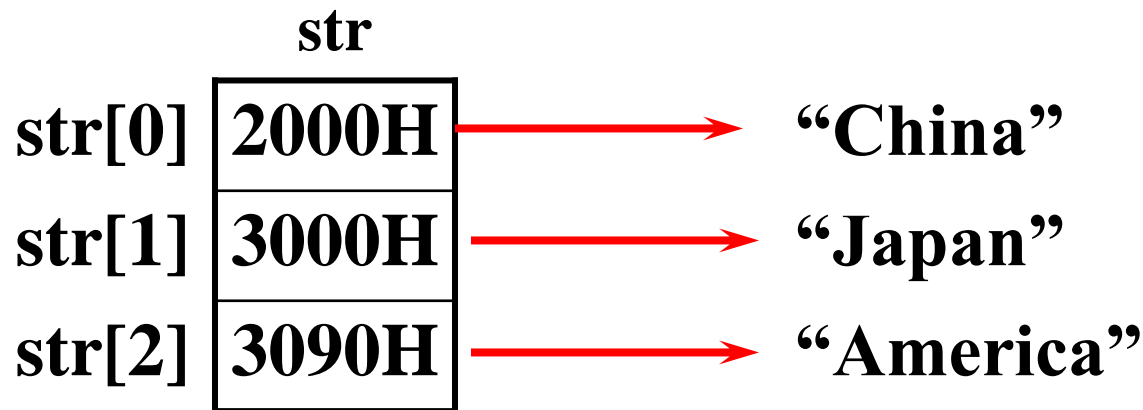
```
*p[i]=*(*(p+i) )
```

**p数组元素
内放地址**



常用字符指针数组，数组中的元素指向字符串首地址，这样比字符数组节省空间。

```
char *str[]={"China", "Japan", "America"};
```



str[1][4] $*(*(str+1)+4)$ **n**

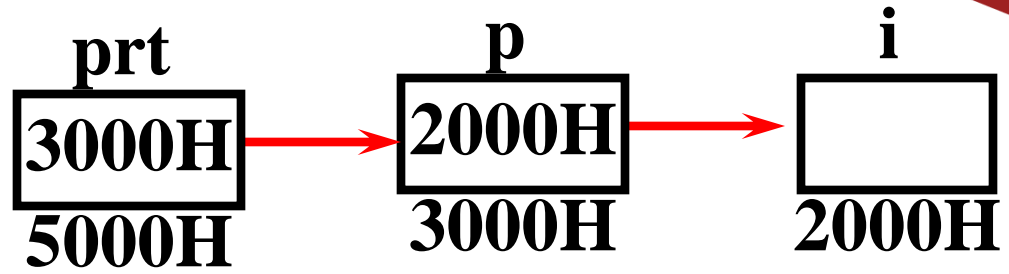
str[2] $(*(str+2)+0)$ **A**

str $*(*(str+0)+0)$ **C

指向指针的指针变量

```
int i,*p;
```

```
p=&i;
```



同样，p也有地址，可以再引用一个指针变量指向它。

```
prt=&p; p=&i
```

```
int i, *p, **prt;
```

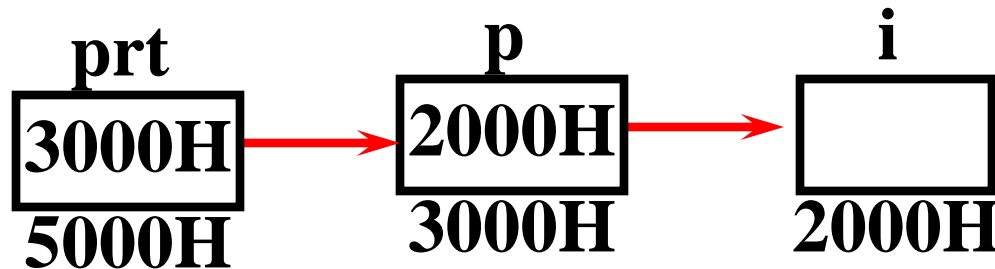
称prt为指向指针的指针变量。其基类型是指向整型数据的指针变量，而非整型数据。

`p=&i; prt=&p; prt=&i;`

`*p=i;`

`**prt=i;`

非法,基类
型不符



指针数组作main()函数的形参

程序是由main()函数处向下执行的。main函数也可以带参数。

其它函数由main函数调用的，即在程序中调用的，但main函数却是由DOS系统调用的，所以main函数实参的值是在DOS命令行中给出的，是随着文件的运行命令一起给出的。

可执行文件名 实参1 实参2 实参n

可执行文件S9_16.EXE

S9_16 CHINA JAPAN AMERICAN<CR>

main函数形参的形式:

main(int argc, char * argv[])

main(int argc, char **argv)

argc为命令行中参数的个数（包括文件名）；

argv为指向命令行中参数（字符串）的指针数组。

S9_16 CHINA JAPAN AMERICAN<CR>

文件名

实参1

实参2

实参3

argc=4

argv →

argv

argv[0]

→ **“S9_16.EXE”**

argv[1]

→ **“CHINA”**

argv[2]

→ **“JAPAN”**

argv[3]

→ **“AMERICAN”**

S9_16 CHINA JAPAN AMERICAN<CR>

文件名

实参1

实参2

实参3

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

S9_16.EXE

```
{ while (argc>1)
```

argc=4

CHINA

```
{ cout<<*argv<<endl;
```

JAPAN

```
++argv;
```

argv

argv

argv[0]

→ “S9_16.EXE”

argv[1]

→ “CHINA”

argv[2]

→ “JAPAN”

argv[3]

→ “AMERICAN”

```
--argc;
```

```
}
```

```
}
```

小结

1、指针变量可以有空值，即指针变量不指向任何地址。

```
int *p;    #include <iostream>    #define NULL 0  
p=0;      int *p; p=NULL;
```

2、两指针可以相减，不可相加。若要进行相减运算，则两指针必须指向同一数组，相减结果为相距的数组元素个数

```
int  a[10],*p1,*p2;           p2-p1 :  9  
p1=a;  p2=a+9;
```

3、指向同一数组的两个指针变量可以比较大小： $p2 > p1$

在内存动态分配存储空间

在定义变量或数组的同时即**在内存为其开辟了指定的固定空间。**

```
int n, a[10];
```

```
char str[100];
```

一经定义，即为固定地址的空间，在内存不能被别的变量所占用。

在程序内我们有时需要根据实际需要开辟空间，如输入学生成绩，但每个班的学生人数不同，一般将人数定得很大，这样占用内存。

```
#define N 100
```

```
.....
```

```
float score[N][5];
```

```
cin>>n;
```

```
for(int i=0;i<n;i++)
```

```
    for(j=0;j<5;j++)
```

```
        cin>>score[i][j];
```

```
.....
```

无论班级中有多少个学生，程序均在内存中开辟 100×5 个实型数空间存放学生成绩，造成内存空间的浪费。

如何根据需要在程序的运行过程中动态分配存储空间?

```
int n;
```

```
cin>>n;
```

```
float score[n][5];
```

错误! 数组的维数
必须是常量

利用 **new** 运算符可以在程序中动态开辟内存空间。

```
new 数据类型[单位数];
```

```
new int[4];
```

在内存中开辟了4个int型的数据空间，即16个字节

`new int;`

在内存中开辟出四个字节的空间

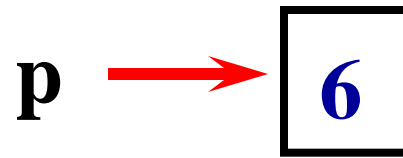
`new` 相当于一个函数，在内存开辟完空间后，返回这个空间的首地址，这时，**这个地址必须用一个指针保存下来**，才不会丢失。

`int *p;`

`p=new int;`



`*p=6;`



`new`开辟的空间

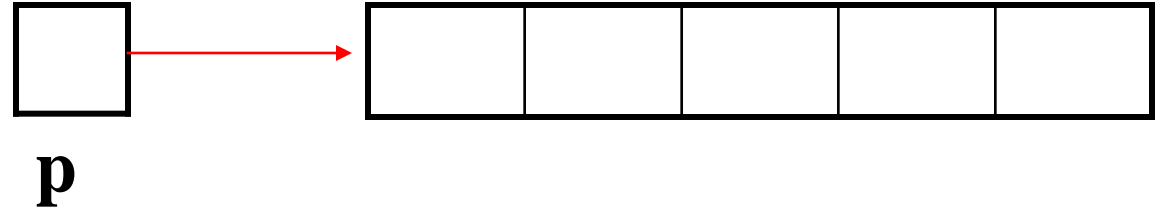
可以用**`*p`**对这个空间进行运算。

同样，利用new运算符也可以开辟连续的多个空间(数组)。

```
int n,* p;
```

```
cin>>n;
```

```
p=new int[n];
```



`p`指向新开辟空间的首地址。

```
for(int i=0;i<n;i++)
```

```
    cin>>p[i];
```

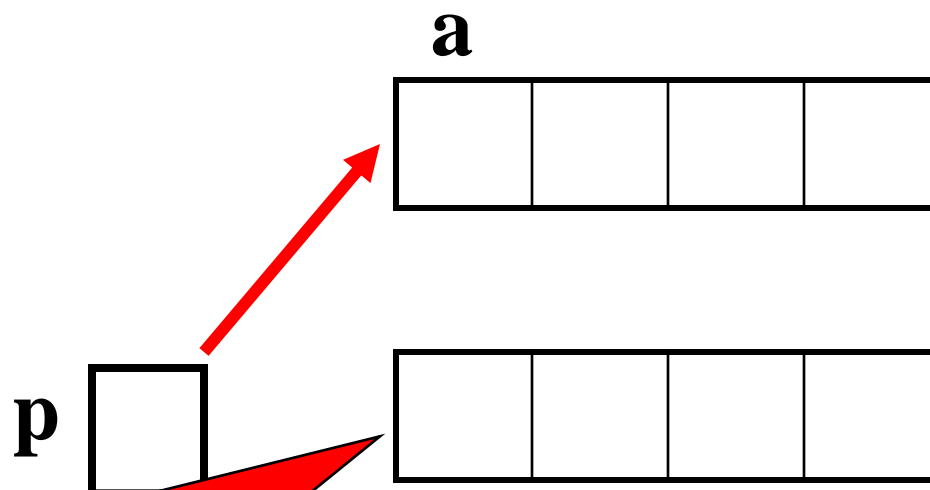
可以用`p[i]`的形式来引用新开辟的内存单元。

注意：用new开辟的内存单元没有名字，指向其**首地址的指针**是引用其的唯一途径，若指针变量重新赋值，则用new开辟的内存单元就在内存中“丢失”了，别的程序也不能占用这段单元，直到重新开机为止。

```
int * p, a[4];
```

```
p=new int[4];
```

```
p=a;
```



该段内存由于失去了“名字”，再也无法引用

new开辟的单元

用 new 运算符分配的空间，不能在分配空间时进行初始化。

同样，用new开辟的内存单元如果程序不“主动”收回，那么这段空间就一直存在，直到重新开机为止。

delete运算符用来将动态分配到的内存空间归还给系统，使用格式为：

```
delete p;
```

```
int *point;
```

```
point=new int;
```

```
.....
```

```
delete point;
```

注意：在此期间，point指针不能重新赋值，只有用new开辟的空间才能用delete收回。

delete也可以收回用new开辟的连续的空间。

```
int *point;
```

```
cin>>n;
```

```
point=new int[n];
```

```
.....
```

```
delete []point;
```

当内存中没有足够的空间给予分配时，new 运算符返回空指针NULL（0）。

以下程序求两个整数的大者，请填空。

```
void main(void )
```

```
{ int *p1, *p2;
```

```
    p1= new int ;    //开辟空间
```

```
    p2= new int ;
```

```
    cin>> *p1>>*p2 ;
```

```
    if (*p2>*p1)    *p1=*p2;
```

```
    delete p2;
```

```
    cout<<"max="<< *p1 <<endl;
```

引用

对变量起另外一个名字（外号），这个名字称为该变量的引用。

<类型> &<引用变量名> = <原变量名>;

其中**原变量名**必须是一个已定义过的变量。如：

```
int max ;
```

```
int &refmax=max;
```

refmax并没有重新在内存中开辟单元，只是**引用****max**的单元。**max**与**refmax****在内存中占用同一地址**，即同一地址两个名字。

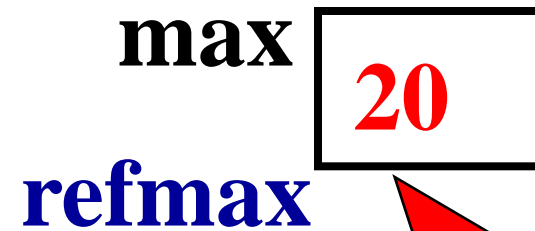
```
int max ;
```

```
int &refmax=max;
```

```
max=5 ;
```

```
refmax=10;
```

```
refmax=max+refmax;
```



max与refmax同一地址

对引用类型的变量，说明以下几点：

1、引用在定义的时候要初始化。

int &refmax;

错误，没有具体的引用对象

int &refmax=max;

max是已定义过的变量

2、对引用的操作就是对被引用的变量的操作。

3、引用类型变量的初始化值不能是一个常数。

如：**int &ref1 = 5;** // 是错误的。

int &ref=i;

4、引用同变量一样有地址，可以对其地址进行操作，即将其地址赋给一指针。

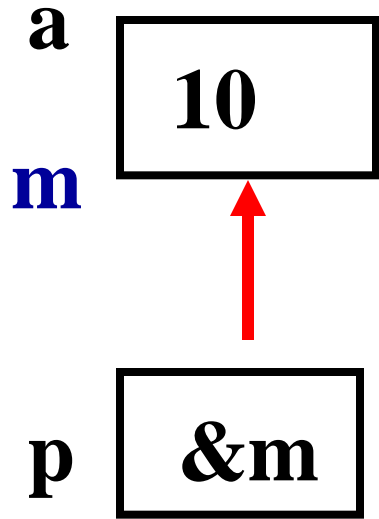
```
int  a, *p;
int  &m=a;
```

&是变量的引用

```
p=&m;
```

&是变量的地址

```
*p=10;
```



指针与引用的区别：

- 1、指针是通过地址**间接**访问某个变量，而引用是通过别名**直接**访问某个变量。
- 2、引用必须初始化，而**一旦被初始化后不得再作为其它变量的别名。**

当&a的前面有**类型符**时（如int
&a），它必然是对引用的声明；如
果前面无类型符（如cout<<&a），则
是取变量的地址。

引用与函数

引用的用途主要是用来作函数的参数或函数的返回值。

引用作函数的形参，实际上是在被调函数中对实参变量进行操作。

```
void change(int &x, int &y)//x,y是实参a,b的别名
```

```
{ int t;
```

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

```
}
```

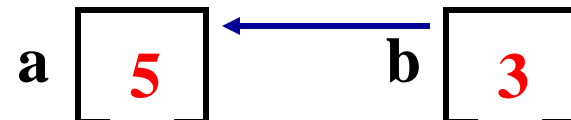
```
void main(void)
```

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

```
    change(a,b); //实参为变量
```

```
    cout<<a<<'\\t'<<b<<endl;
```

```
}
```



输出： 5 3

引用作为形参，实参是**变量而不是地址**，这与指针变量作形参不一样。

形参为整型引用

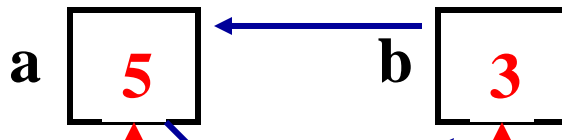
```
void change(int &x, int &y)
{   int t;
    t=x; x=y; y=z;
}

void main(void)
{   int a=3,b=5;
    change(a,b); //实参为变量
    cout<<a<<'\t'<<b<<endl;
}
```

形参为指针变量

```
void change(int *x, int *y)
{   int t;
    t=*x; *x=*y; *y=z;
}

void main(void)
{   int a=3,b=5;
    change(&a,&b); //实参为地址
    cout<<a<<'\t'<<b<<endl;
}
```



函数的返回值为引用类型

可以把函数定义为引用类型，这时函数的返回值即为某一变量的引用（别名），因此，它相当于返回了一个变量，所以可对其返回值进行赋值操作。这一点类同于函数的返回值为指针类型。

```
int a=4;
```

```
int &f(int x)
```

函数返回a的引用，即a的别名

```
{ a=a+x;
```

```
    return a;
```

```
}
```

```
void main(void)
```

```
{ int t=5;
```

输出 9 (a=9)

```
    cout<<f(t)<<endl;
```

```
    f(t)=20;
```

先调用，再赋值 a=20

```
    cout<<f(t)<<endl;
```

输出25 (a=25)

```
    t=f(t);
```

先调用，再赋值 t=30

```
    cout<<f(t)<<endl; }
```

输出60 (a=60)

我们都知道，函数作为一种程序实体，它有名字、类型、地址和存储空间，一般说来函数不能作为左值（即函数不能放在赋值号左边）。但如果将函数定义为返回引用类型，因为返回的是一个变量的别名，就可以将函数放在左边，即给这个变量赋值。

Thank you !