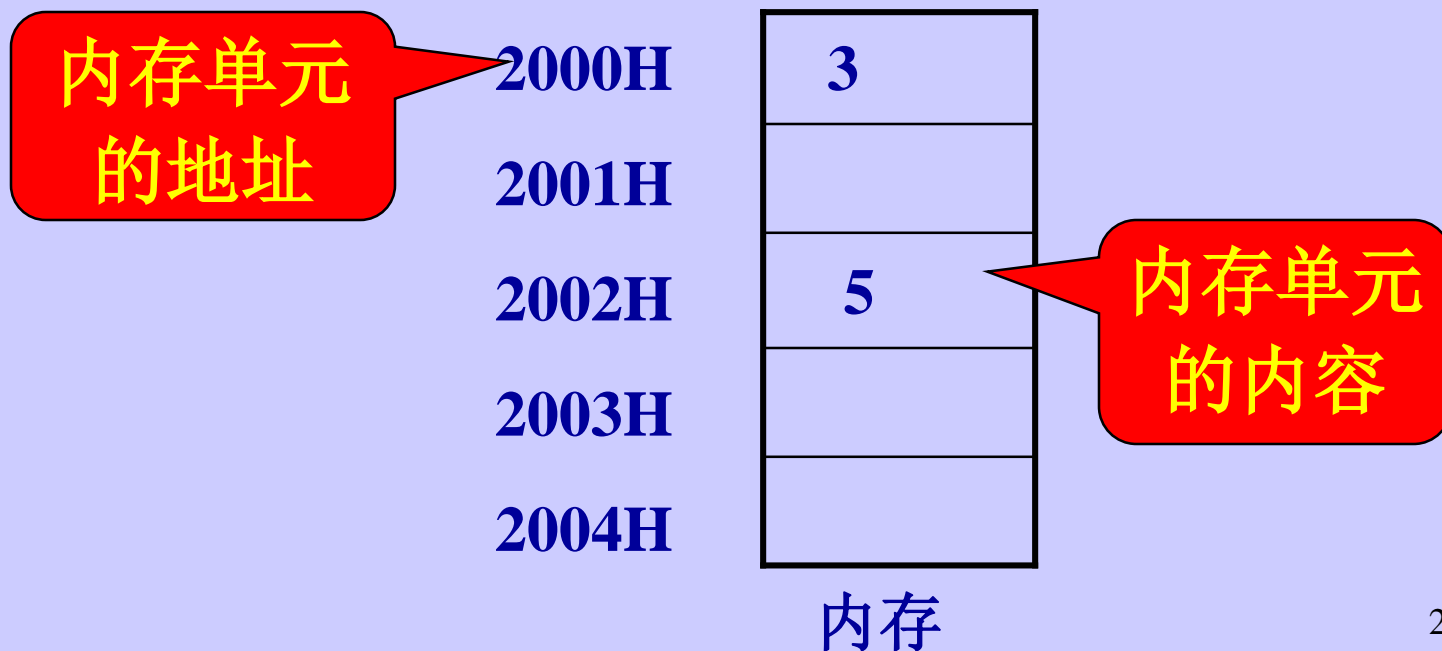


第八章 指针和引用

指针的概念

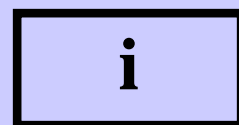
数据在内存中是如何存取的？

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**。



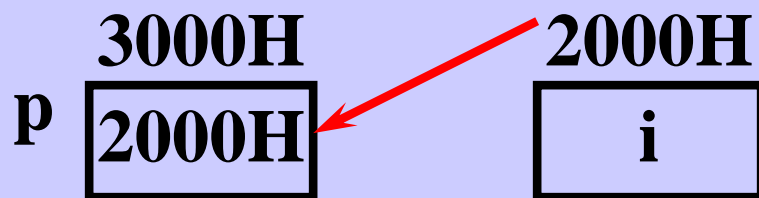
1、直接访问

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。

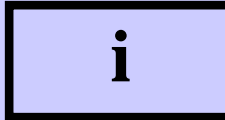


一个变量的地址称为该变量的指针。

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** 

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。

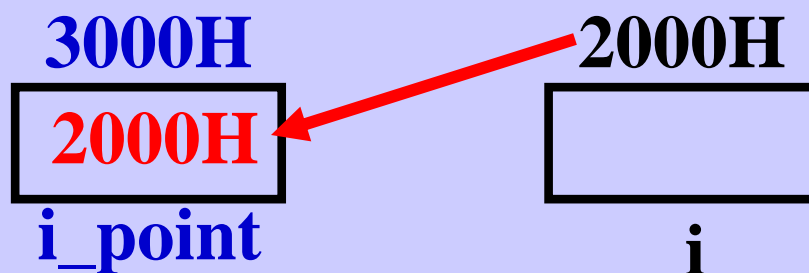
指针变量定义的一般形式为：

类型标识符 *变量名  
`int *i_point;`

指针**变量**同样也可以赋值：

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

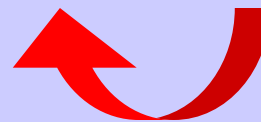
```
i_point=&i;
```



也可以在定义**指针变量**时赋初值：

```
int i;
```

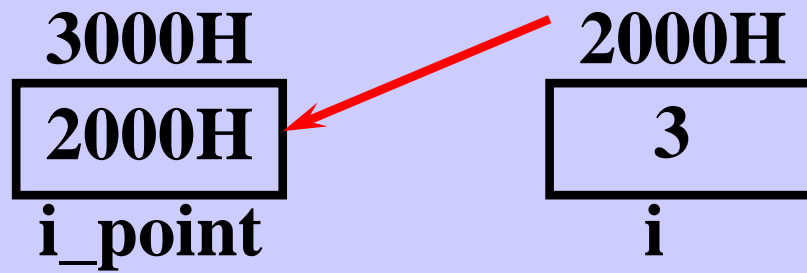
```
int *i_point=&i;
```



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”₅。



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

表示类型 (Indicates type)

`*i_point=3;`

表示指向 (Indicates pointing)

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)
```

非法

```
{ int a=10, b=100;
```

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

指针变量赋值

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

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

10

100

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

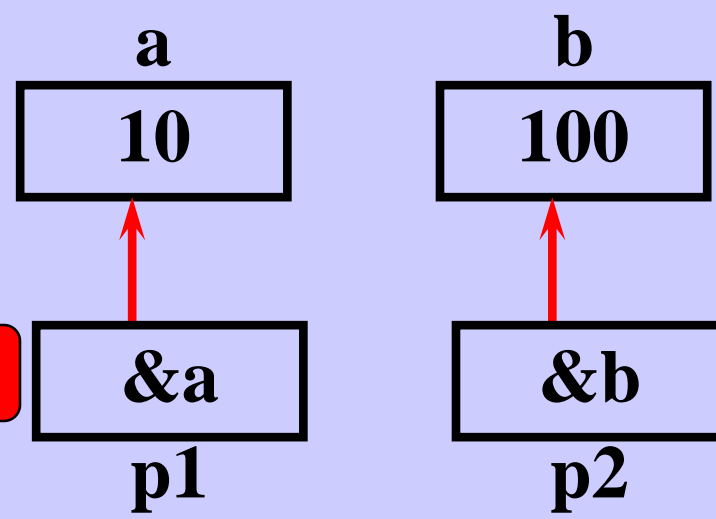
10

100

```
}
```

表示
指向

指针变量引用



```
void main(void)
```

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

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

指针变量赋值

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

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

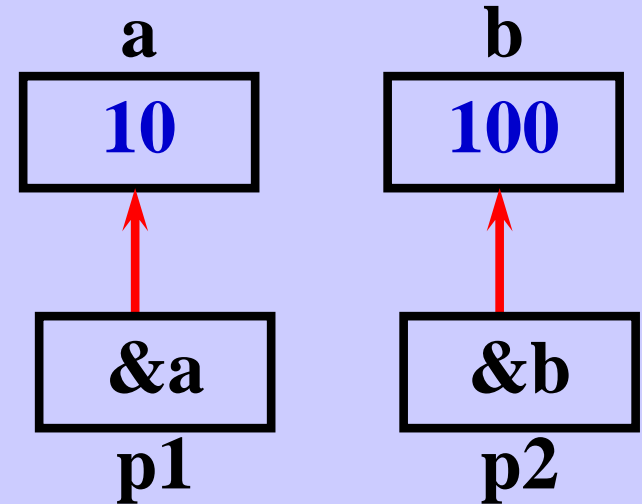
通过指针对
变量赋值

```
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;
```

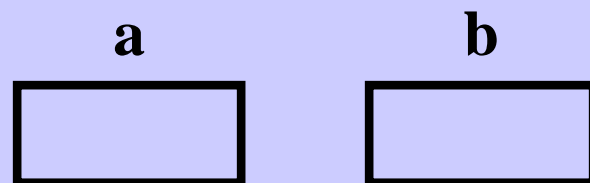
```
cout<<a<<'\t'<<b<<endl;
```

```
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



随机

p1

随机

p2

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;
```

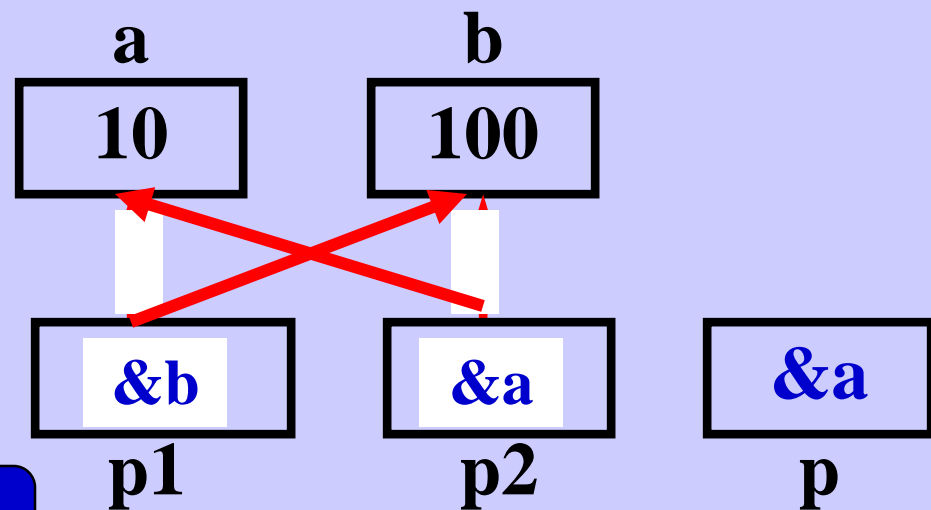
```
if (a<b)
```

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

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

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

```
}
```



交换地址

10 100

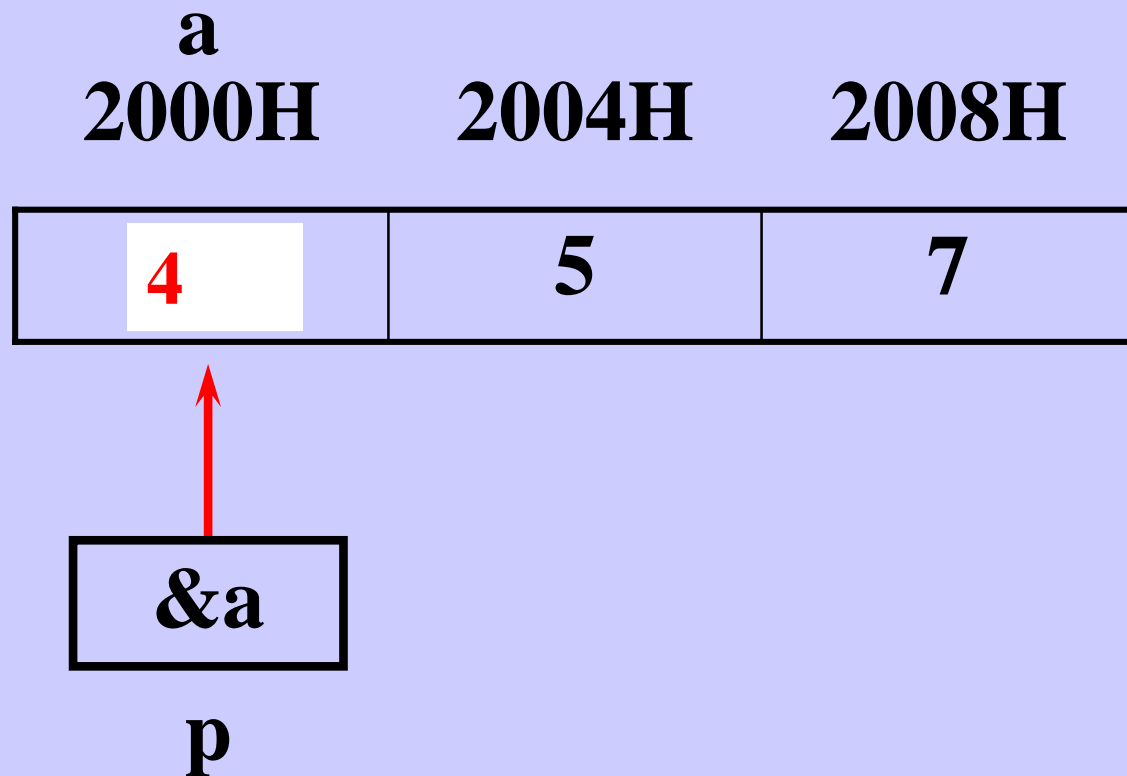
100 10

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;
```

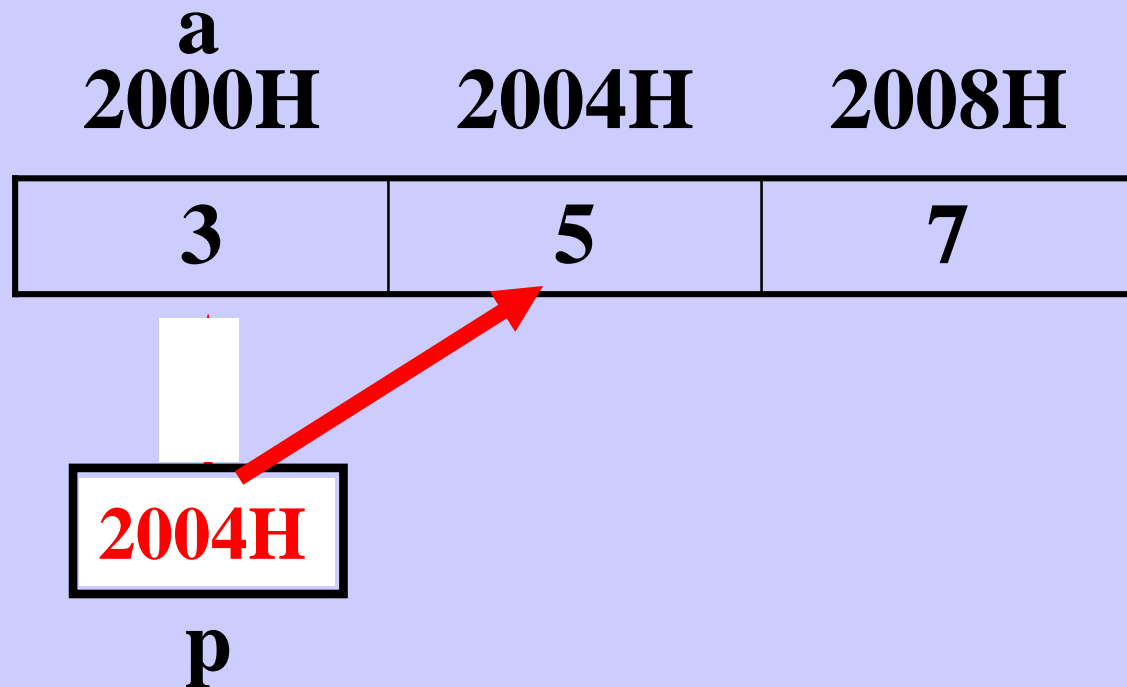


(*p)++; 相当于**a++**。表达式为**3**, **a=4**

++, --, * 优先级相同，都是右结合性。

```
int a=3, *p;
```

```
p=&a;
```

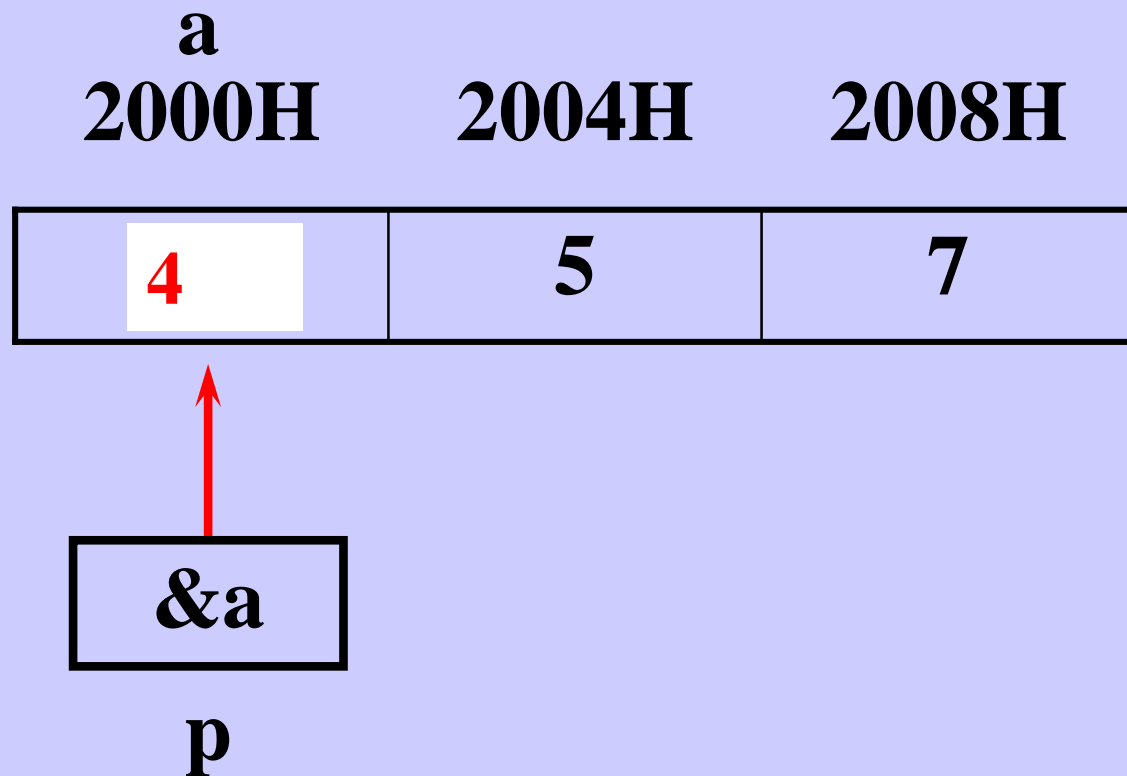


*p++; *(p++)首先*p，然后p=p+1,指针指向下一个int单元 表达式为3, p=2004H。

++, --, * 优先级相同，都是右结合性。

```
int a=3, *p;
```

```
p=&a;
```



```
++*p
```

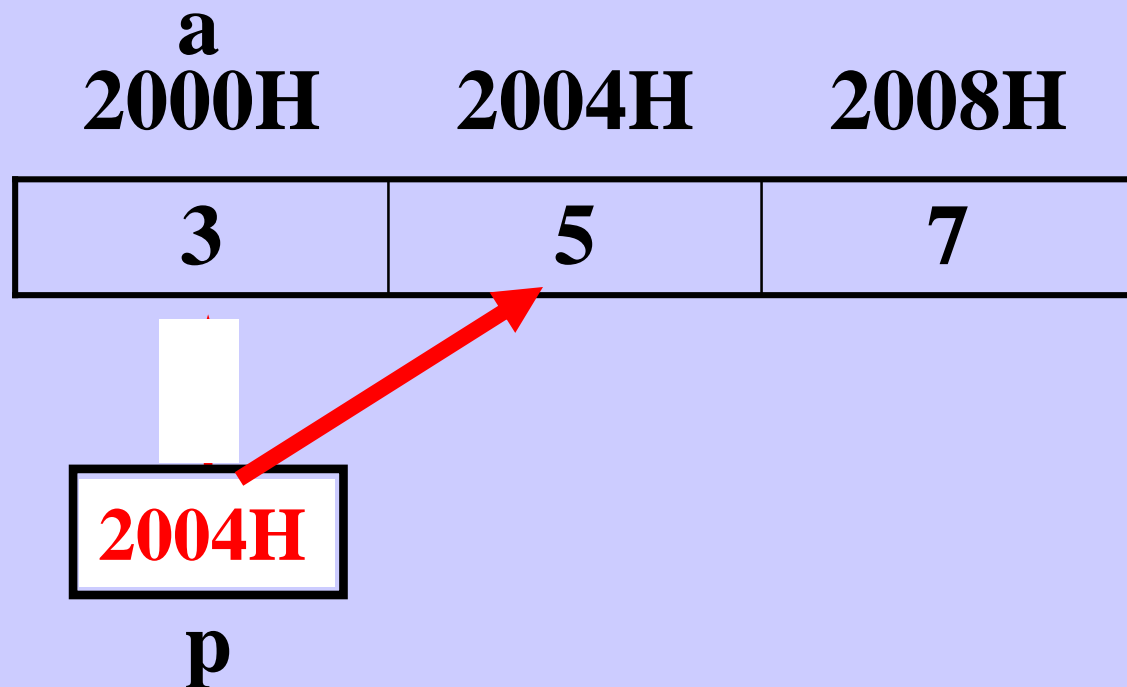
```
++(*p)
```

```
*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 *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;
```

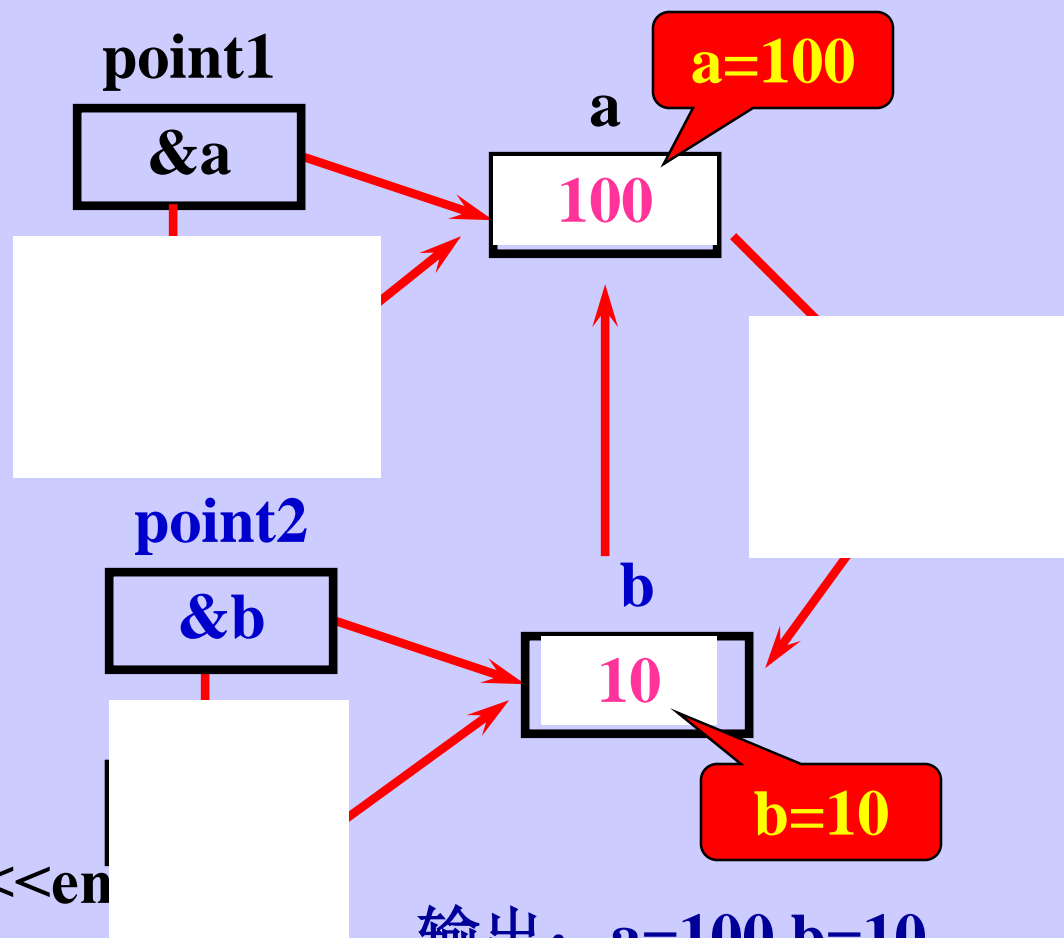
```
  if (a<b)
```

```
      swap (point1, point2);
```

```
  cout<<"a="<<a<<" ,b="<<b<<endl;
```

```
  cout<<*point1<<" ,"<<*point2<<endl;
```

```
}
```



输出: a=100,b=10

100,10

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;
```

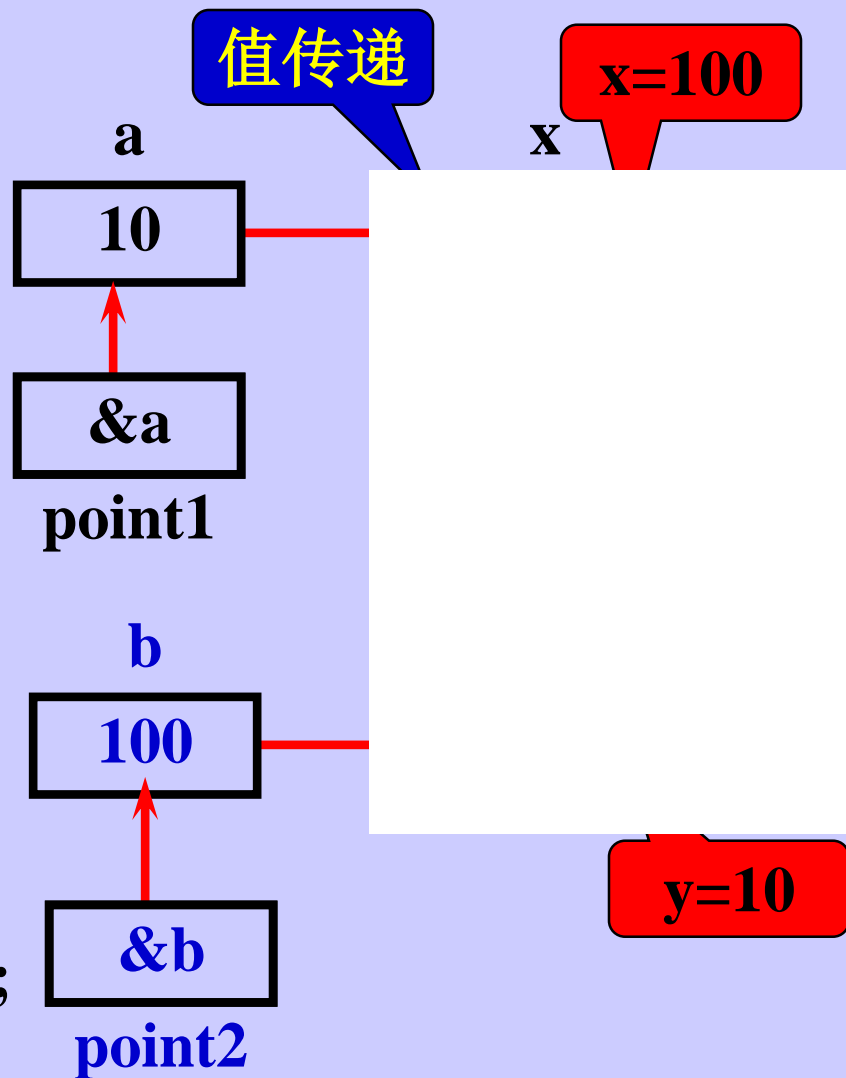
```
  if (a<b)
```

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

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

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

```
}
```



输出: a=10,b=100

10,100

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。

```
void grt(int *x , int *y , int *z)
```

```
{  cout<< ++*x<<'<< ++*y<<'<<*(z++)<<endl;}
```

```
int  a=10, b=40, c=20;
```

```
void main(void)
```

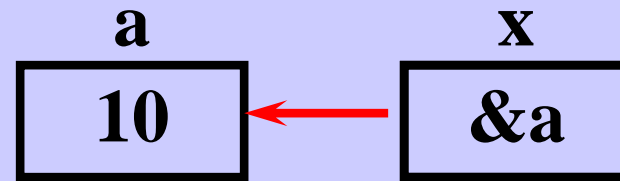
```
{  prt (&a, &b, &c);
```

11, 41, 20

```
    prt (&a, &b, &c);
```

12, 42, 20

```
}
```



++*x: *x=*x+1

***(z++): *z ; z=z+1**

```
int s( int *p)
```

```
{  int sum=10;
```

```
    sum=sum + *p;
```

```
    return sum;
```

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

```
{  int a=0, i, *p, sum;
```

```
    for (i=0; i<=2; i++)
```

```
    {  p=&a;
```

```
        cin>>*p;
```

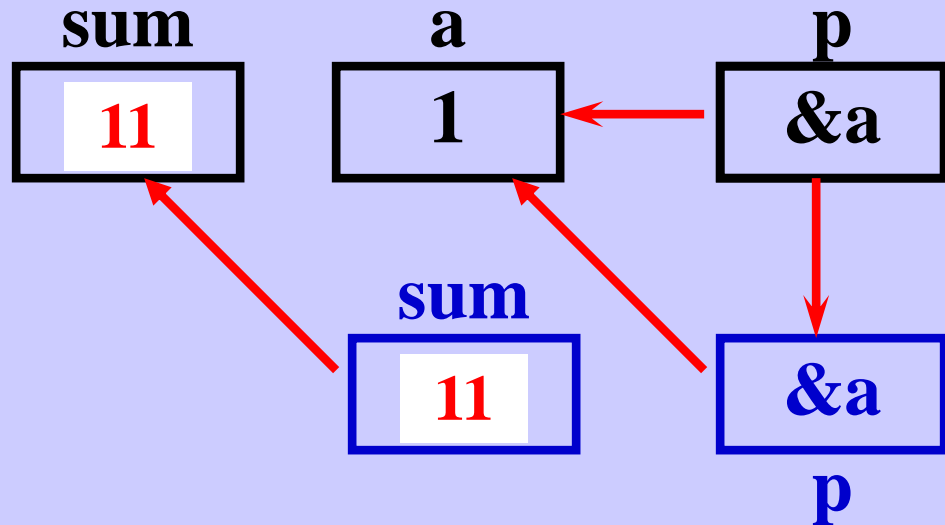
```
        sum=s(p);
```

```
        cout<<"sum="<<sum<<endl;
```

```
    }
```

```
}
```

输入: 1 3 5<CR>



sum=11

sum=13

sum=15

```
sub( int *s)
```

```
{ static int t=0;
```

```
  t=*s + t;
```

```
  return t;
```

```
}
```

```
void main(void)
```

```
{ int i, k;
```

```
  for (i=0; i<4; i++)
```

```
  { k=sub(&i);
```

```
    cout<<"sum="<<k<<"\t";
```

```
  }
```

```
  cout<<"\n";
```

```
}
```

```
i=0  t=*s+t=0  k=0  sum=0
```

```
i=1  t=*s+t=1  k=1  sum=1
```

```
i=2  t=*s+t=3  k=3  sum=3
```

```
i=3  t=*s+t=6  k=6  sum=6
```

```
sum=0
```

```
sum=1
```

```
sum=3
```

```
sum=6
```

```
int *p;
```

```
void main(void)
```

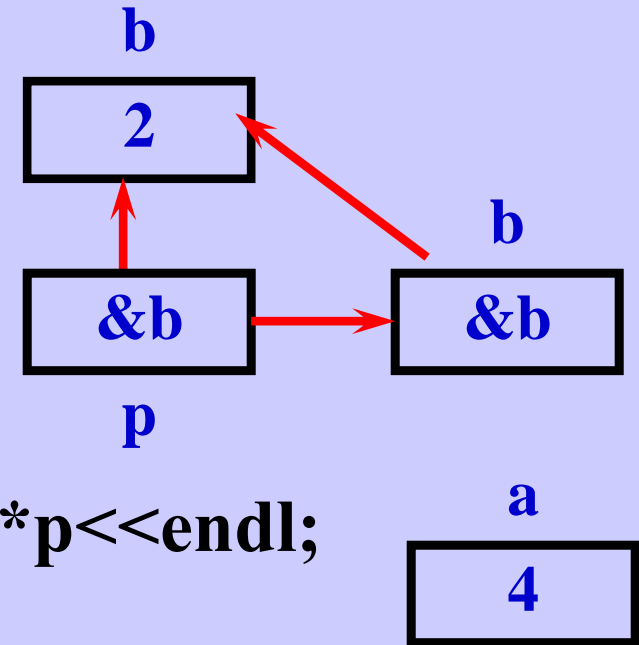
```
{ int a=1, b=2, c=3;
```

```
  p=&b;
```

```
  pp(a+c, &b);
```

```
  cout<<"(1)"<<a<<" "<<b<<" "<<*p<<endl;
```

```
}
```



```
pp(int a, int *b)
```

```
{ int c=4;
```

```
  *p=*b+c;
```

```
  a=*p-c;
```

```
  cout<<"(2)"<<a<<" "<<*b<<" "<<*p<<endl;
```

```
}
```

(2) 2 6 6

(1) 1 6 6

$*p = *b + 4 = 2 + 4 = 6$

$a = 6 - c = 2$

举例：最大最小值、方程根

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

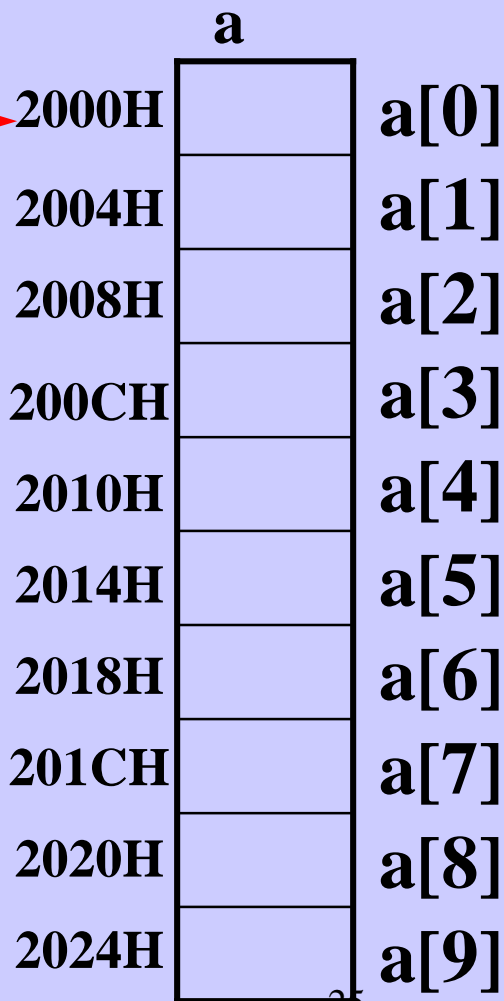
```
p=a;
```

直接用数组名赋值

&a[0]

p

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;
```

```
*p=1;
```

```
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[0]

p

2000H

2004H

2008H

200CH

2010H

2014H

2018H

201CH

2020H

2024H

a

1

2

a[0]

a[1]

a[2]

a[3]

a[4]

a[5]

a[6]

a[7]

a[8]

a[9]

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

$*(a+1)=2;$

$*(a+1)$ 与 $a[1]$ 等同。

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

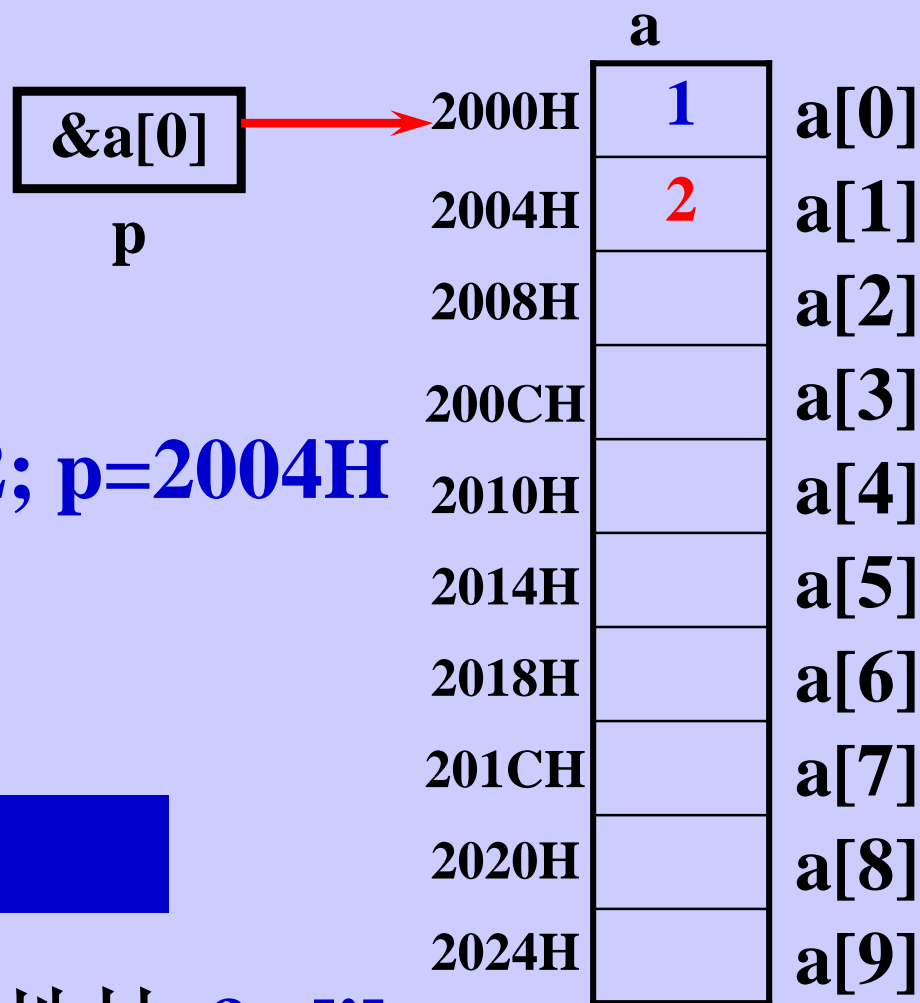
$*++a=2;$

错误

a 为常量，不可赋值。

$p+i$ 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;
```

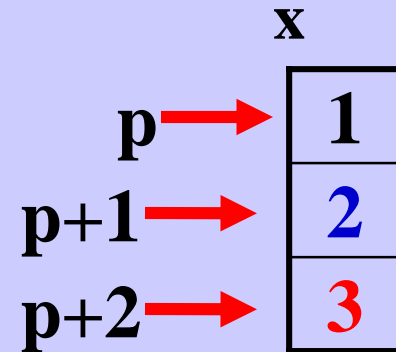
```
    s=1; p=x;
```

```
    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**

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]

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

输入10个整数，将其中最小的数与第一个数对换，把最大的数与最后一个数对换。写3个函数：①输入10个数；②进行处理；③输出10个数。

编写函数 `int fun(int x, int *pp)`,其功能是，求出能整除`x`且不是偶数的各整数，并按照从小到大的顺序放在`pp`指向的内存中，函数返回值为这些整数的个数。若`x`的值为30，数组中的数为1，3，5，15，函数返回4。


```
int fun(int x, int *pp)
{ int k=0;
  for(int i=1;i<x;i++)
  { if(i%2&& x%i==0)
    { *(pp++)=i;
      k++;
    }
  }
  return k;
}
```

```
void main(void)
{ int a[1000],x,n;
  cin>>x;
  n=fun(x,a);
  for(int i=0;i<n;i++)
    cout<<a[i]<<'\\t';
  cout<<endl;
}
```

输入一行字符串，将字符串中所有下标为奇数位置上的字母转换为大写（若不是小写字符则不必转换）。

```
void change(char *pchar)
```

```
{ while(*pchar)
```

```
{ if(*pchar>='a' &&*pchar<='z')
```

```
    *pchar=*pchar-32;
```

```
    pchar++;
```

```
    if(*pchar==0)
```

```
        break;
```

```
    pchar++;
```

```
}
```

```
}
```

```
void main(void)
```

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

```
    cin.getline(str,100);
```

```
    change(str);
```

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

```
}
```

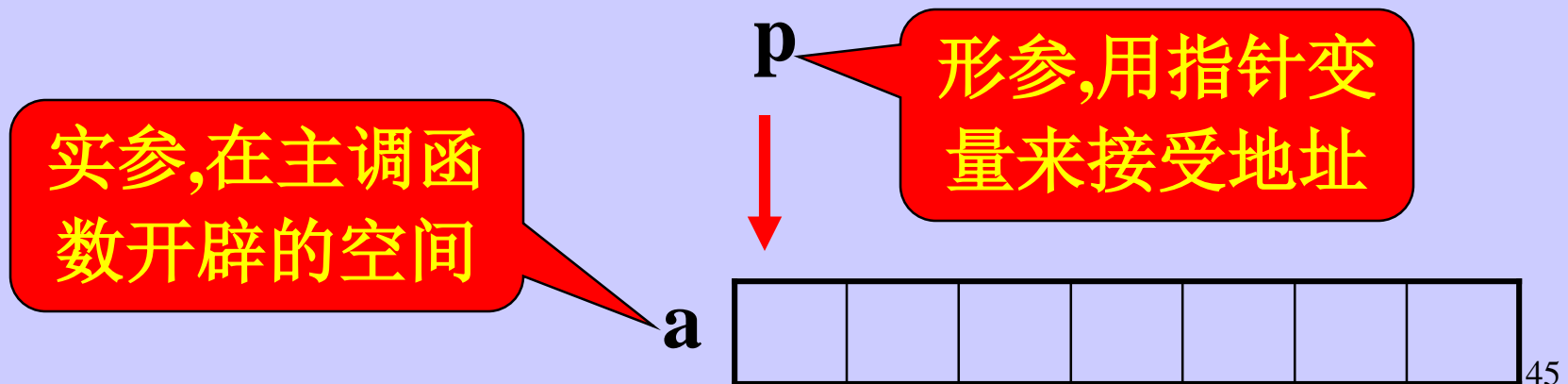
数组名作函数参数

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]

a[1]=*(a+1) *(a+1)+2=&a[1][2]

***(*(a+1)+2)=a[1][2]**

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

(*(a+1))[1]=*(*(a+1)+1)=a[1][1]

***(a+1)[1]=* ((a+1)[1])=*(*((a+1)+1))=** (a+2)=a[2][0]**

注意二维数组的各种表示法，a为常量。

	a	1900H	1904H	1908H	190CH
1900H	a[0]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1910H	a[1]	a[1][0]	a[1][1]	a[1][2]	a[1][3]
1920H	a[2]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

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

设数组的首地址为1900H， 则：

a为 1900H

*a为 1900H

a+2为 1920H

*a+2为 1908H

*(a+1)+2为 1918H

**a为 1

*(a+9)为 19

(a+1)[1]为 1920H

```
void main(void)
```

p+1 a[0]+1

```
{ static int
```

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



p

```
int *p; a[0]是列指针
```

```
for(p=a[0]; p<a[0]+12 ; p++)
```

```
{ if((p-a[0])%4==0) cout<<endl;
```

a+1

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

类型不匹配!

```
} for(p=a; p<a+12 ; p++)
```

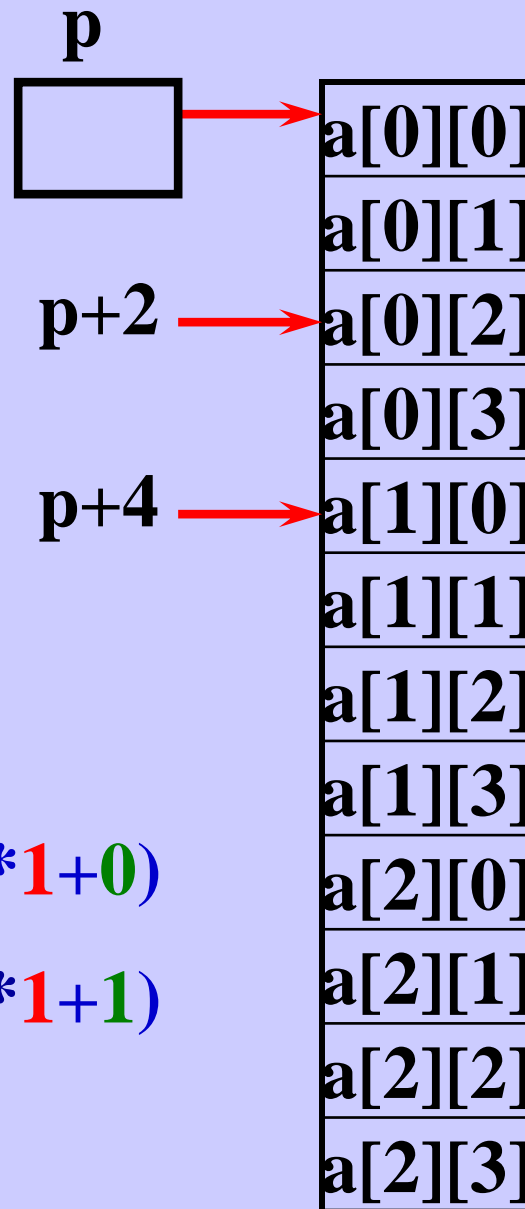
```
}
```

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

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]

`int a[3][4],*p;`

如何用 **p** 表示二维数组中的任一元素？



`p=a[0]` `*p` `*(a[0]+0)` `a[0][0]`

`p+2` → `a[0][2]`

`p+1` `*(a[0]+1)` `a[0][1]`

`p+4` → `a[1][0]`

`p+2` `*(a[0]+2)` `a[0][2]`

`p+3` `*(a[0]+3)` `a[0][3]`

`p+4` `*(a[0]+4)` `a[1][0]` **`*(p+4*1+0)`**

`p+5` `*(a[0]+5)` `a[1][1]` **`*(p+4*1+1)`**

`a[i][j]` **`*(p+4*i+j)`** `*(p+m*i+j)`

m为第二维的维数。是个常数

```
fun(int *q1, int *q2, int *q3)
```

```
{ *q3=*q2+*q1; }
```

```
void main ( void)
```

```
{ int i , j, a[3][3]={1,1},*p1,*p2,*p3;
```

```
p1=a[0], p2=a[0]+1,p3=a[0]+2;
```

```
for(i=2;i<9;i++)
```

```
    fun(p1++, p2++, p3++);
```

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

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

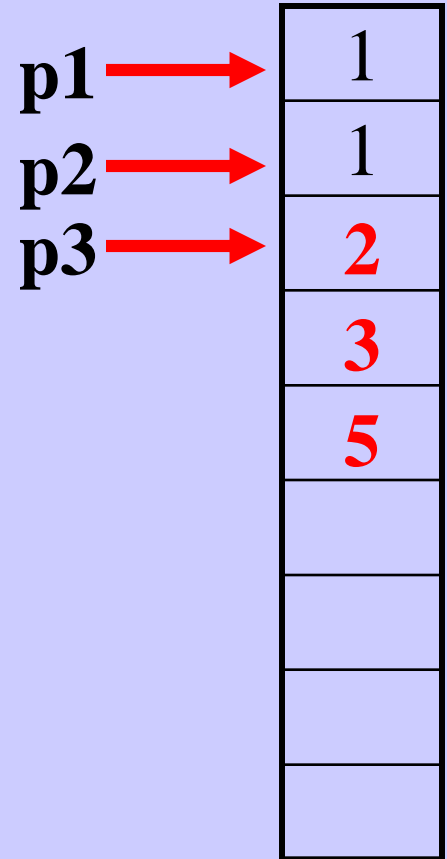
```
    { cout<<a[i][j]<<'\t';
```

```
        cout<<endl;
```

```
}
```

程序输出的第一行为1

第二行为1 第三行为2



i=2 *q3=2

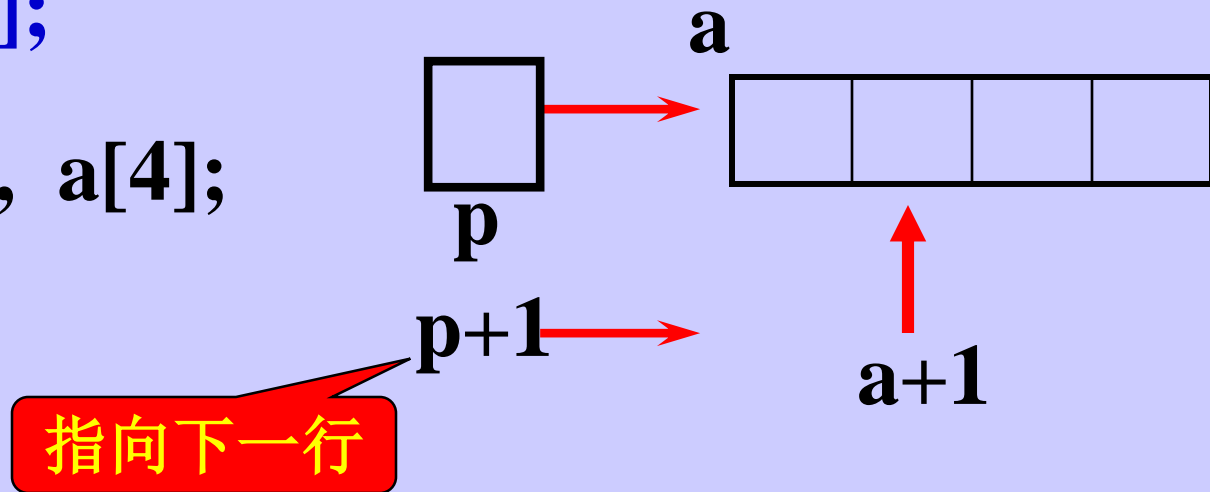
i=3 *q3=3

i=4 *q3=5

指向由m个整数组成的**一维数组**的指针变量

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

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



`p+1`指针加16个字节。`a+1`指针加4个字节。


```
int (*p)[4], a[3][4];
```

a		2000H	2004H	2008H	200CH
p	→	a[0][0]	a[0][1]	a[0][2]	a[0][3]
p+1	→	a[1][0]	a[1][1]	a[1][2]	a[1][3]
p+2	→	a[2][0]	a[2][1]	a[2][2]	a[2][3]

p为行指针，可以直接将二维数组名**a**赋给**p**。这样，**a**与**p**等同。 **p=a**

a[2][3] ***(*(a+2)+3)**

p[2][3] ***(*(p+2)+3)**

两种表示
完全等价

a为常量
p为变量

若有以下的定义和语句，则下面各个符号的正确含义是：

int a[3][4], (*p)[4];

p=a;

		a	2000H	2004H	2008H	200CH
p	→		a[0][0]	a[0][1]	a[0][2]	a[0][3]
p+1	→		a[1][0]	a[1][1]	a[1][2]	a[1][3]
p+2	→		a[2][0]	a[2][1]	a[2][2]	a[2][3]

1. **p+1** a数组第一行元素的首地址 为行指针

2. ***(p+2)** a数组第二行元素的首地址 为列指针 **&a[2][0]**

3. ***(p+1)+2** **&a[1][2]** 为列指针

4. ***(p+2)** 数据元素 ***(p+2)=a[0][2]**

若有以下的定义和语句，则对a数组元素的非法引用是：

```
int a[2][3], (*pt)[3];
```

```
pt=a;
```

1) `pt[0][0]` 2) `*(pt+1)[2]`

3) `*(pt[1]+2)` 3) `*(a[0]+2)`

`*(pt+1)[2]` 右结合性 $= *((pt+1)[2])$
 $= *(*pt+1+2)$
 $= *(*pt+3) = pt[3][0]$, 非法

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**。（举例）

求二维数组a[3][4]的平均值。

```
void main(void)
```

```
{    float score[3][4] = { {65,67,70 ,60}, {80,87,90,81},  
    {90,99,100,98} };
```

```
    float sum=0;
```

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

```
        for(int j=0;j<4;j++)
```

```
            sum=sum+score[i][j];
```

```
    cout<<"aver="<<sum/12<<endl;
```

```
void main(void)
```

```
{    float score[3][4] = { {65,67,70 ,60}, {80,87,90,81},  
    {90,99,100,98} };
```

```
    float aver;
```

```
    aver=fun1(score, 3);
```

```
    aver=fun2(*score, 12);
```

```
p 161 8.15
```

字符串的**指针**和指向字符串的**指针变量**

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

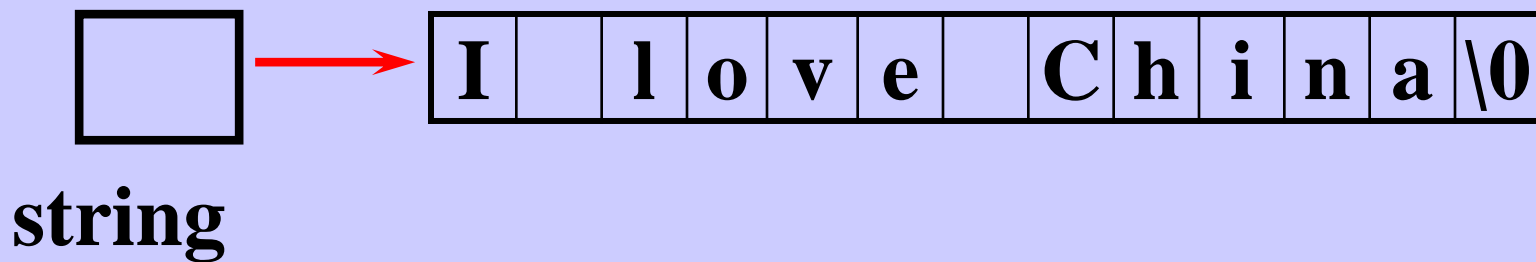
```
cin.getline(string); //从键盘输入
```


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

```
void main(void)
```

```
{ char *string;
```

```
    string="I love China";
```

```
}
```

指针变量赋值，合法

具体字符

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

```
char *string;
```

```
cin.getline(string);
```

指针未赋值就作指向运算

将字符串a复制到字符串b。

```
void main(void)
```

```
{ char a[ ]="I am a boy", b[20];
```

```
int i;
```

```
for(i=0; *(a+i)!='\0'; i++)
```

```
    *(b+i)=*(a+i);
```

```
    *(b+i)='\0';
```

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

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

```
}
```

$i=0$ $*(b+i)=*(a+i)$

$b[i]=a[i]$

$i=1$

$i=2$

a

I		a	m		a		b	o	y	\0
---	--	---	---	--	---	--	---	---	---	----



I		a							y	\0
---	--	---	--	--	--	--	--	--	---	----

b

必须以\0结束

```
void main(void)
```

```
{ char a[ ]="I am a boy", b[20];
```

```
char *p1, *p2;
```

```
int i;
```

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

```
for(; *p1!='\0'; p1++,p2++)
```

```
    *p2=*p1;
```

```
    *p2='\0';
```

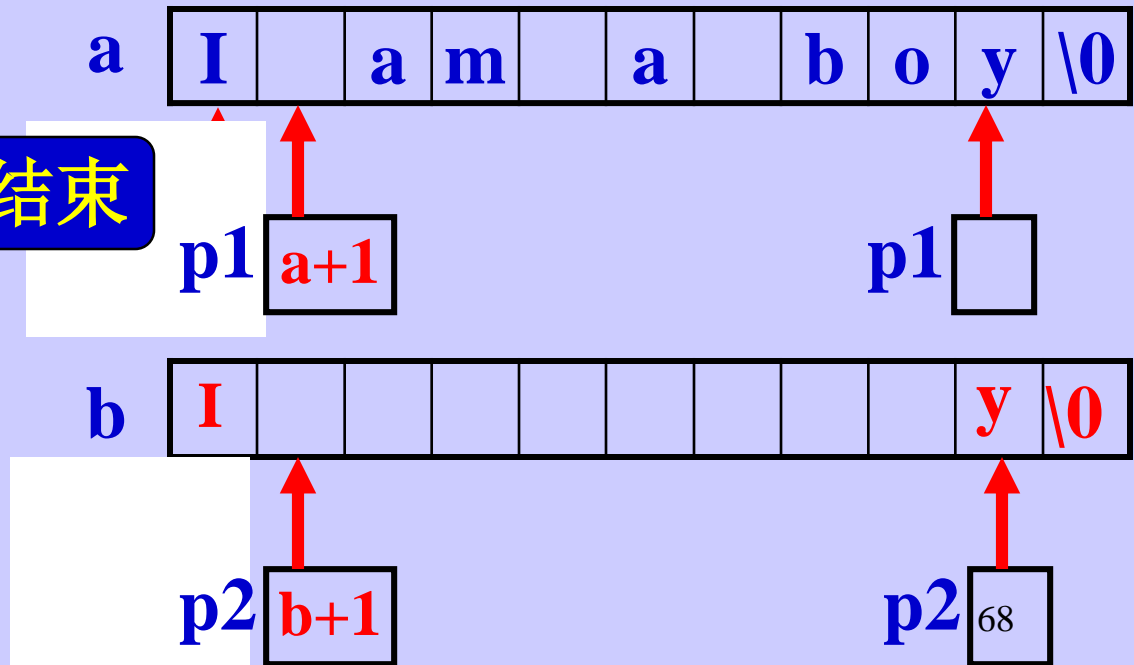
必须以\0结束

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

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

```
}
```

***p2=*p1**



```
void main(void)
```

```
{ char a[ ]="I am a boy", b[20];
```

```
char *p1, *p2;
```

```
for(; *p1!='\0'; )
```

```
int i;
```

```
*p2++=*p1++;
```

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

```
*p2='\0';
```

```
for(; *p1!='\0'; p1++,p2++)
```

```
*p2=*p1;
```

```
while(*p2++=*p1++);
```

```
*p2='\0';
```

```
for(; *p2++=*p1++ ; );
```

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

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

```
for(; (*p2++=*p1++)!='\0' ; );
```

```
}
```

以下程序判断输入的字符串是否“回文”，若是回文，输出YES。

```
void main(void)
{ char s[81], cr, *pi, *pj;
  int i, j, n;
  cin.getline(s);  n=strlen(s);
  pi=__s__; pj=__s+n-1__; //pi指向串开始, pj指向最后
  while(*pi==' ') pi++;
  while(*pj==' ') pj--;
  while( ( __pi<pj__ ) &&(*pi==*pj) )
      { pi++; pj--; }
  if((pi<pj) cout<<"NO"<<endl;
  else cout<<"YES\n";
}
```

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++)
```

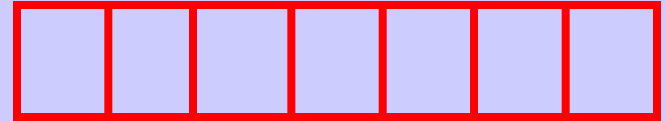
```
to[i]=from[i];
```

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

```
}
```

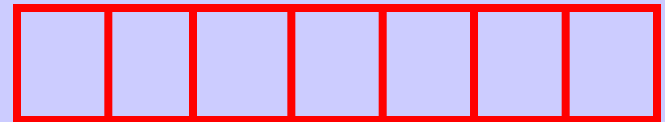
from

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"};`

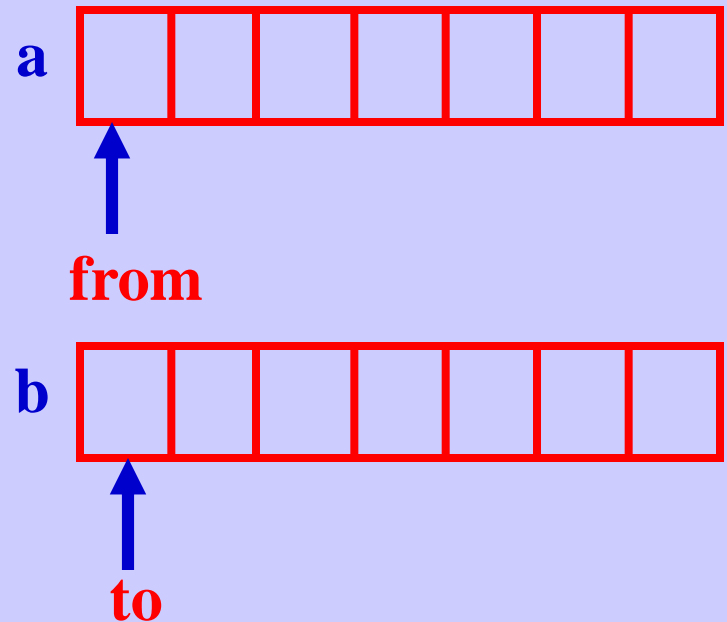
`char b[]={"You are a student"};`

`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;
```

```
}
```

当s[i]等于字符 'c'时, i前进, j不

动
输出: hane

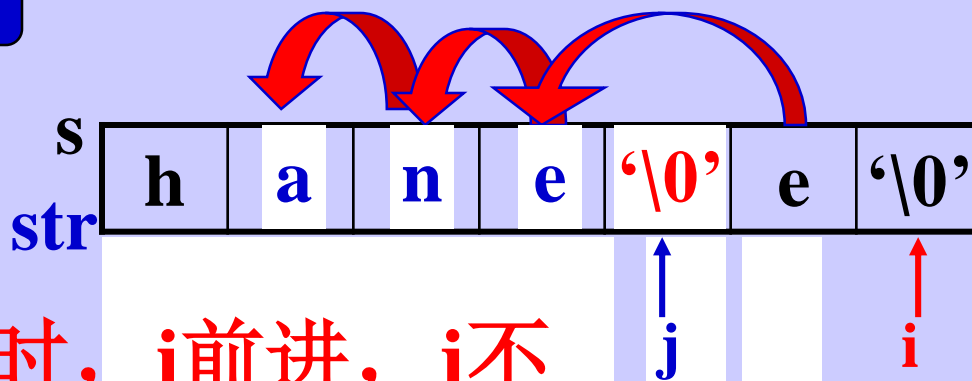
```
void main(void)
```

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

```
    fun(str);
```

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

```
}
```



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

```
{ char t;
```

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

```
}
```

```
void main(void)
```

```
{ 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;
```

```
}
```

AD

BC

BB

有一字符串，包含n个字符。写一函数，将此字符串中从第m个字符开始的全部字符复制成为另一个字符串。

```
void main(void)
```

```
{ char str1[100]={“I am a student”},str2[100];
```

```
    int m;
```

```
    cin>>m;
```

```
    fun(str1,str2,m);
```

```
    cout<<str2<<endl;
```

```
}
```

```
void fun(char *p1, char *p2, int m)
```

```
{  int n=strlen(p1);
```

原字符串长度

```
    if(n<m)
```

```
    { cout<<"No trans\n";*p2=0;  return;}
```

```
    for(int i=m, p1=p1+m-1; i<n;i++)
```

```
        *p2++=*p1++;
```

从第m个字符开始

```
    *p2='\0';
```

```
    return;
```

```
}
```

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

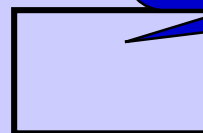
直接用函数名为指针变量赋值。

`int max (int x, int y)`

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

`}`

p



`p=max;`

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。

int (*p)(int , int);

定义

p=max;

p=min;

赋值

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

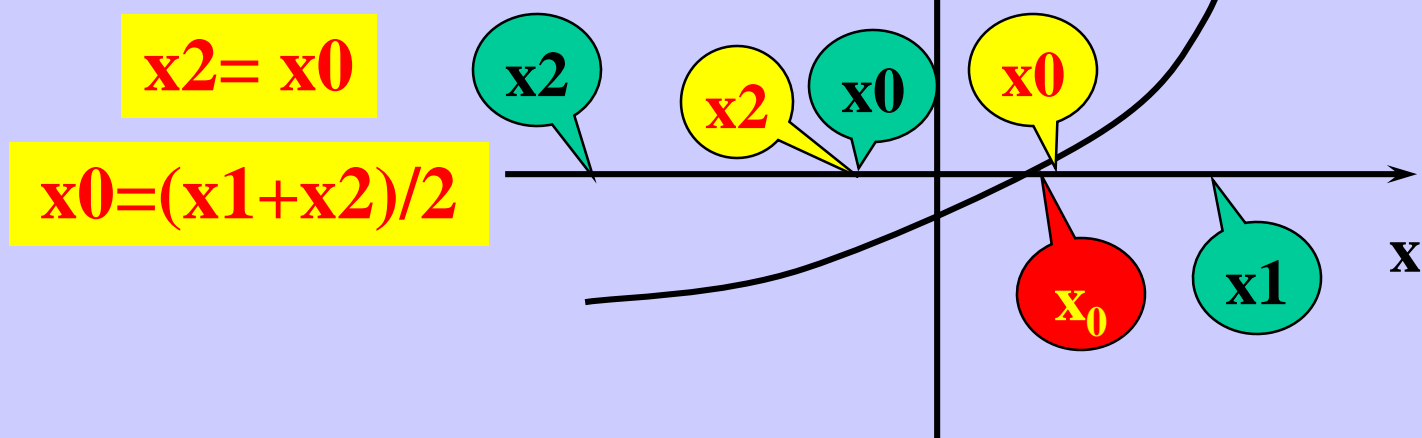
通过指针变量调用

实际上就是用p替换函数名

指向函数的指针变量作函数参数（实现通用函数）

用二分法求方程的解。 $f1(x)=x^2-3$

二分法求解方程



- 1、在 x 轴上取两点 x_1 和 x_2 , 要确保 x_1 与 x_2 之间有且只有方程唯一的解。
- 2、做 $x_0 = (x_1 + x_2) / 2$ 。
- 3、若 $|f(x_0)|$ 满足给定的精度, 则 x_0 即是方程的解, 否则, 若 $f(x_0) \cdot f(x_1) < 0$, 则方程的解应在 x_1 与 x_0 之间, 令 $x_2 = x_0$, 继续做2。同理, 若 $f(x_0) \cdot f(x_2) < 0$, 则方程的解应在 x_2 与 x_0 之间, 令 $x_1 = x_0$, 继续做2, 直至满足精度为止。

用二分法求方程的解。 $f1(x)=x^2-3$

```
float f1(float x)
```

```
{return x*x-3;}
```

已知x求f1(x)

```
void main(void)
```

```
{ float x1, x2, x0;
```

```
do
```

```
{ cout<<"Input two real number\n";
```

```
cin>>x1>>x2;
```

输入初值

```
}while (f1(x1)*f1(x2)>0);
```

判断x1与x2之间
是否有方程根

```
do
```

求中值

```
{ x0=(x1+x2)/2;
```

```
if ( (f1(x1)*f1(x0)) <0 )
```

```
x2=x0;
```

循环迭代

```
else x1=x0;
```

```
}while (fabs (f1(x0))>=1e-6);
```

```
cout<<x0<<endl;
```

循环结束条件

```
}
```

当求解方程 $f_2(x)=3x^2-5x-3$ 时，同样

```
float f2(float x)
```

```
{return 3*x*x-5*x-3;}
```

```
main( )
```

```
{ float x1, x2, x0;
```

```
do
```

```
{ cout<<"Input two real number\n";
```

```
cin>>x1>>x2;
```

```
}while (f2(x1)*f2(x2)>0);
```

```
do
```

```
{ x0=(x1+x2)/2;
```

```
if ( (f2(x1)*f2(x0)) <0 )
```

```
x2=x0;
```

```
else x1=x0;
```

```
}while (fabs (f2(x0))>=1e-6);
```

```
cout<<x0<<endl;
```

```
}
```

可以看到，虽然算法相同，仅是方程不同，两个程序不能通用。

可以用指向函数的指针变量，设计相同算法的通用函数。

```
float devide(float (*fun)(float ))
```

```
{ float x1, x2, x0;
```

```
do
```

```
{ cout<<"Input
```

```
cin>>x1>>x2;
```

```
}while ( fun(x1) * fun(x2) >0);
```

```
do
```

```
{ x0=(x1+x2)/2;
```

```
if ( fun(x1) * fun(x0) <0 )
```

```
x2=x0;
```

```
else x1=x0;
```

```
}while (fabs ( fun(x0) )>=1e-6);
```

```
return x0;
```

```
}
```

形参用指向函数的
指针变量fun
接受函数名

用fun调用函数，
实际调用的是实参函数

```
float f1(float x)
```

```
{return x*x-3;}
```

```
float f2(float x)
```

```
{return 3*x*x-5*x-3;}
```

```
main( )
```

```
{
```

```
float y1,y2;
```

```
y1=devide(f1);
```

```
y2=devide(f2);
```

```
cout<<y1<<"\t"<<y2;
```

```
}
```

调用函数，
实参为函数名f1

实参：实际的函数名（函数地址）

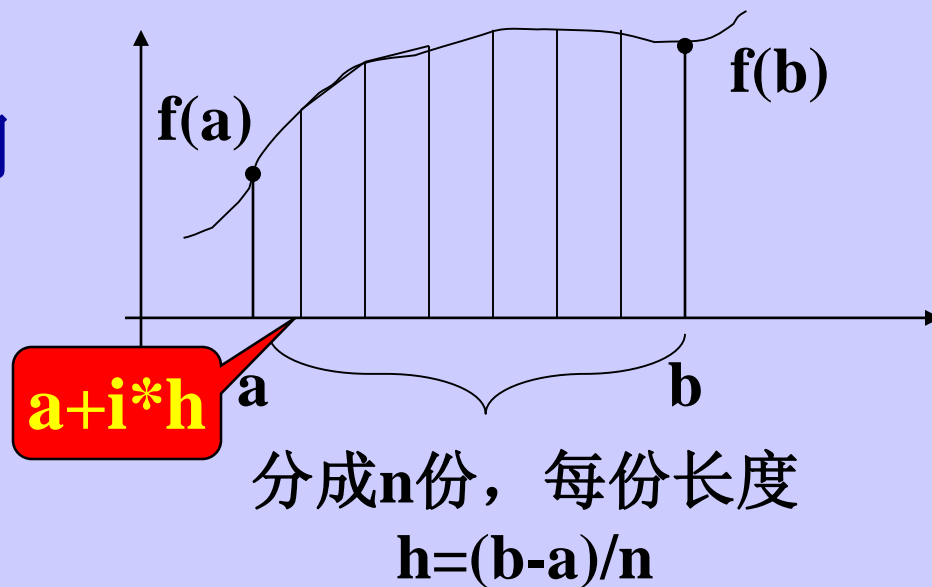
形参：指向函数的指针变量

与实参函数的类型完全一致（返回值、参数）

通用函数：所有的内部函数调用都用函数指针调用

梯形法求定积分的公式

积分结果为曲线与x轴之间部分的面积，也即为每份面积之和。



任意一份：高： h

上底： $f(a+i*h)$ 下底： $f(a+(i+1)*h)$

$$S = \sum [(上底 + 下底) * 高 / 2]$$

$$= \sum [(f(a+i*h) + f(a+(i+1)*h)) * h / 2]$$

其中 $i=0 \sim (n-1)$ $a+n*h=b$

$$S = \sum [(上底 + 下底) * 高 / 2]$$

$$= \sum [(f(a+i*h) + f(a+(i+1)*h)) * h / 2]$$

其中 $i=0 \sim (n-1)$ $a+n*h=b$

将上式展开，除 $f(a)$ ， $f(b)$ 外，每一底边都参与运算两次，所以有：

$$S = [(f(a) + f(b)) / 2 + \sum f(a+i*h)] * h \quad (i=1 \sim (n-1))$$

$$y = (f(a) + f(b)) / 2;$$

$$h = (b - a) / n;$$

$$\text{for}(i=1; i < n; i++)$$

$$y = y + f(a + i * h);$$

$$y = y * h;$$

初值

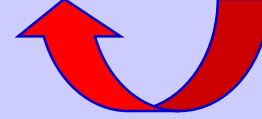
计算累加和

可利用这一公式，计算所有函数的定积分，也即做成通用公式，具体的函数由指向函数的指针变量来代替，在主调函数中为这个指针变量赋值。

```
float f1(float x)
```

```
{ return x*x*x+2*x*x+9; }
```

pf=f1



```
float jifen(float (*pf)(float ), float a, float b)
```

```
{ y=(pf(a)+pf(b))/2;
```

```
h=(b-a)/2;
```

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

```
y=y+pf(a+i*h);
```

```
y=y*h;
```

```
return y;
```

```
void main(void)
```

```
{ float y;
```

```
y=jifen(f1, 1.0, 3.0);
```

```
cout<<y<<endl;
```

```
}
```

```
y2=jifen(f2, 2.0, 5.0);
```

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

```
return (str1); 束
```

str2向p赋值

```
}
```

```
void main(void)
```

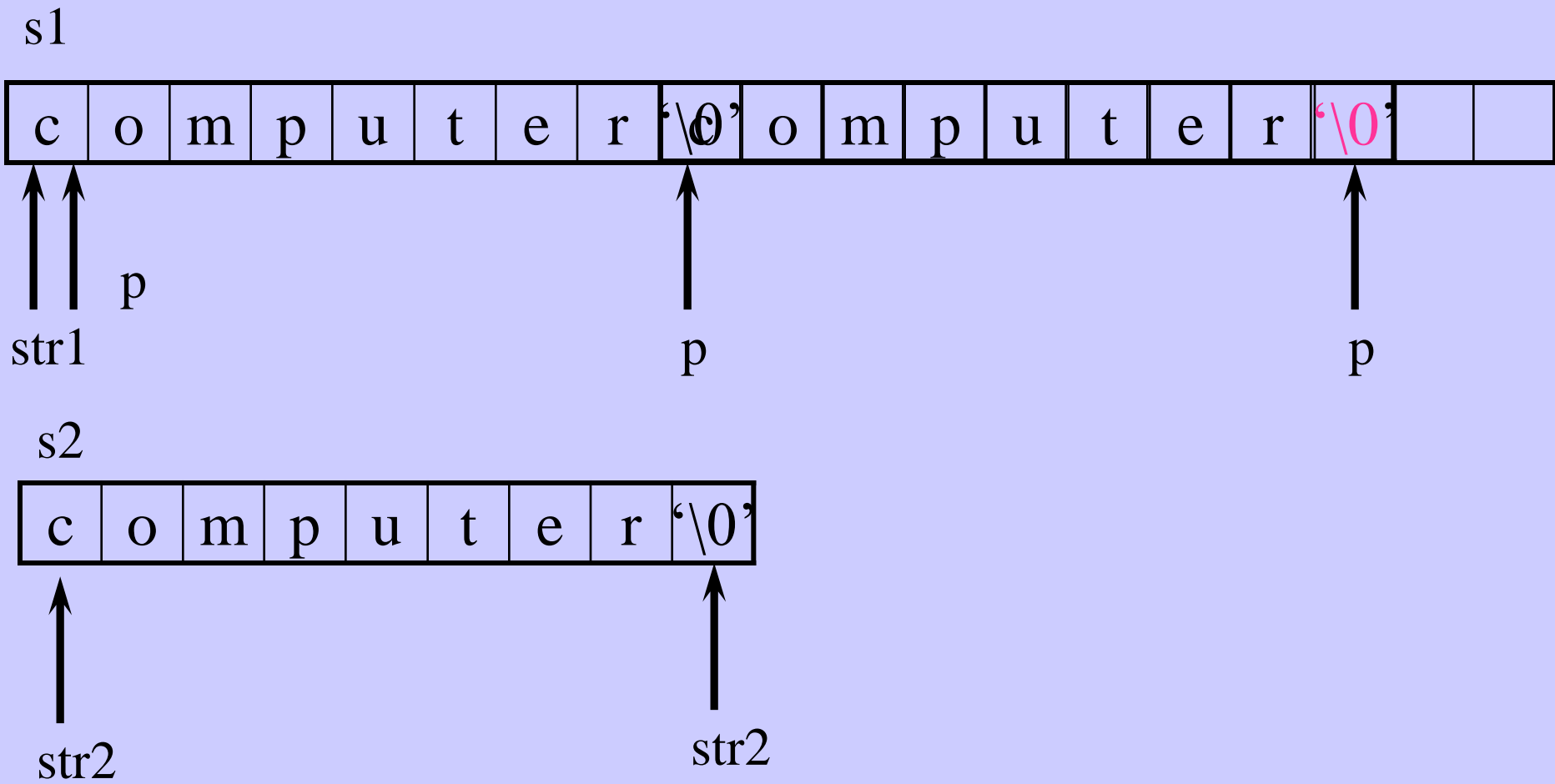
```
{char s1[80]="computer", s2[ ]="language", *pt;
```

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

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

computerlanguage

```
}
```



已知函数 `int *f(int *)`，有以下说明语句：

`int *p, *s, a;`

函数正确的调用形式是：

A) `a=*f(s)` B) `*p=f(s)` C) `s=f(*p)` D) `p=f(&a)`

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;
```

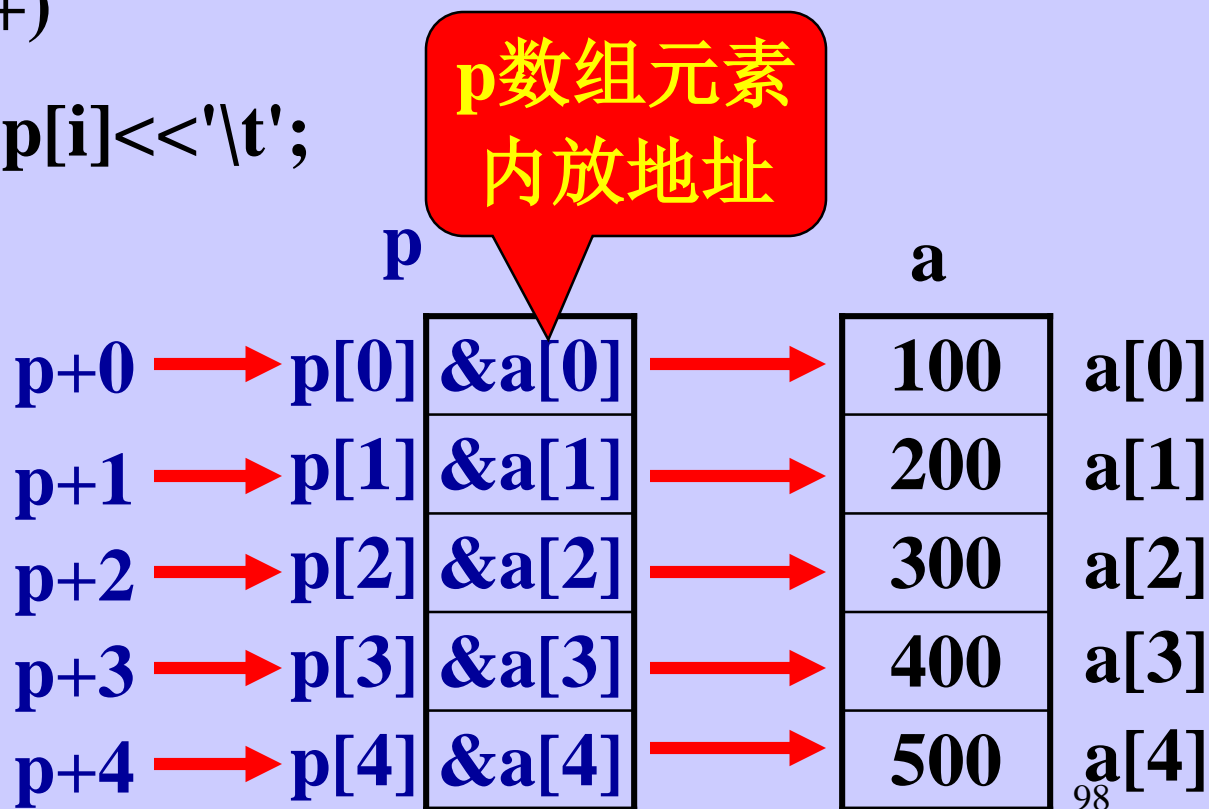
```
    for(i=0;i<5;i++)
```

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

```
    cout<<endl;
```

```
}
```

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



```
void main(void)
```

```
{ int a[12]={1,2,3,...11,12};
```

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

```
    for(i=0; i<4;i++)
```

```
        p[i]=&a[i*3];
```

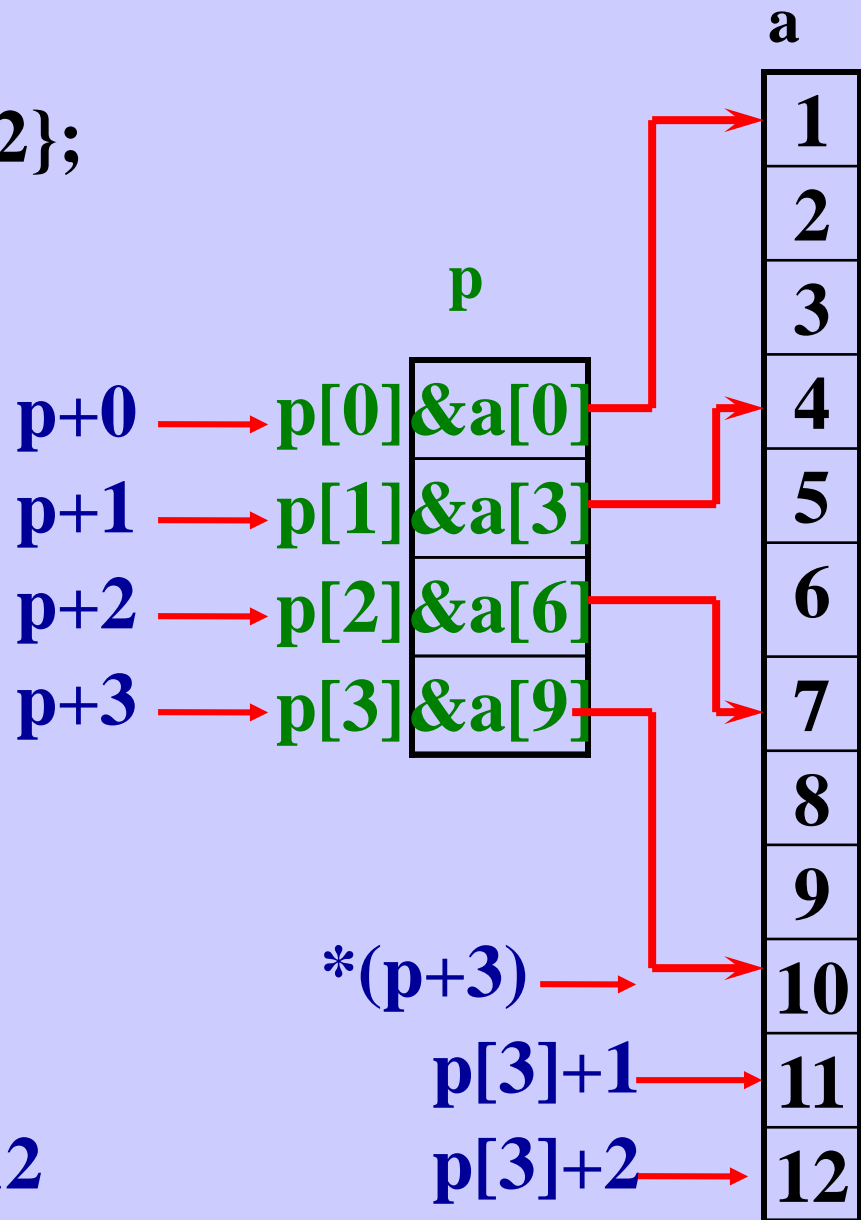
```
    cout<<p[3][2]<<endl;
```

```
}
```

```
p[3][2]=*(* (p+3)+2)
```

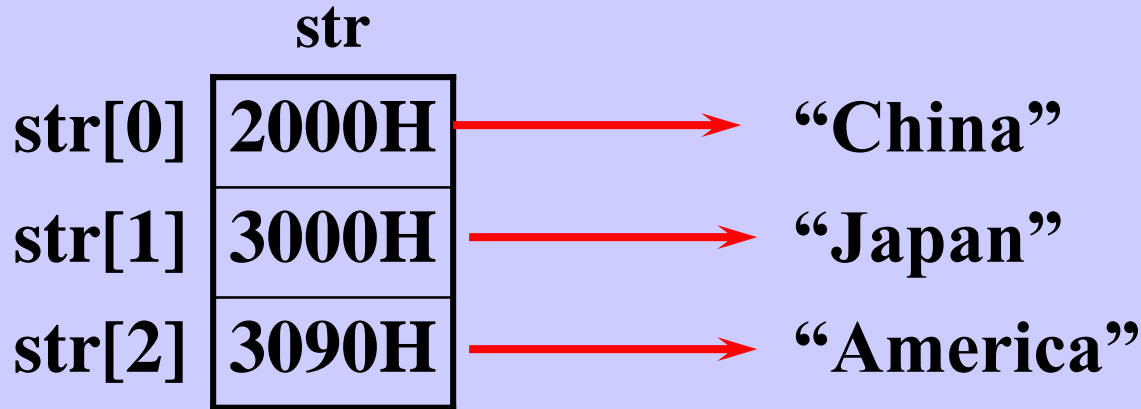
```
=*(p[3]+2)
```

12



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**

将若干字符串按字母顺序（由小到大）输出。

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

```
void sort(char *alpha[ ], 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(alpha[k], alpha[j])>0)
```

```
        k=j;
```

```
  if (k!=i)
```

```
  { temp=alpha[i];
```

```
    alpha[i]=alpha[k];
```

```
    alpha[k]=temp;
```

```
  }
```

```
}
```

```
}
```

```
void print(char *alpha[ ], int n)
```

```
{ int i;
```

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

```
    cout<<alpha[i]<<endl;
```

```
}
```

选择法排序

交换地址

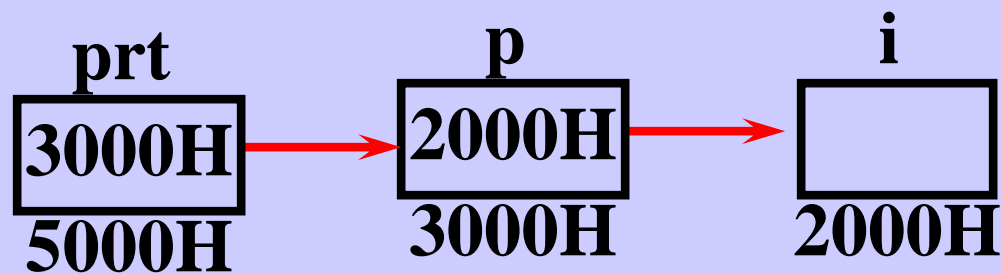
alpha[0]
alpha[1]
alpha[2]
alpha[3]
alpha[4]

“Follow me”
“Basic”
“Great Wall”
“FORTRAN”
“Computer design”

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。

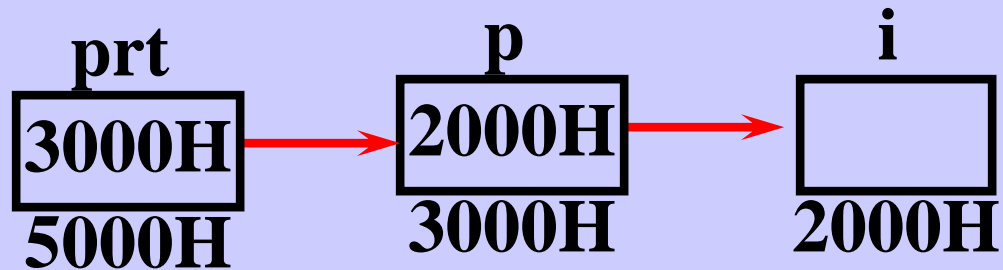
p=&i; prt=&
p;

*p=i;

**prt=i;

prt=&i; prt=p;

非法,基类
型不符




```
void main(void)
```

```
{ 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;
```

```
    p=&num[0]
```

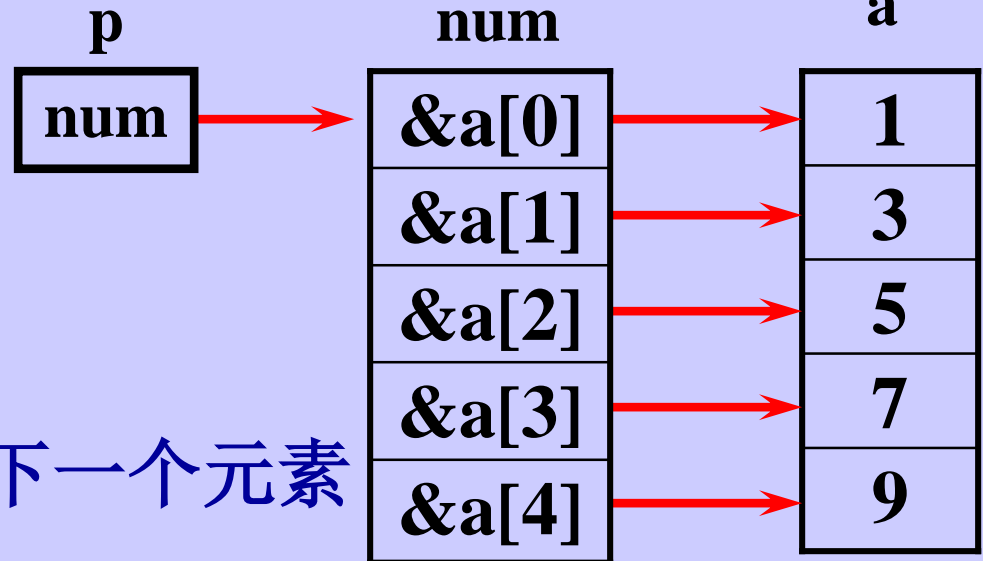
```
    for(i=0;i<5;i++)
```

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

```
        p++;
```

```
    }
```

```
} p=p+1;指向num数组下一个元素
```



1 3 5 7 9

```
void main(void)
```

```
{char *alpha[ ]={“Follow me”, “Basic”, “Great Wall”,  
“FORTRAN”, “Computer design”};
```

```
char **p;
```

```
int i;
```

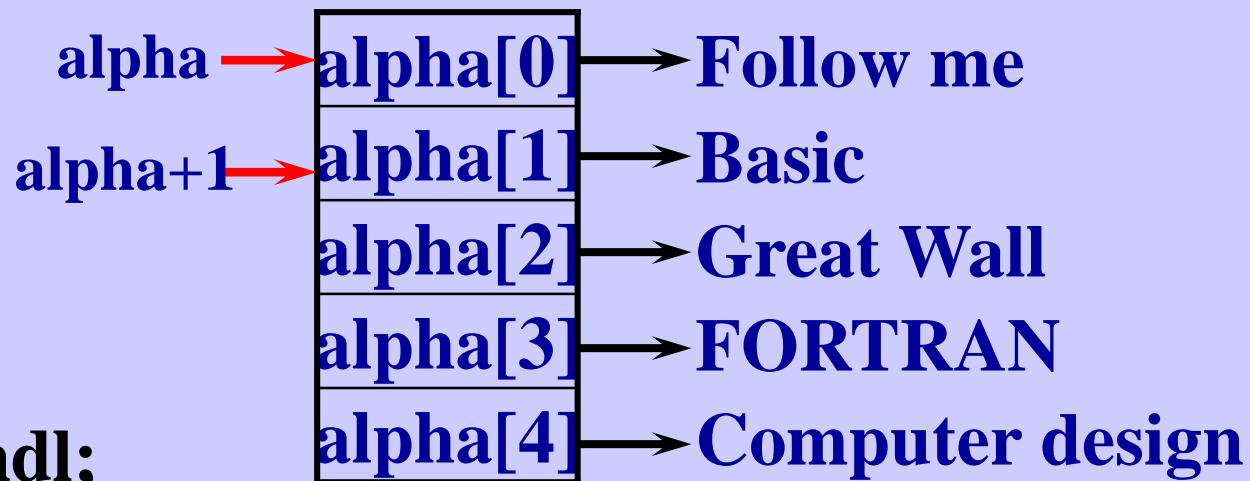
```
for(i=0; i<5; i++)
```

```
{ p=alpha+i;
```

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

```
}        i=0     p=alpha
```

```
}        i=1     p=alpha+1
```



```
void main(void)
```

```
{ char *n[4]={“China”, “Japan”, “England”, “Germany”};
```

```
char **p;
```

```
int i;
```

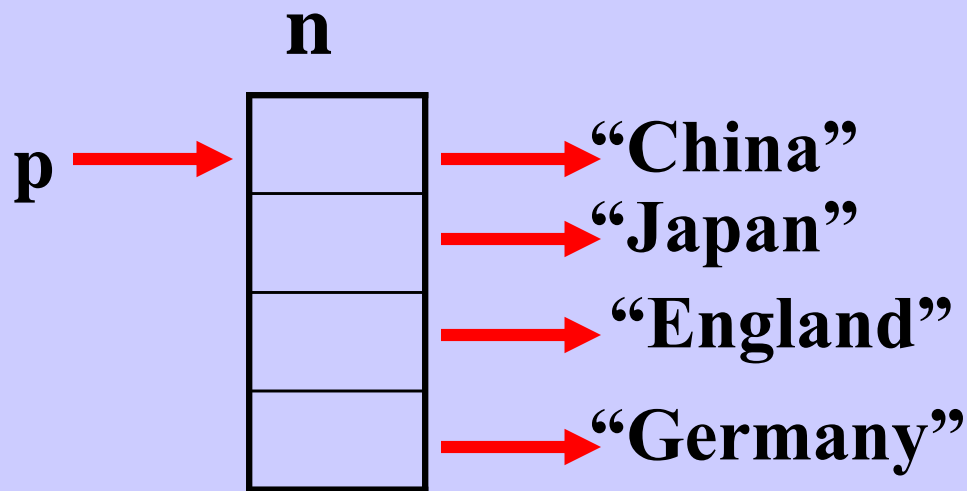
```
p=n;
```

```
for(i=0; i<4;i++,p++)
```

```
cout<<(char) (*(*p+2)+1)<<endl;
```

```
}
```

$*(*p+2)+1 = *(p[0]+2)+1 = p[0][2]+1$



输出: j

q

h

s

以下程序的输出结果是：

```
char *alpha[6]={“ABCD”,“EFGH”,“IJKL”,“MNOP”,  
                “QRST”,“UVWX”};
```

```
char **p;
```

```
main( )
```

```
{ int i;
```

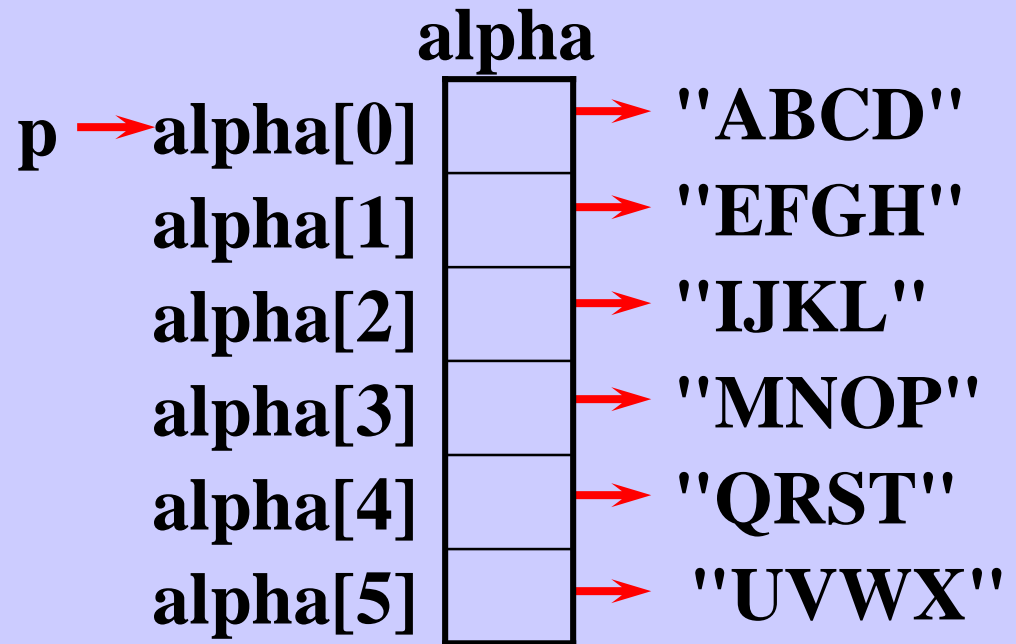
```
  p=alpha;
```

```
  for(i=0;i<4;i++)
```

```
    cout<<*(p[i]);
```

```
  cout<<endl;
```

```
}
```



$*(p[i]) = *(*(p+i)) = *(*(p+i)+0)$

输出： AEIM

以下程序的输出结果是：

```
char *alpha[6]={“ABCD”,“EFGH”,“IJKL”,“MNOP”,  
                “QRST”,“UVWX”};
```

```
char **p;
```

```
main( )
```

```
{ int i;
```

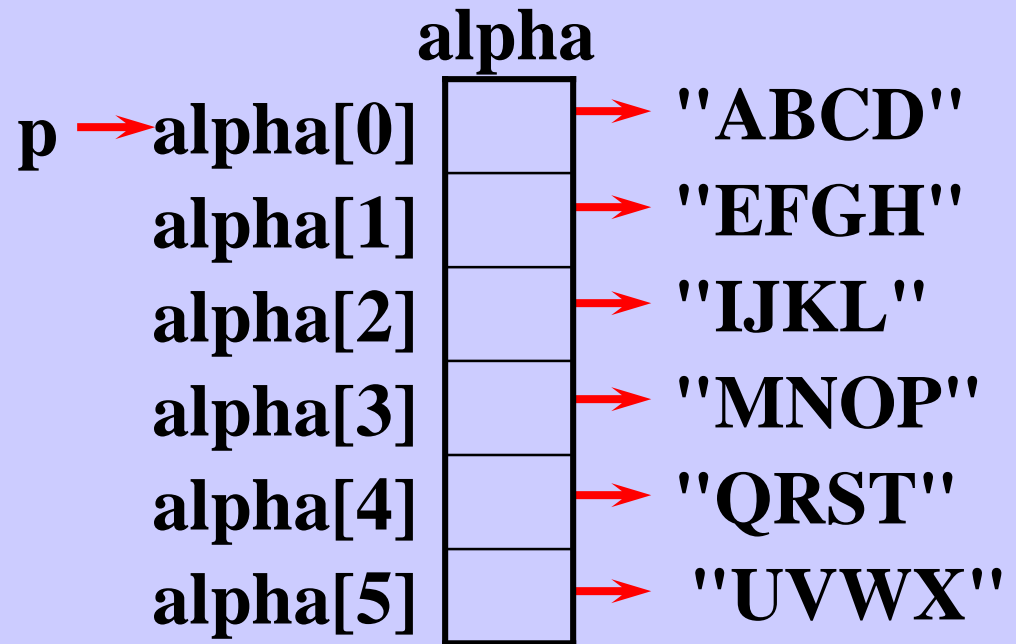
```
  p=alpha;
```

```
  for(i=0;i<4;i++)
```

```
    cout<<(*p)[i];
```

```
    cout<<endl;
```

```
}
```



$(*p)[i] = *(p+0)[i] = p[0][i]$

输出： ABCD

若有以下定义，则下列哪种表示与之等价。

char s[3][5]={“aaaa”,“bbbb”,“cccc”};

A) char **s1= {“aaaa”,“bbbb”,“cccc”};

B) char *s2[3]= {“aaaa”,“bbbb”,“cccc”};

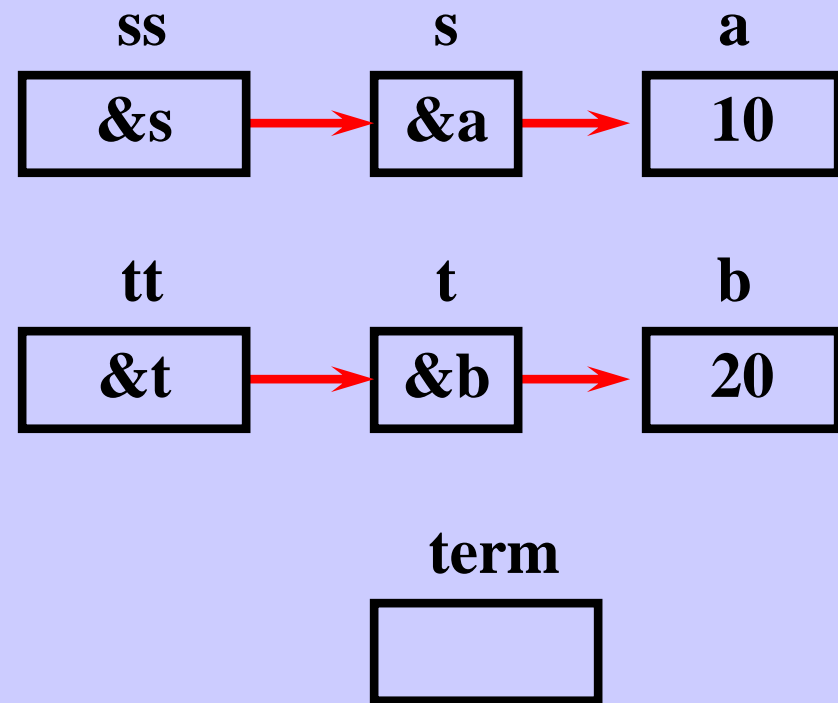
C) char s3[][3]= {“aaaa”,“bbbb”,“cccc”};

D) char s4[][4]= {“aaaa”,“bbbb”,“cccc”};

以下程序调用函数swap_p将指针s和t所指单元(a和b)中的内容交换，请填空。

```
void swap_p (int **ss, int **tt)
{   int term;
    term= **ss;
    **ss= **tt;
    **tt=term;
}

main( )
{   int a=10, b=20, *s,*t;
    s=&a; t=&b;
    swap_p(&s,&t);
    printf(“%d  %d”,a,b);
}
```



假设有说明：

```
char *argv[ ]={"hello", "nanjing", "jiangsu"};
```

```
char **pargv=argv;
```

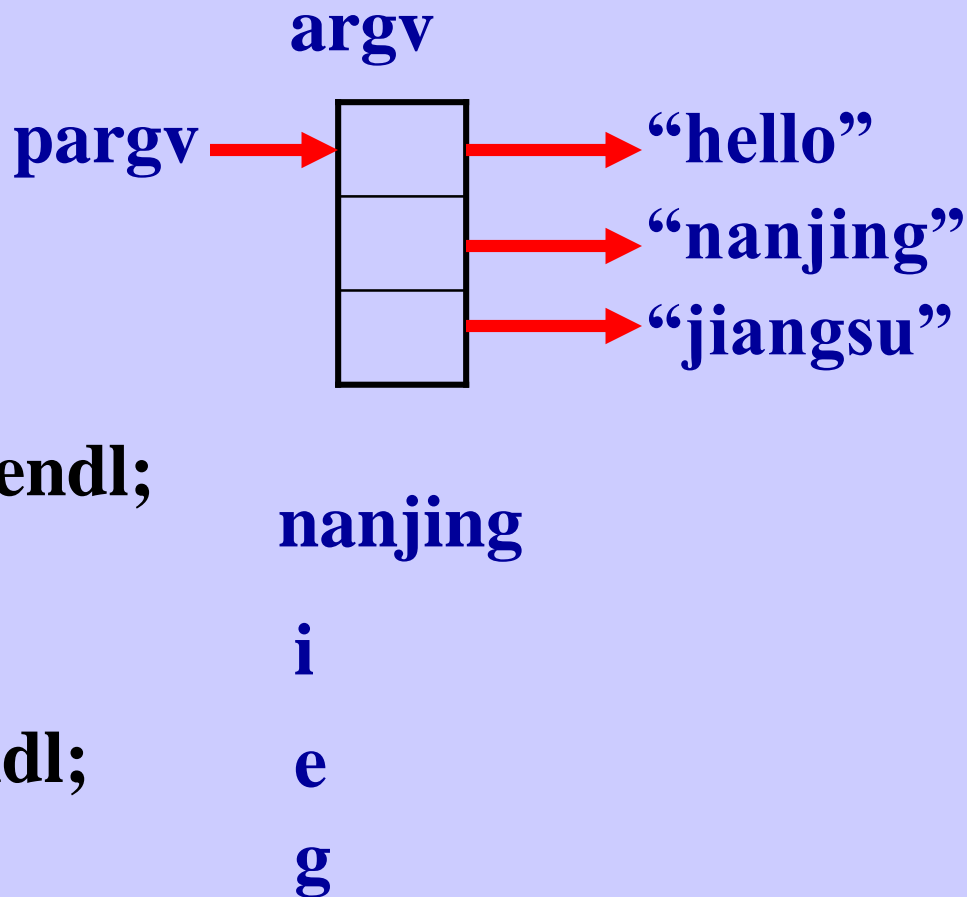
以下语句的输出结果如何？

```
cout<<*(pargv+1)<<endl;
```

```
cout<<(char)(**pargv+1)<<endl;
```

```
cout<<*(*pargv+1)<<endl;
```

```
cout<<*(*pargv+2)+4)<<endl;
```




```

void main()
{
    char *s[]={“1995”,“1996”,“1997”,“1998”};

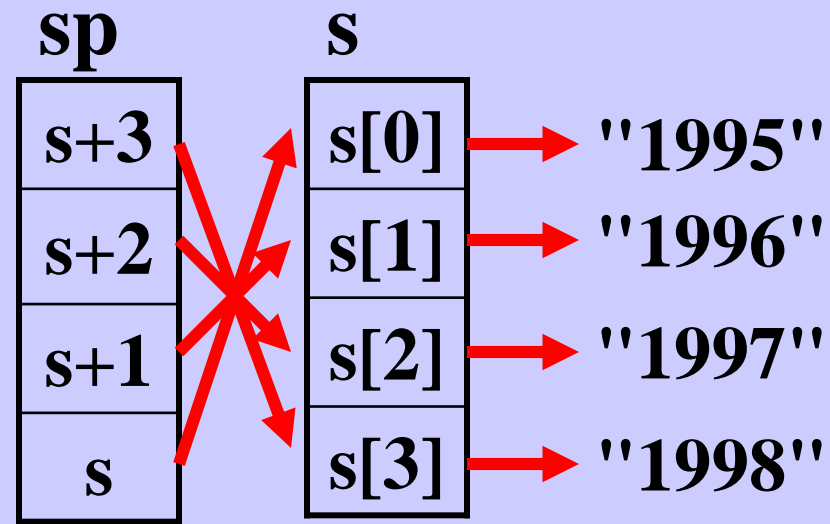
    char **sp[ ]={s+3,s+2,s+1,s};

    char ss[5];

    ss[0]**sp[0];
    ss[1]=*(*sp[1]+1);
    ss[2]=*(*sp[2]+2);
    ss[3]=*sp[3][3]+6;
    ss[4]=‘\0’;

    cout<<ss<<endl;
}

```



SS

1	9	9	7	0
---	---	---	---	---

总结:

通过行指针引用二维数组元素

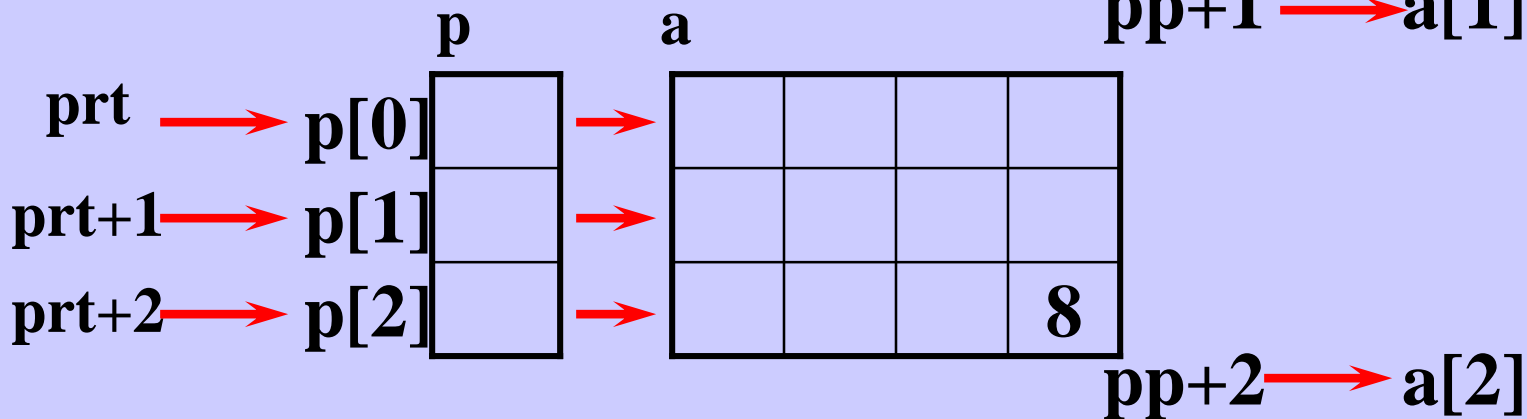
```
int a[3][4], *p[3], (*pp)[4], **prt;
```

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

```
    p[i]=a[i];
```

```
pp=a;
```

```
prt=p;
```



`a[2][3]` `*(*(pp+2)+3)`

`a[2][3]` `*(*(p+2)+3)`

`a[2][3]` `*(*(a+2)+3)`

`a[2][3]` `*(*(prt+2)+3)`

`a[2][3]` `pp[2][3]` `p[2][3]` `prt[2][3]`

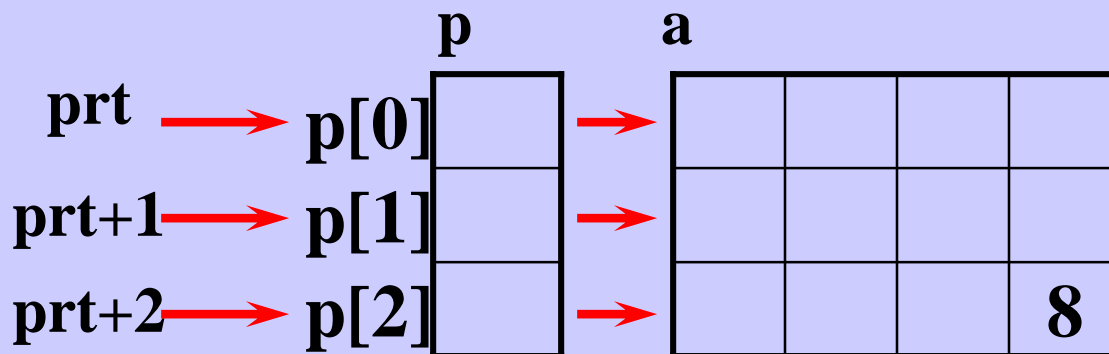
```
int  a[3][4], *p[3], (*pp)[4], **prt;
```

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

```
    p[i]=a[i];
```

```
pp=a;
```

```
prt=p;
```



a: 二维数组名, **常量**

p: 指针数组名, **常量**

pp: 指向具有四个元素的一维数组的指针**变量**

prt : 指向指针的**指针变量**

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>

三个形参

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;
```

```
}
```

```
}
```

B9_1 Beijing China<CR>

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

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

Beijing

```
    cout<< *(++argv)<<endl;
```

China

```
}
```

```
argc=3
```

argv

argv[0]	→ “B9_1.EXE”
argv[1]	→ “Beijing”
argv[2]	→ “China”

```
argc=3 / 2    Beijing
```

```
argc=2 / 1    China
```

```
argc=1
```

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.h"    #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内存空间？

```
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

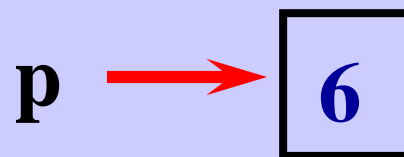
`new` 相当于一个函数，在内存开辟完空间后，返回这个空间的首地址，这时，**这个地址必须用一个指针保存下来，才不会丢失。**

`int *p;`

`p=new int;`



`*p=6;`



`new`开辟的空间

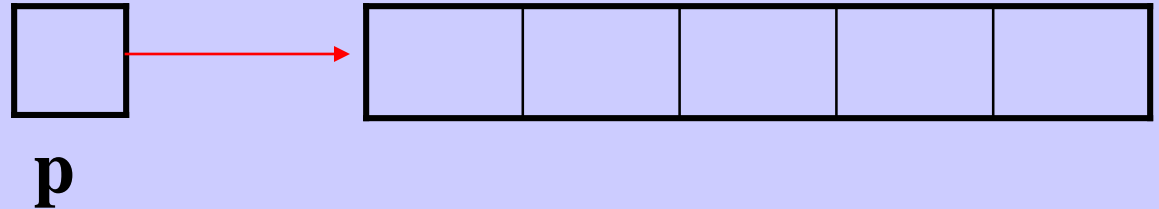
可以用**`*p`**对这个空间进行运算。

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];
```

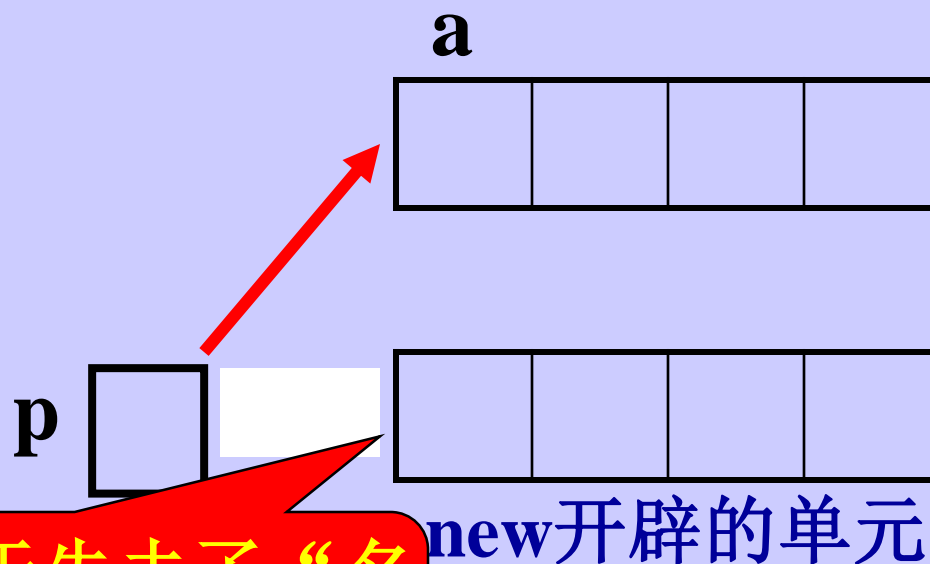
可以用`p[i]`的形式来引用新开辟的内存单元。

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

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）。

以下程序求两个数的大者，请填空。

```
void main(void )
```

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

```
    p1= new int ;
```

```
    p2= new int ;
```

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

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

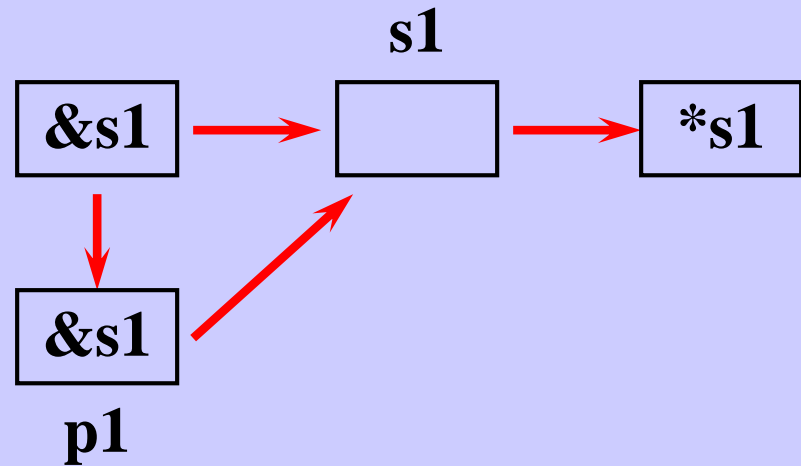
```
    delete p2;
```

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

```

main( )
{  int *s1, *s2;
    sub1(&s1,&s2);  sub2(&s1,&s2);
    cout<<*s1<<'\t'<<*s2<<endl;
    sub3(s1, s2);  sub4(s1,s2);
    cout<<*s1<<'\t'<<*s2<<endl;
}

```



```

sub1( int  **p1, int  **p2)
{ *p1=new int ; *p2=new int ; }

sub2(int  **p1, int  **p2)
{ **p1=10; **p2=20; **p1=**p2; }

sub3(int  *p1, int  *p2)
{ p1=new int ; p2=new int ; }

sub4( int  *p1, int  *p2)
{ *p1=1; *p2=2; *p2=*p1; }

```

20	20
1	1

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;
```

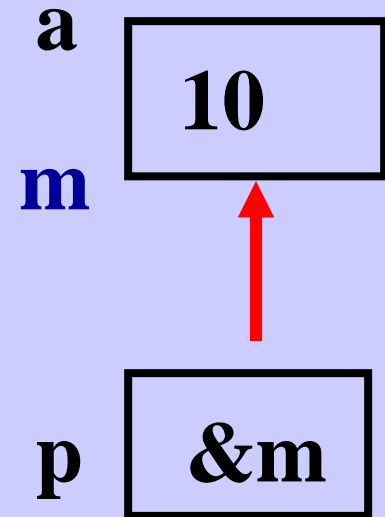
&是变量的引用

```
int &m=a;
```

&是变量的地址

```
p=&m;
```

```
*p=10;
```



5、可以用动态分配的内存空间来初始化一个引用变量。

```
float &reff = * new float ; //用new开辟一个空间，取一个别名reff
```

```
reff= 200;    //给空间赋值
```

```
cout << reff ; //输出200
```

```
delete &reff; //收回这个空间
```

这个空间只有别名，但程序可以引用到。

```
float *p, a;
```

```
p=new float;
```

```
float a=* new float;
```

错误！

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。

```
int m=10;
```

```
int &y=10;          int &z;
```

```
float &t=&m;   int &x=m;
```

以下的声明是非法的

1、企图建立数组的引用 **int & a[9];**

2、企图建立指向引用的指针 **int & *p;**

3、企图建立引用的引用 **int & &px;**

对常量（用**const**声明）的引用

```
void main(void)  
{  
    const int &r=8; //说明r为常量，不可赋值  
    cout<<"r="<<r<<endl;  
    // r+=15;           //r为常量，不可作赋值运算  
    cout<<"r="<<r<<endl;  
}
```

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;
```

```
}
```

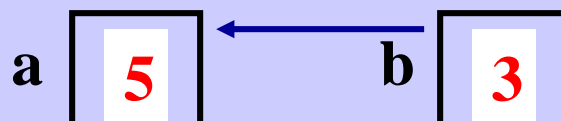
```
void main(void)
```

```
{ 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

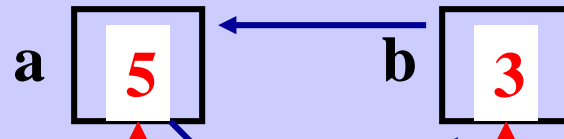
```
void change(int &x, int &y)
{ int t;
  t=x; x=y; y=z;
}

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;
}
```



```
void dd(int &x, int &y, int z)
```

```
{  x=x+z;      x=8      x=13
```

```
   y=y-x;      y=-4     y=-17
```

```
   z=10;       z=10     z=10
```

```
   cout<<"(2)"<<x<<"\t"<<y<<"\t"<<z<<endl;
```

```
}
```

```
void main(void)      (2) 8    -4    10
```

```
{  int  a=3,b=4,c=5;  (2) 13   -17   10
```

```
   for(int i=0;i<2;i++) (1) 13   -17   5
```

```
       dd(a,b,c);
```

```
   cout<<"(1)"<<a<<"\t"<<b<<"\t"<<c<<endl;
```

```
}
```

```
void f1( int *px)  {  *px+=10;}
```

```
void f2(int &xx)  {  xx+=10;}
```

```
void main(void)
```

```
{    int x=0;
```

x=0

```
    cout<<"x="<<x<<endl;
```

x=10

```
    f1(&x);
```

x=20

```
    cout<<"x="<<x<<endl;
```

```
    f2(x);
```

```
    cout<<"x="<<x<<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)

一个函数返回引用类型，必须返回某个类型的变量。

语句： `getdata()=8;`

就相当于 `int &temp=8;`

`temp=8 ;`

注意：由于函数调用返回的引用类型是在函数运行结束后产生的，所以函数不能返回自动变量和形参。

返回的变量的引用，这个变量必须是全局变量或静态局部变量，即存储在静态区中的变量。

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。

```
int &f(int &x)
{  static int t=2;
   t=x++;
   return t;
}

void main(void)
{  int a=3;
   cout<<f(a)<<endl;
   f(a)=20;
   a=a+5;
   cout<<f(a)<<endl;
   a=f(a);
   cout<<f(a)<<endl;
}
```

输出 3 a=4 t=3
t=20 a=5
a=10
输出 10 a=11
a=11
输出 11 a=12

const类型变量

当用const限制说明标识符时，表示所说明的数据类型为常量类型。可分为const型常量和const型指针。

可用const限制定义标识符量，如：

```
const int MaxLine = 1000;
```

```
const float Pi = 3.1415926
```

用const定义的标识符常量时，一定要对其初始化。在说明时进行初始化是对这种常量置值的唯一方法，不能用赋值运算符对这种常量进行赋值。如：

```
MaxLine = 35;
```

const 型指针

1)禁写指针

声明语句格式为： 数据类型 * const 指针变量名

如： int r=6;

```
int * const pr=&r;
```

则指针pr被禁写，即pr将始终指向一个地址，成为一个指针常量。它将不能再作为左值而放在赋值号的左边。（举例说明）

同样，禁写指针一定要在定义的时候赋初值。

虽然指针被禁写，但其间接引用并没有被禁写。即可以通过pr对r赋值。*pr=8;

```
void main(void)
```

```
{
```

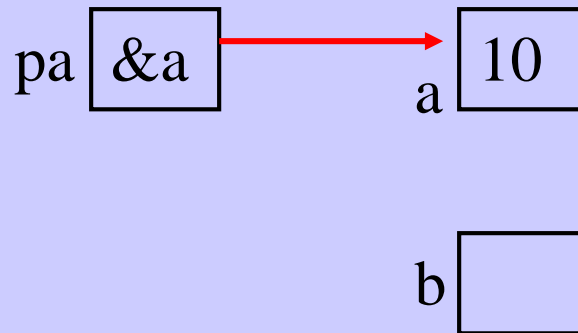
```
    int a,b;
```

```
    int *const pa=&a; //一定要赋初值，pa是常量，不能在程序中  
                      //被改变
```

```
    *pa=10;           //可以间接引用
```

```
    pa=&b;             //非法，pa为常量
```

```
}
```



2) 禁写间接引用

声明语句格式如下：

```
const 数据类型 *指针变量名;
```

所声明的指针指向一禁写的实体，即间接引用不能被改写。如：

```
const int *p;
```

所以程序中不能出现诸如 `*p=` 的语句，但指针`p`并未被禁写，因而可对指针`p`进行改写。


```
void main(void)
```

```
{
```

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

```
    const int *pa=&b;    //可以不赋初值
```

```
    pa=&a;                //指针变量可以重新赋值
```

```
    cout<<*pa<<endl; //输出3
```

```
    *pa=10;              //非法，指针指向的内容不能赋值
```

```
    a=100;               //变量可以重新赋值
```

```
    cout<<*pa<<endl; //输出100
```

```
}
```

即不可以通过指针对变量重新赋值

3)禁写指针又禁写间接引用

将上面两种情况结合起来，声明语句为下面的格式

`const 数据类型 *const 指针变量名`

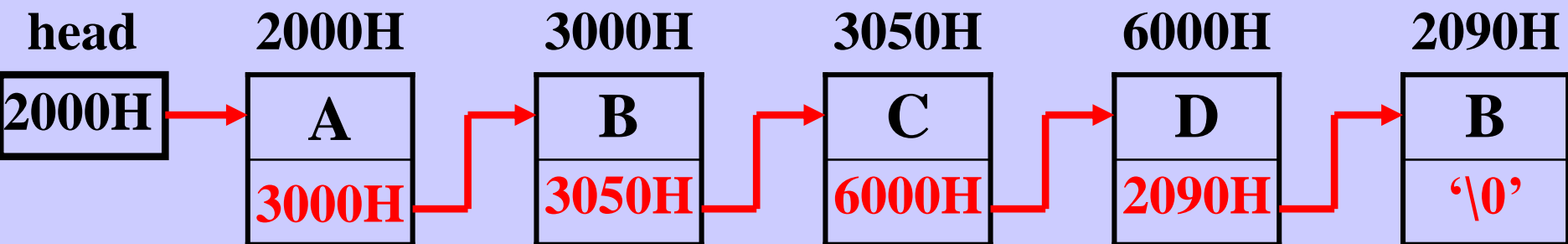
如：`const int *const px=&x`

说明：`px`是一个指针常量，它指向一禁写的实体，并且指针本身也被禁写，诸如：`px=` `*px=` 此类的语句都是非法的。

在定义时必须赋初值。

用指针处理链表

一、链表概述



链表是由一个个结点组成，每一个结点是一个结构体类型的变量，各个结点的类型相同，但其地址不一定连续。具体结点的个数根据需要动态开辟。

每个结点由两部分组成，第一部分放若干数据，第二部分是指针变量，放下一结点的地址。链表头是一指针变量，放第一个结点的地址，若结点的第二部分的值为NULL，表示此链表结束。

二、如何处理链表

1、建立链表

链表结点的结构:

```
struct student
{
    int num;

    float score;

    struct student *next;
};
```

指向同一结构体类型的指针变量

```
#define STU struct student

STU

{
    int num;

    float score;

    STU *next;
};
```

指向同一结构体类型的指针变量

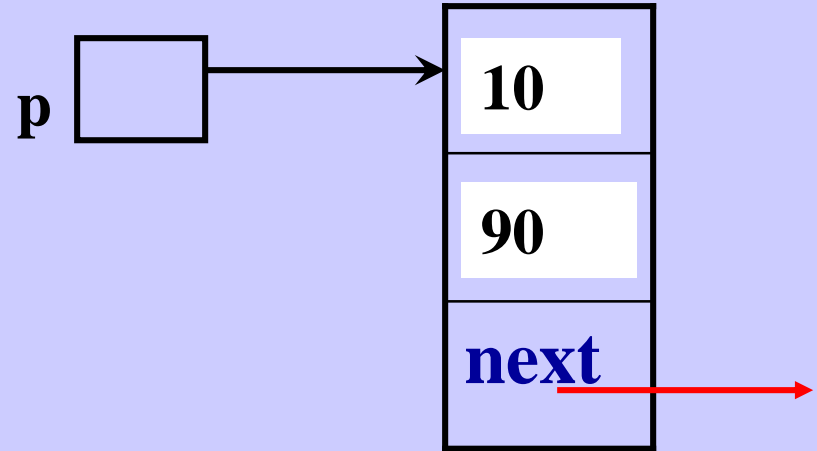
```
struct student
```

```
{ int num;
```

```
    float score;
```

```
    struct student *next;
```

```
};
```



```
struct student *p; //定义了结构体类型的指针
```

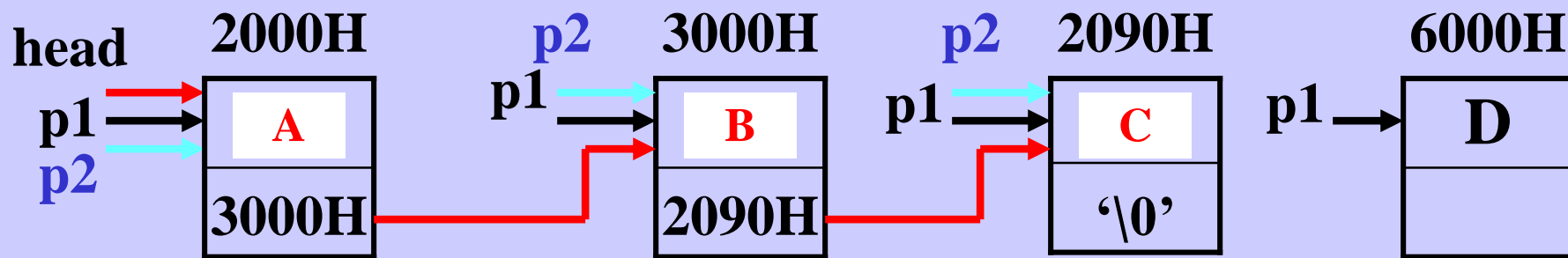
```
p=new student; //用new开辟一结构体空间，将地址赋给p
```

```
p->num=10; //为新开辟的结构体空间中的num成员赋值
```

```
p->score=90;
```

用指针引用结构体内的成员

(*p).num



1、首先定义两个结构体类型的指针 `STU *p1, *p2;`

2、用new在内存中开辟一个结构体变量的空间，将地址赋给p1。

```
p1=new student; /* STU struct student */
```

3、将数据赋给刚开辟的变量空间。

```
cin>>p1->num>>p1->score;
```

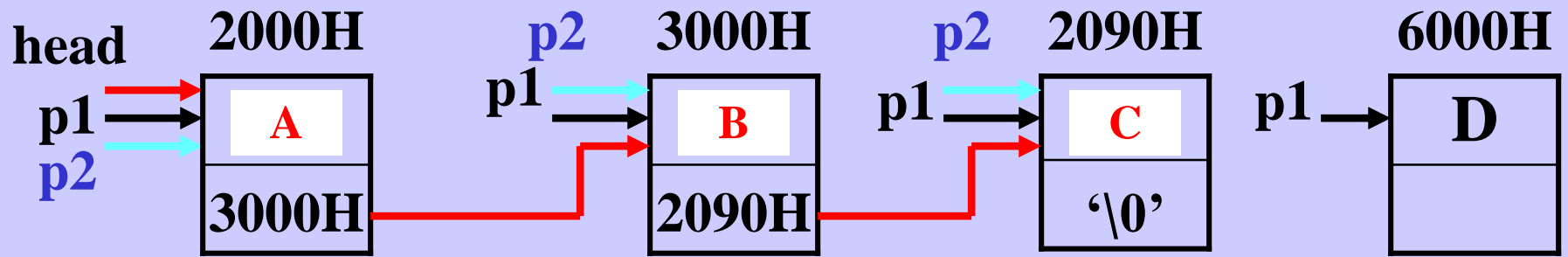
4、若输入的数据有效，将首地址作为链表头，`head=p1`；令`p2=p1`，p1继续用new开辟新的内存空间。

```
p1=new student; /* STU struct student */
```

5、将下一个数据赋给新开辟的变量空间。

```
cin>>p1->num>>p1->score;
```

6、若输入的数据有效，将p2与p1连接起来，`p2->next=p1` 再令`p2=p1`，p1继续用new开辟新的内存空间。做5。若输入的数据无效，p2就是链表的尾，则`p2->next=NULL`。



```
STU  *p1, *p2, *head;
head=NULL;
p1=p2=new student;
cin>>p1->num>>p1->score;
if (p1->num!=0)
    head=p1;
```

第一结点

```
p1=new student;
cin>>p1->num>>p1->score;
if (p1->num!=0)
{ p2->next=p1; p2=p1;}
```

第二结点

```
p1=new student;
cin>>p1->num>>p1->score;
if (p1->num!=0)
{ p2->next=p1; p2=p1;
```

第三结点

```
p1=new student;
cin>>p1->num>>p1->score;
if (p1->num==0)
    p2->next=NULL;
return (head);
```

返回链表头

最后结点


```
STU *creat( )
```

```
{ STU *head, *p1,*p2;
```

```
  n=0;
```

n为全局变量，表示结点数

```
  head=NULL;
```

```
  p1=p2=new student;
```

```
  cin>>p1->num>>p1->score;
```

```
  while (p1->num!=0)
```

```
  {   n=n+1;
```

```
      if (n==1) head=p1;
```

```
      else      p2->next=p1;
```

```
      p2=p1;
```

```
      p1=new student;
```

开辟新结点

```
      cin>>p1->num>>p1->score;
```

```
  }
```

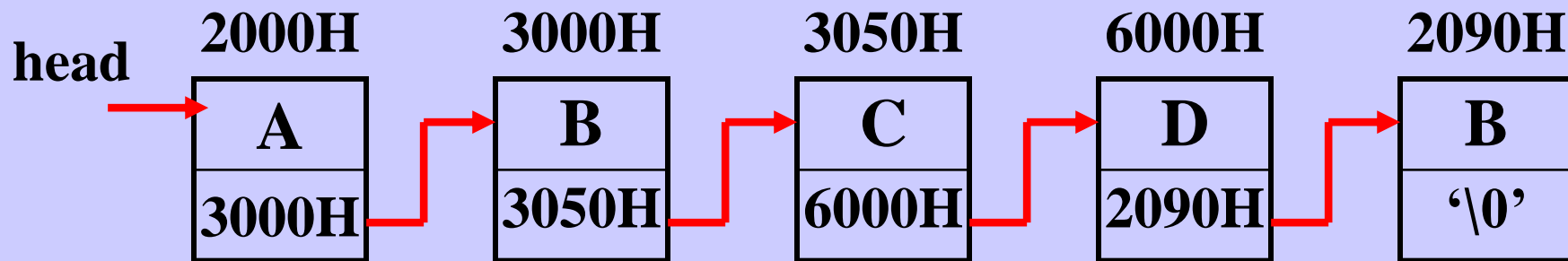
向新结点输入数据

```
  p2->next=NULL;
```

```
  return(head);
```

不满足输入条件，结束

```
}
```



2、输出链表

```
void print(STU * head)
```

```
{ STU *p;
```

```
  p=head;
```

```
  while(p!=NULL)
```

```
  { cout<<p->num<<'\t'<<p->score<<'\n';
```

```
    p=p->next;
```

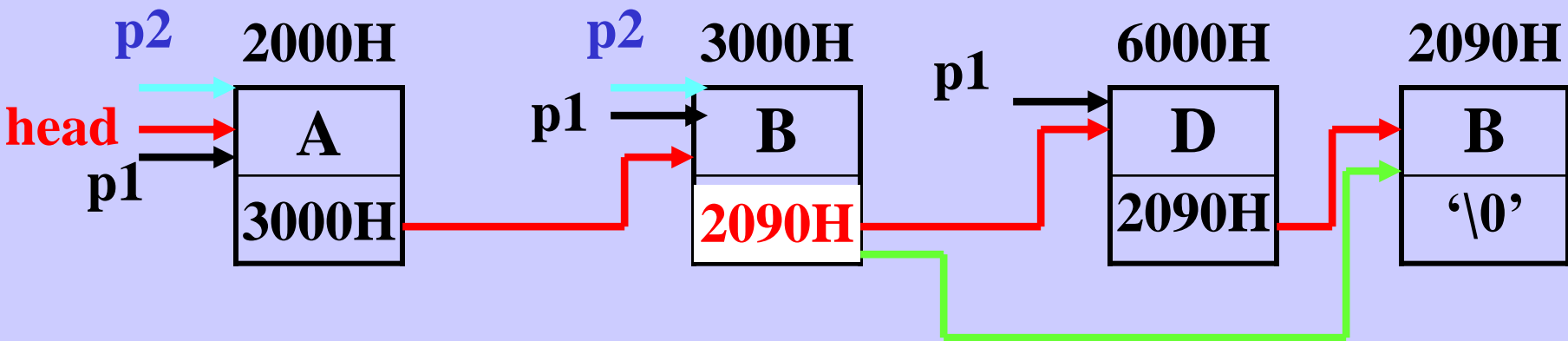
```
  }
```

```
}
```

输出数据

p指向下一结点

3、删除链表



- 1、首先定义两个结构体类型的指针 `STU *p1, *p2;`
- 2、将链表的表头赋给p1, `p1=head;`
- 3、判断p1所指向的结点是否是要删除的结点 `p1->num == a1`。
- 4、若`p1->num != a1`, `p2=p1`; p1指向下一个结点`p1=p1->next`,继续判断下一个结点是否是要删除的结点。继续做3。
- 5、若`p1->num == a1`, 则p1当前指向的结点就是要删除的结点, 将p2的指针成员指向p1所指的下一个结点。

这样就删除了一个结点。

`p2->next=p1->next;`

特殊情况：

- 1、若链表为空链表，返回空指针。**
- 2、删除的结点为头结点时，head指向下一个结点**
- 3、链表内没有要删除的结点，返回提示信息。**

```

struct student *del(struct student * head, int num)
{
    struct student *p1,*p2;
    if (head==NULL)
    {
        cout<<"list null\n"; return NULL;
    }
    p1=head;
    while (num!=p1->num&& p1->next!=NULL)
    {
        p2=p1; p1=p1->next;
    }
    if(num==p1->num)
    {
        if (num==head->num) head=p1->next;
        else p2->next=p1->next;
        n=n-1;
    }
    else cout<<"Not found\n";
    return head;
}

```

链表为空

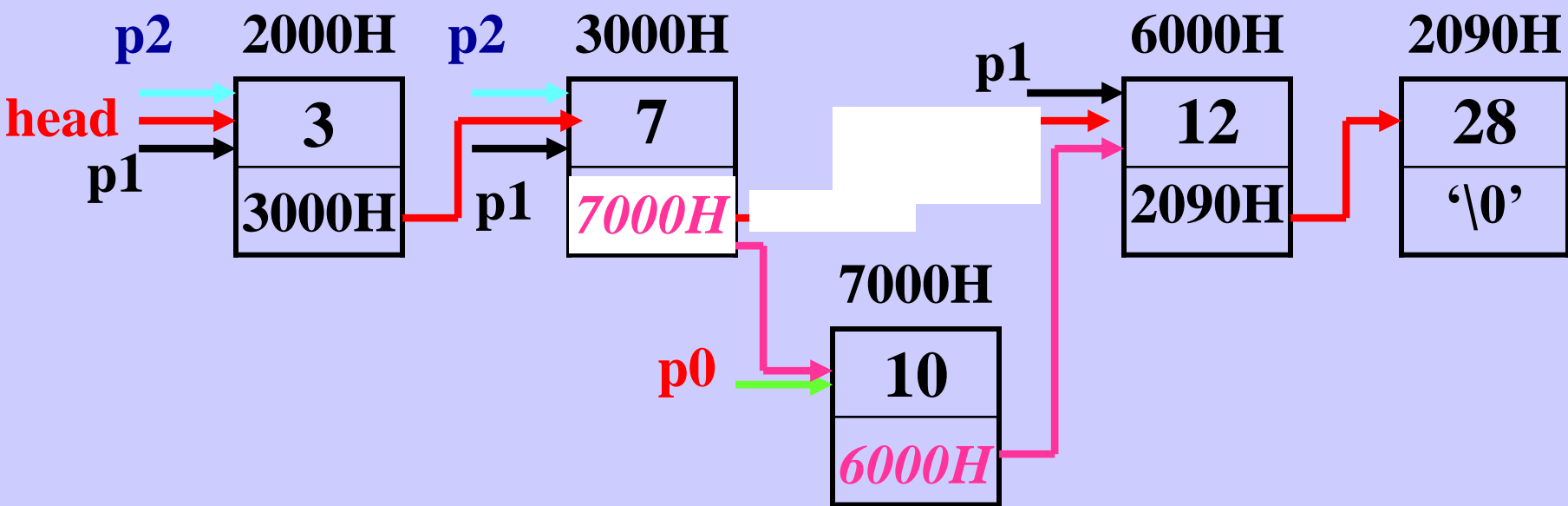
未找到结点，循环

找到结点

结点为第一个

循环结束，没有要找的结点

4、插入结点：要插入结点的链表是排序的链表。插入10。



1、定义三个结构体指针变量 `STU *p1,*p2,*p0`; `p0`指向要插入的结点。`p1=head`;

2、比较`p1->num`与`p0->num`, 若`p1->num < p0->num`, `p2=p1`;
`p1=p1->next`; 继续比较。

3、若`p1->num >= p0->num`, `p0`应插在`p1`与`p2`之间, 则`p2->next=p0`;
`p0->next=p1`;

特殊情况:

1、若链表为空链表，将插入结点作为唯一的结点，
head=p0;返回。

2、若插入结点中的数据最小，则插入的结点作为
头结点。

p0->next=head;

head=p0;

3、插入到链尾，插入结点为最后一个结点。

p2->next=p0;

p0->next=NULL;

STU *insert(STU * head, STU * stud)

```
{ STU *p0,*p1,*p2;
```

```
  p1=head;  p0=stud;
```

```
  if (head==NULL)
```

链表为空

```
    { head=p0; p0->next=NULL;}
```

```
  else
```

```
    while((p0->num>p1->num)&&(p1->next!=NULL))
```

```
        { p2=p1; p1=p1->next; }
```

未找到结点，循环

```
    if (p0->num<=p1->num)
```

```
        { if (head==p1) head=p0;
```

插入在第一个结点前

找到结点

```
        else p2->next=p0;
```

```
        p0->next=p1;
```

```
    }
```

```
    else {p1->next=p0; p0->next=NULL;}
```

插入在最后一个后

```
  n=n+1;    return (head);
```

```
}
```



```
void main(void) { STU *head, stu;
int del_num;
head=creat( );
print(head);
cout<<“Input the deleted number:\n”;
cin>>del_num;
head=del(head,del_num);
print(head);
cout<<“Input the inserted record:\n”;
cin>>stu.num>>stu.score;
head=insert(head, &stu);
print(head);
}
```

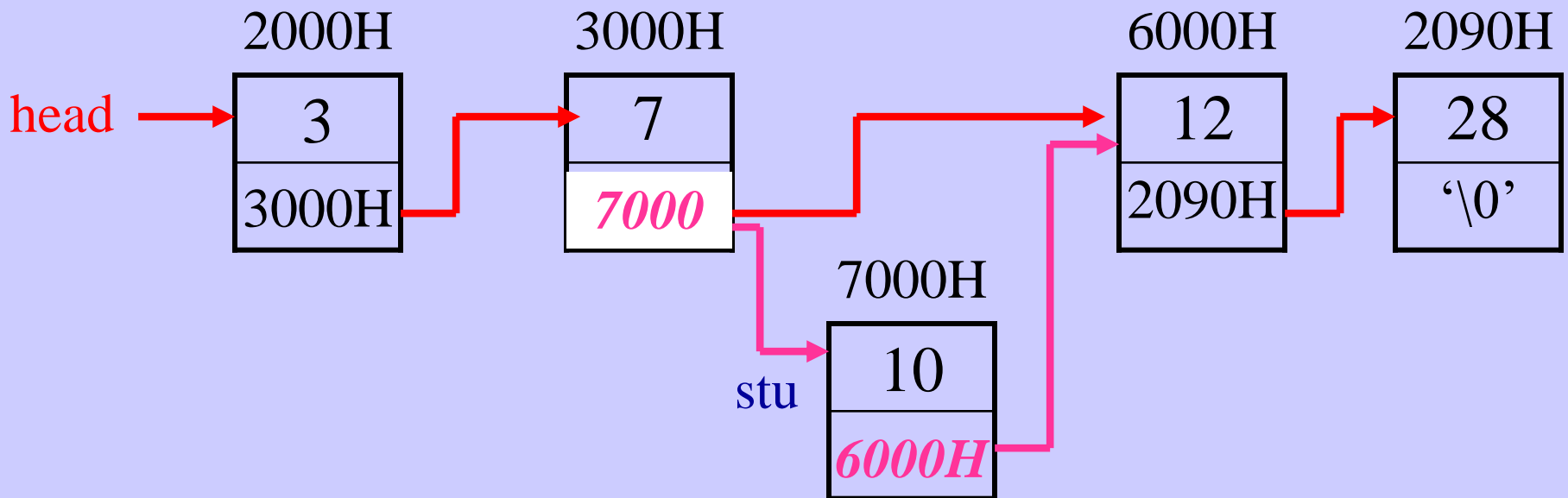
变量, 固定空间

建立链表

打印链表

删除结点

插入结点



```
void main(void)
```

```
{ STU *head,*stu;
```

```
int num;
```

指针，可以赋值

```
.....
```

```
cout<<“Input the inserted record:\n”;
```

```
stu=new student;
```

如果要插入结点，动态开辟新结点

```
cin>>stu->num>>stu->score;
```

```
while (stu->num!=0)
```

```
{ head=insert(head, stu);
```

```
print(head);
```

```
cout<<“Input the inserted record:\n”;
```

```
stu=new student;
```

重新开辟空间

```
cin>>stu->num>>stu->score;
```

```
}
```

```
}
```

用typedef定义类型

typedef定义新的类型来代替已有的类型。

<code>typedef</code>	已定义的类型	新的类型
----------------------	--------	------

```
typedef float REAL
```

REAL **x, y;** **float** **x, y;**

1、typedef可以定义类型，但不能定义变量。

2、tyoedef只能对已经存在的类型名重新定义一个类型名，而不能创建一个新的类型名。

```
typedef struct student REC x, y, *pt;
```

```
{ int i;
```

```
struct student x, y, *pt;
```

```
int *p;
```

} REC;

```
typedef char *CHARP;
```

```
CHARP p1,p2;          char *p1, *p2;
```

```
typedef char STRING[81];
```

```
STRING s1, s2, s3;    char s1[81], s2[81], s3[81];
```

1、先按定义变量的方法写出定义体 **char s[81];**

2、把变量名换成新类型名 **char STRING[81];**

3、在前面加typedef **typedef char STRING[81];**

4、再用新类型名定义变量 **STRING s;**

```
#define REAL float    编译前简单替换
```

typedef: 编译时处理，定义一个类型替代原有的类型。