

# 计算机程序设计基础

Programming Fundamentals

韩文弢

清华大学计算机系



# 第六章 指针

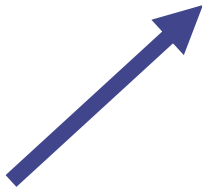
- 
- ① 基本概念
  - ② 指针变量
  - ③ 指针与数组
  - ④ 引用类型

## 6.1 基本概念

指针是C语言中最重要的特性之一，要成为一个好的 C/C++ 程序员，就必须理解指针的概念，并善于使用它。

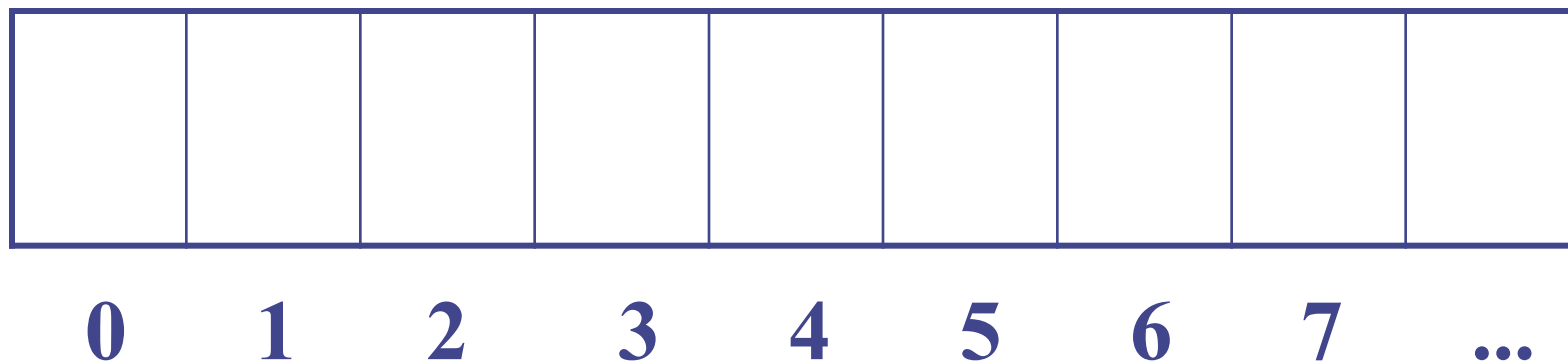


# 国家宝藏

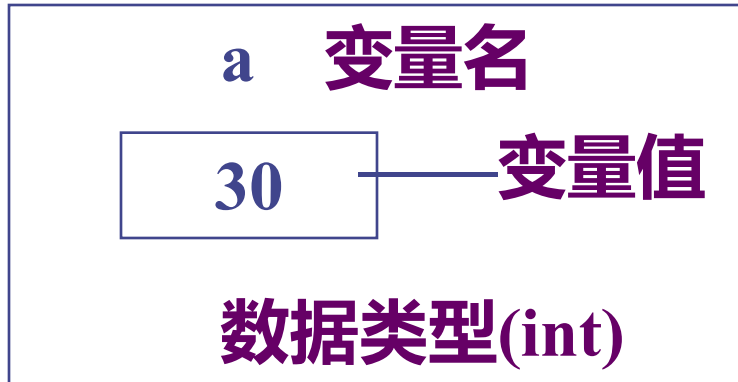


# Why 指针？

思考：如何访问内存中的数据？

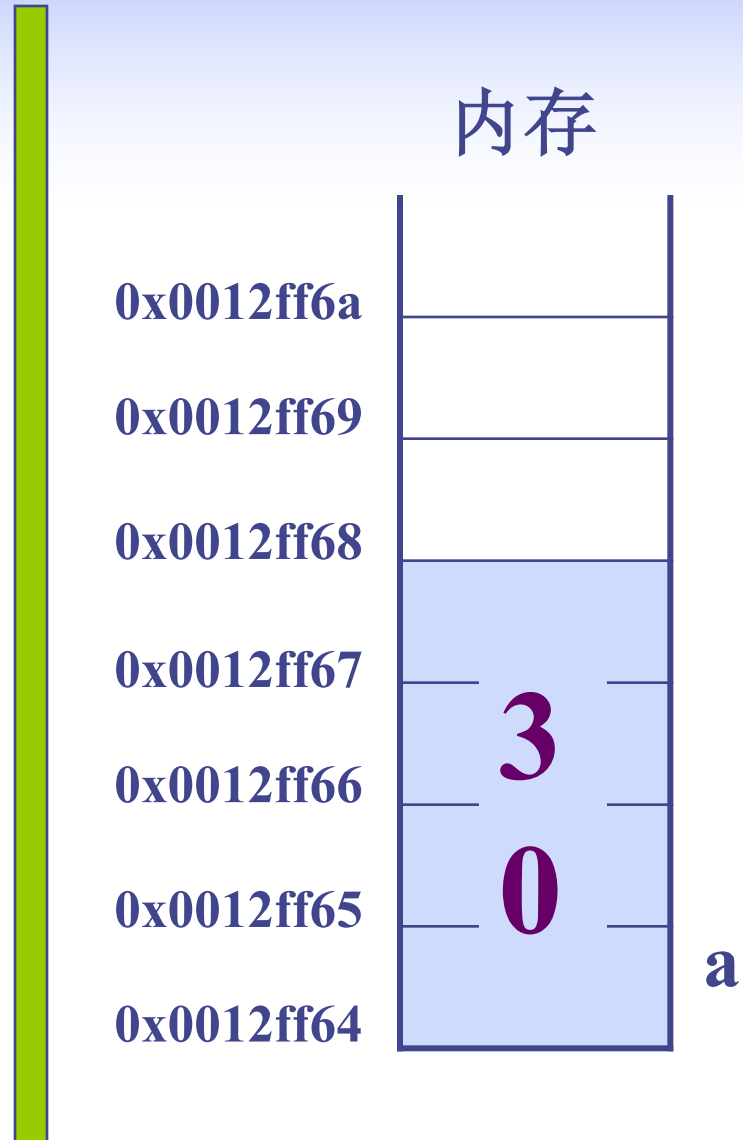


1. 字节为单位
2. 数据长度不同



高手的苦恼：

1. 内存有多大？
2. 如何指哪打哪？



# 如何访问任意内存单元中的数据？

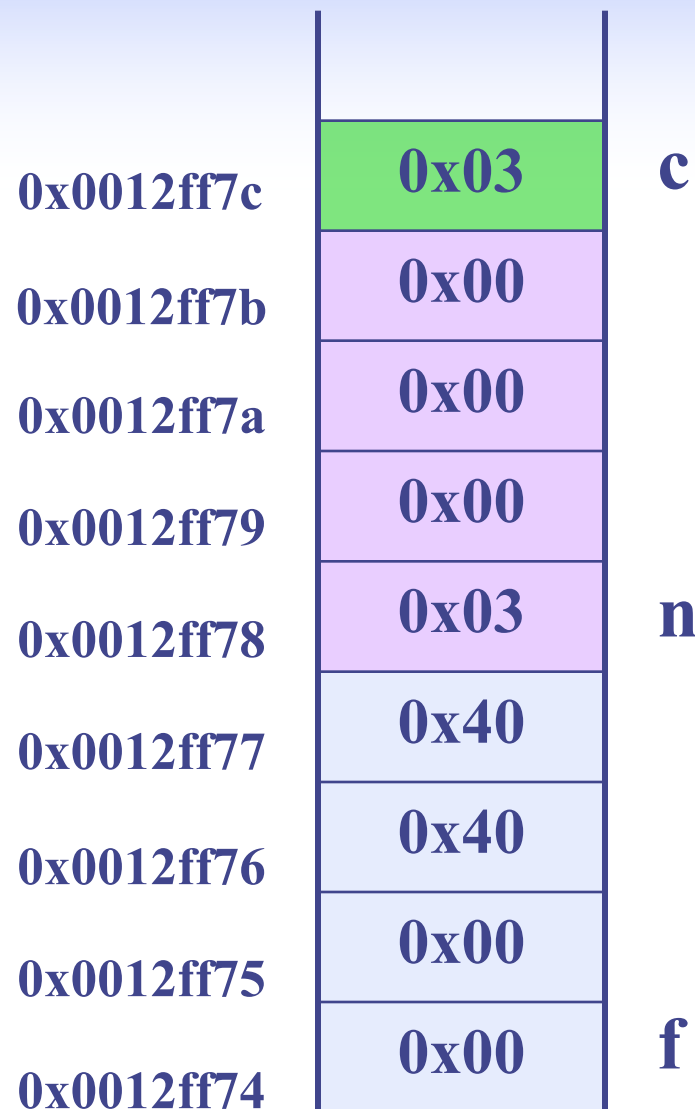
访问一个数据需要知道：

- 它的起始地址
- 它的数据类型

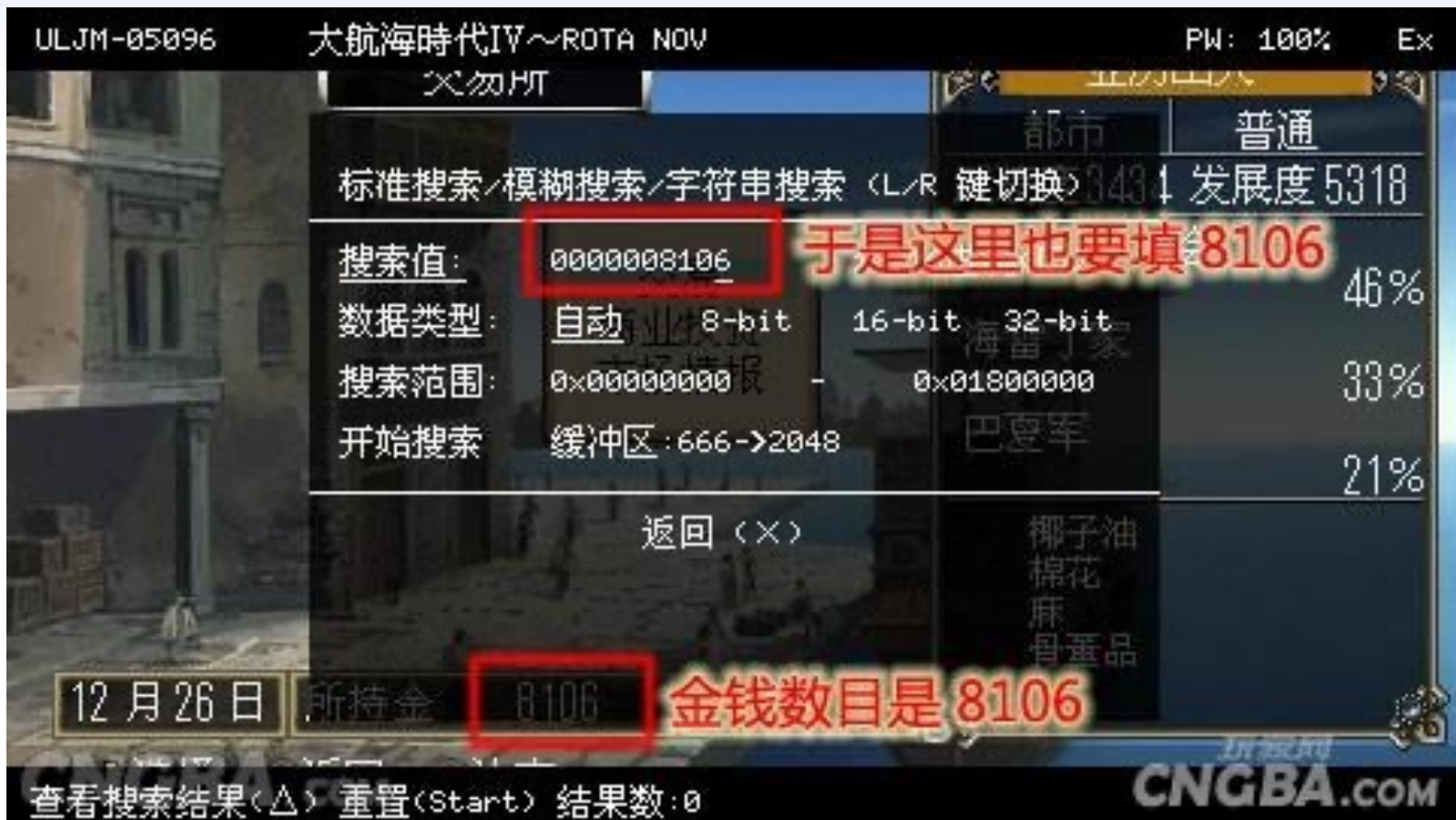
```
char c;
int n;
float f;
```

```
c = 3;      // 整数3
n = 3;      // 整数3
f = 3;      // 实数3
```

起始地址 + 数据类型 = 指哪打哪







# 什么是指针？

The C Programming Language (K & R):

*A pointer is a variable that contains the memory address of another variable.*

## 指针是一种特殊的变量：

1. 变量名：与一般的变量命名规则相同；
2. 变量的值：是另一个变量的内存地址；
3. 变量的类型：包括两种类型，
  - 指针变量本身的类型，即**指针类型**，它的长度为4个字节；
  - 指针**指向**的变量（数据）的类型。

指针 = 变量  $\neq$  地址

买椟还珠...

# 第六章 指针

- 
- ① 基本概念
  - ② 指针变量
  - ③ 指针与数组
  - ④ 引用类型

## 6.2.1 指针的定义

指针定义的一般形式:

<b>基类型</b> <b>*指针变量名;</b>
---------------------------

例如:

```
int x;      /* 定义了一个整型变量 */
```

```
int *p;      /* 定义了一个指向整型变量的指针变量p */
```

现在我们有了一种新的数据类型：

**int \***       “pointer to int”

**double \***   “pointer to double”

**char \***      “pointer to char”

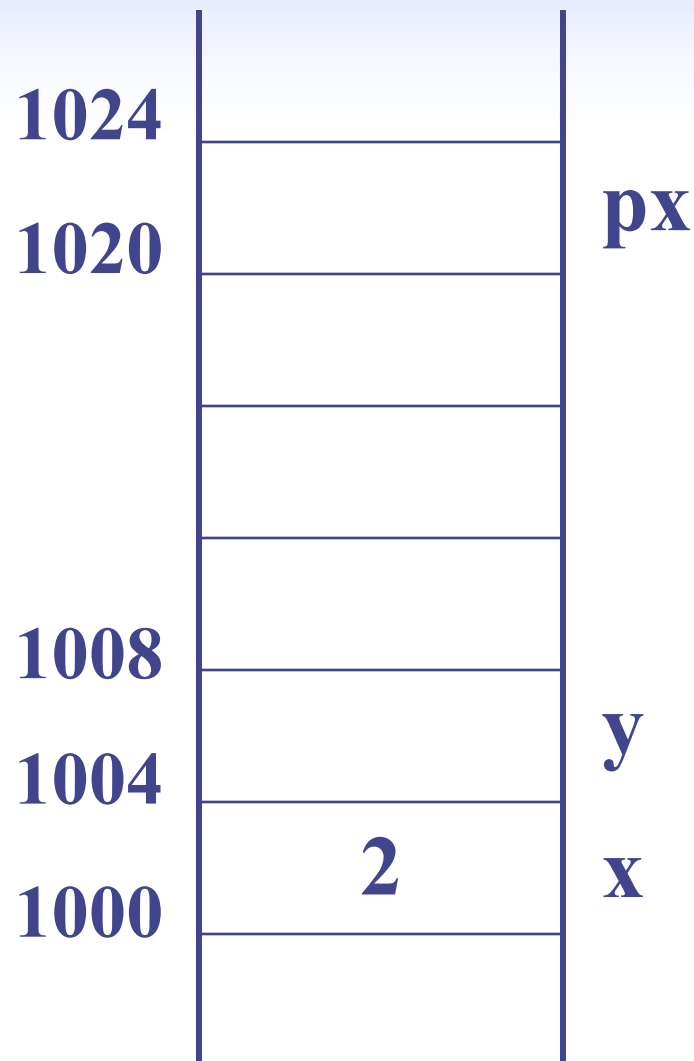
相同点？ 不同点？

```
int x, y, *px;
```

```
x = 2;
```

一些问题:

1. 如何把变量 **x** 的地址存放在指针 **px** 当中?
2. 如何使用指针 **px** 来访问或修改变量 **x** 的值?
3. 为何要如此自找麻烦?





## 6.2.2 &运算符

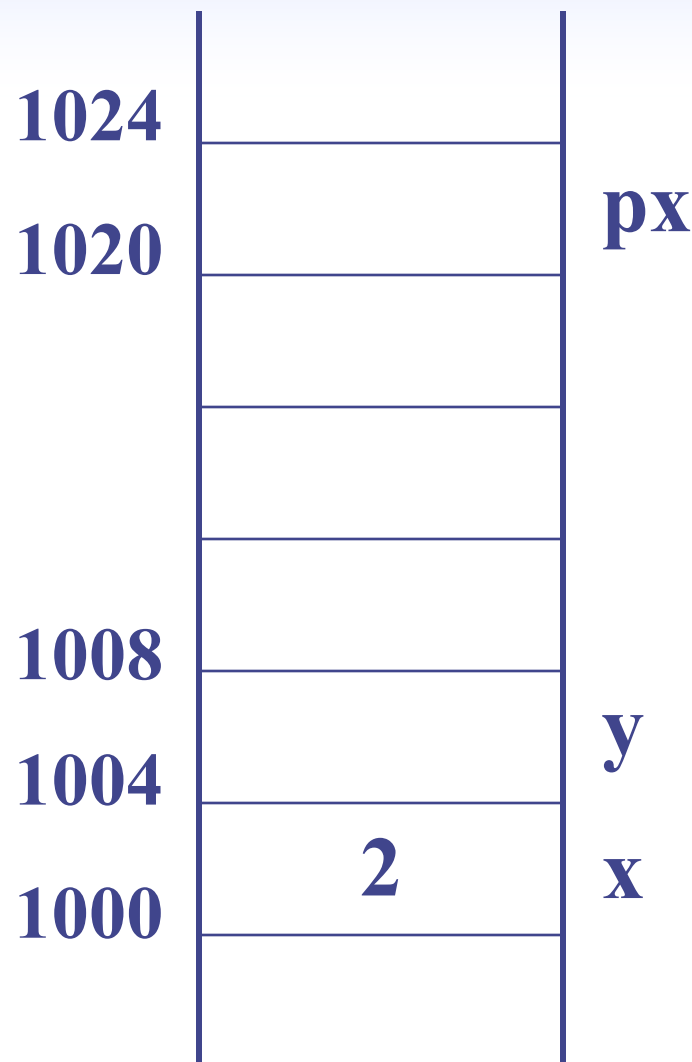
```
int x, y, *px;
```

```
x = 2;
```

1. 如何把变量 **x** 的地址存放在指针 **px** 当中?

能否: **px = 1000;**

- 1000从何而来?
- 类型不匹配。



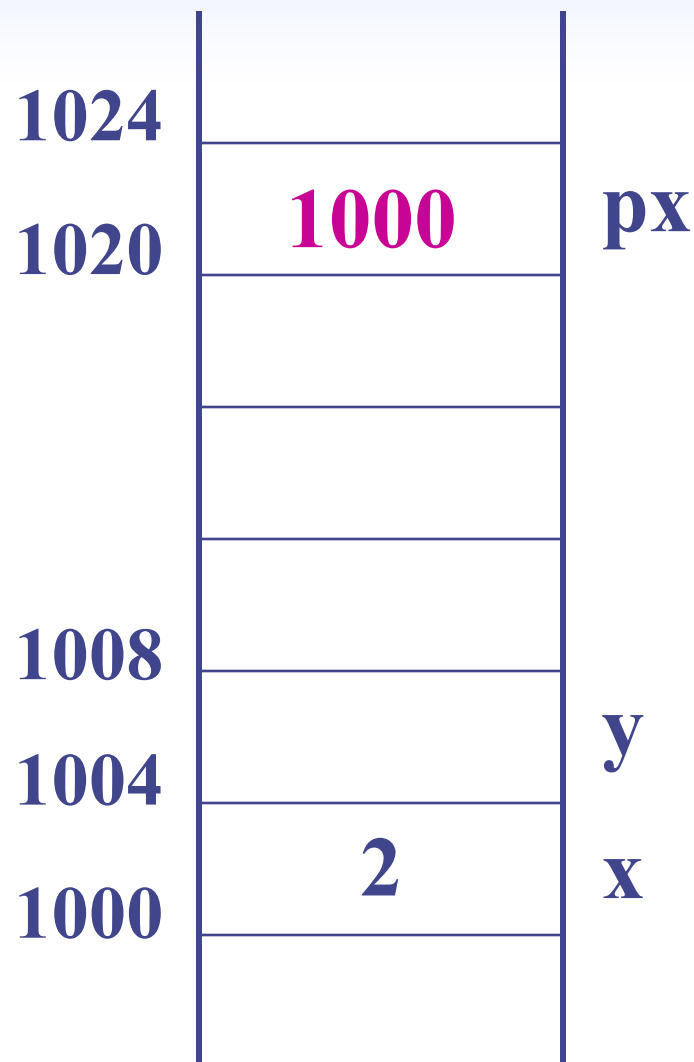
我们想要做的事情：

**px** = **x** 的内存地址；

我们写的语句：

**px = &x;**

**&：** 取地址运算符



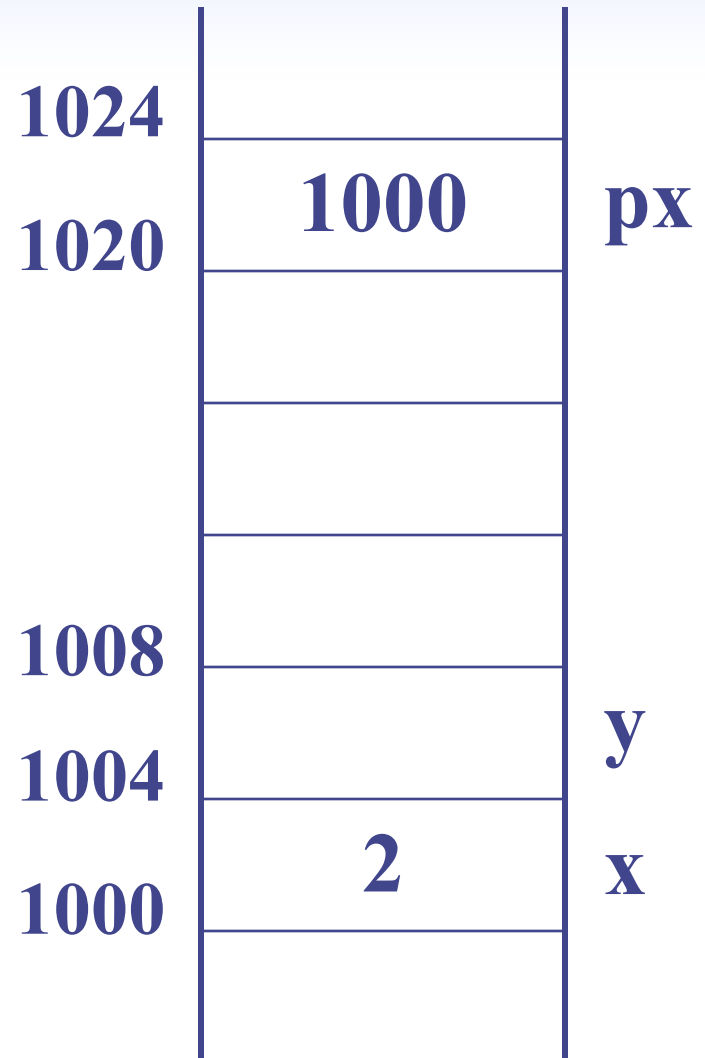
## 6.2.3 \*运算符

```
int x, y, *px;
```

```
x = 2;
```

```
px = &x;
```

2. 如何通过指针 **px** 来访问变量 **x** 的值?



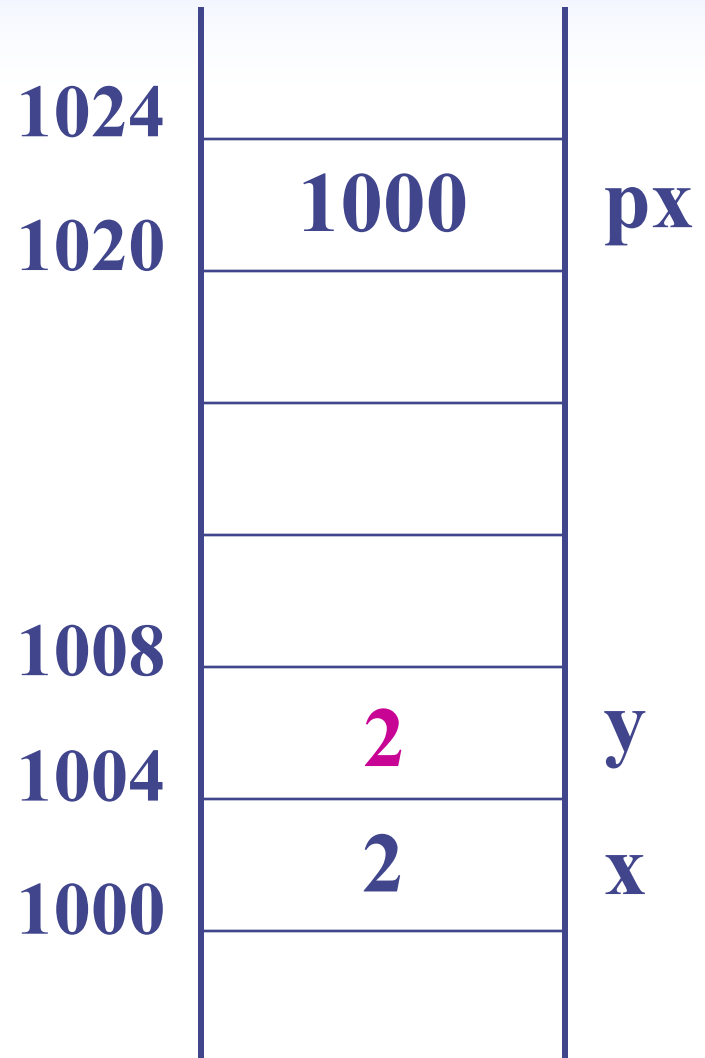
我们想要做的事情：

$y = \text{px}$ 所指向的变量的值；

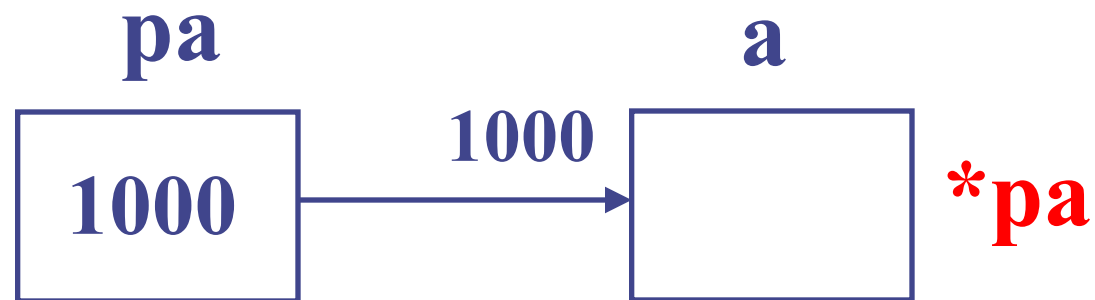
我们写的语句：

$y = *px;$

\*：指针运算符（或称“间接访问”运算符）



**pa = &a;**



```
int  a, b, c, *p, *q;
```

```
a  =  1;
```

```
b  =  2;
```

```
c  =  3;
```

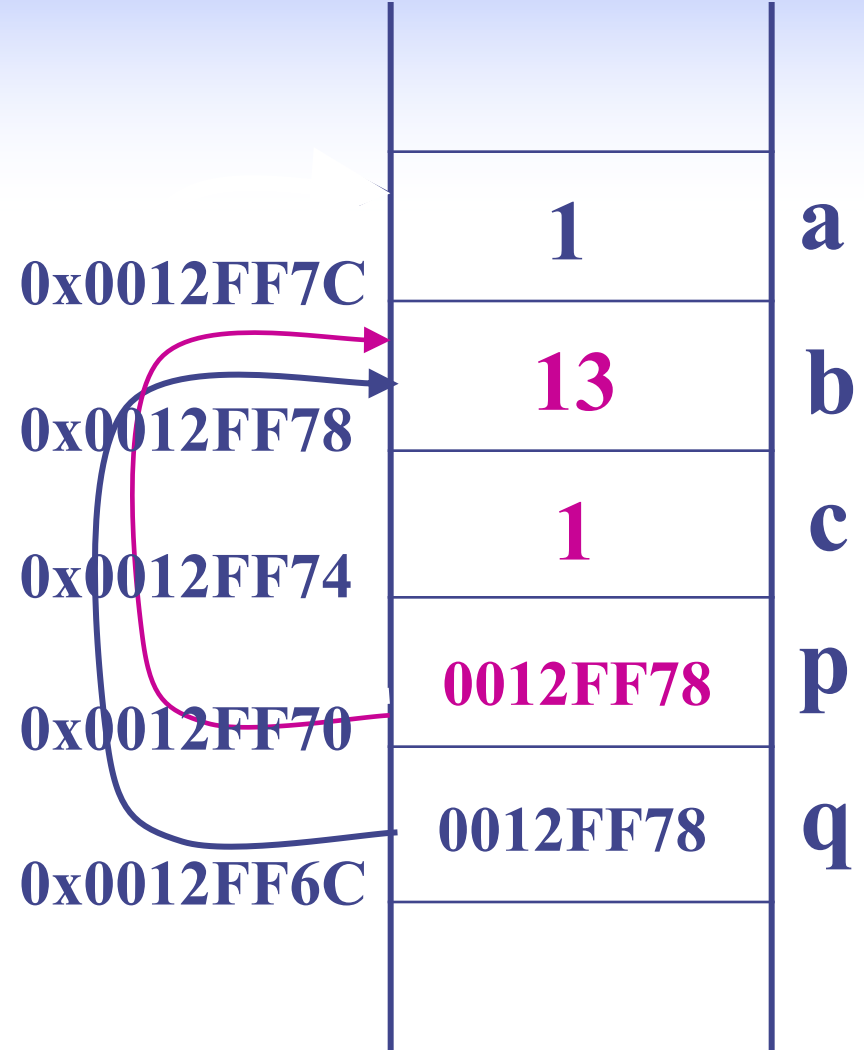
```
p  =  &a;
```

```
q  =  &b;
```

```
c  =  *p;
```

```
p  =  q;
```

```
*p  =  13;
```



- 指针常量NULL表示一个指针不指向任何有效的数据。

- `int x, *p;`

`p = &x;`

`*p = 1;`

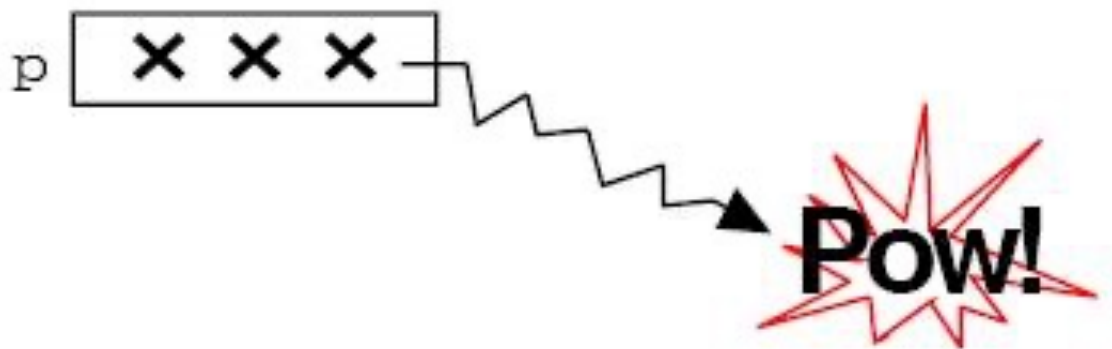
`p = NULL;`

`*p = 2;`



`<stdio.h>`

一个其值为NULL的指针不同于一个未初始化的指针。一个指针在定义后，是未被初始化的，此时若对它进行访问，将会出错。



今天在公司听到一句惨绝人寰的骂程序员的话：

“你这个没有对象的野指针！”



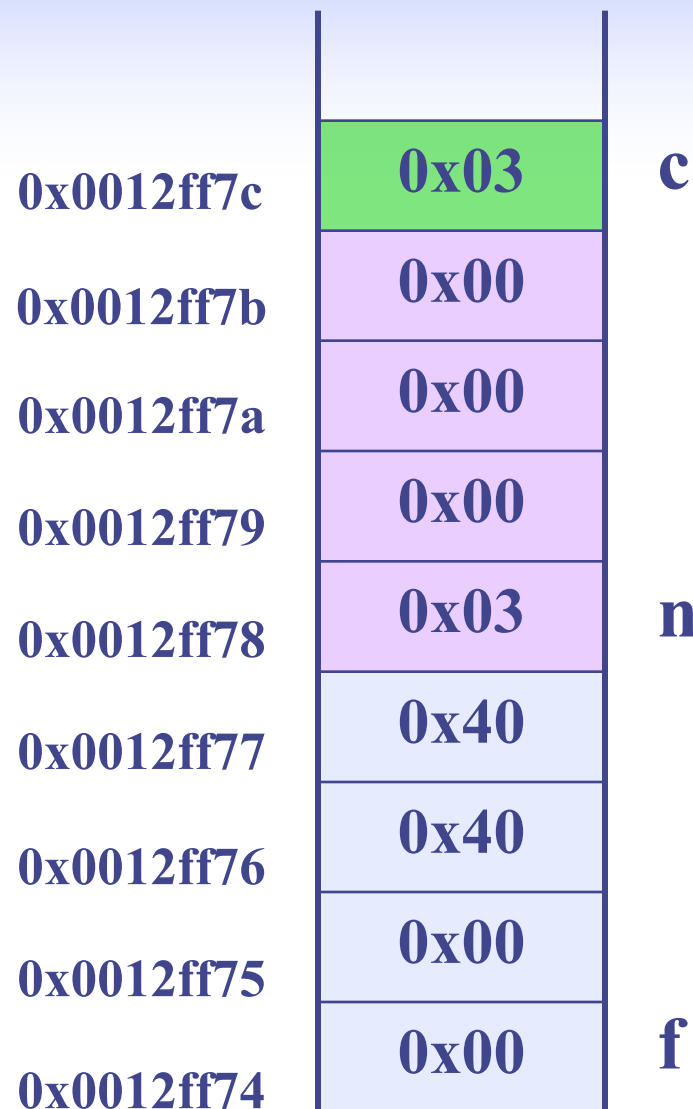
# 分析程序

```
int foobar(int *pi)
{
    *pi = 1024;
    return *pi;
}
int main()
{
    int *pi2;
    int ival = foobar(pi2);
    cout << ival;
    return 0;
}
```

```
char c;
int n;
float f;
```

```
c = 3;      // 整数3
n = 3;      // 整数3
f = 3;      // 实数3
```

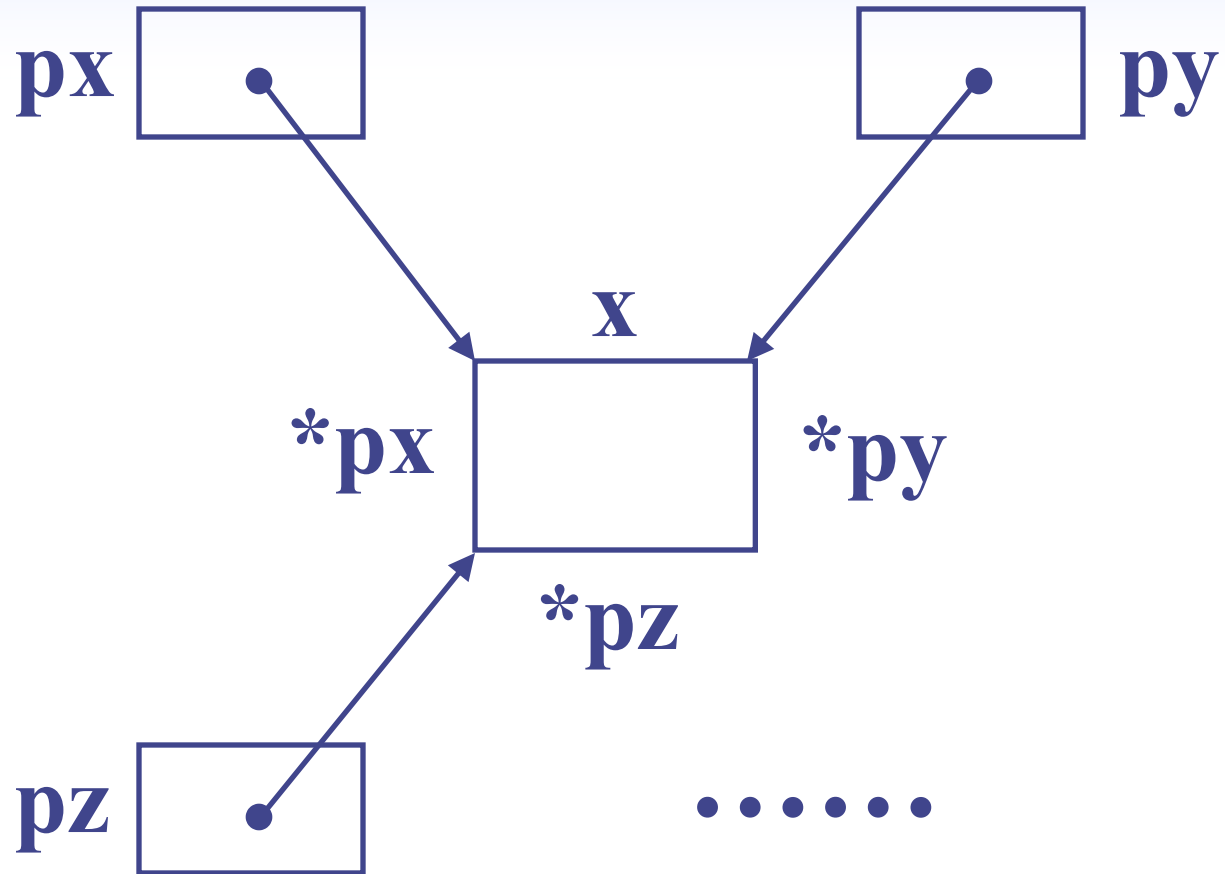
如何实现游戏修改器？



# 指针的基本概念.....

## 6.2.4 指针能做什么？

用处之一：提供了一种共享数据的方法，可以在程序的不同位置、使用不同的名字（指针）来访问相同的一段共享数据。如果在一个地方对该数据进行了修改，那么在其他的地方都能看到修改以后的结果。



```
void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
void main( )
{
    int a, b;

    a = 4;
    b = 7;
    swap(a, b);
    cout << a << ' ' << b;
}
```

输出结果:

4, 7

main的栈帧	
a	b
4	7

swap函数的栈帧		
x	y	temp
7	4	

```

void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

void main( )
{
    int a, b;
    a = 4;
    b = 7;
    swap(a, b);
    cout << a << ' ' << b;
}

```

```

void swap(int *pa, int *pb)
{
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}

```

```

void main( )      a、b的访问?
{

```

```

    int a, b;
    a = 4;
    b = 7;
    swap(&a, &b);
    cout << a << ' ' << b;
}

```

main的栈帧	
a	b
7	4

swap的栈帧		
temp	pa	pb





# 传值还是传地址？

- 传值：给予
- 传地址：索取

问题描述：计算一元二次方程的根。

$$a x^2 + b x + c = 0$$

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

```
void main( )
{
    double  a, b, c, x1, x2;

    /* 从键盘读入方程式的系数a, b, c */
    GetCoefficients(?a, ?b, ?c);

    /* 求解方程式的两个根x1, x2 */
    SolveQuadratic(?a, ?b, ?c, ?x1, ?x2);

    /* 显示方程式的两个根x1, x2 */
    DisplayRoots(?x1, ?x2);
}
```

```
main( )
{
    double  a, b, c, x1, x2;

    /* 从键盘读入方程式的系数a, b, c */
    GetCoefficients(&a, &b, &c);

    /* 求解方程式的两个根x1, x2 */
    SolveQuadratic(a, b, c, &x1, &x2);

    /* 显示方程式的两个根x1, x2 */
    DisplayRoots(x1, x2);
}
```

## 6.2.5 指针分析

```
void main()    /* 有何问题? */
{
    int binky;
    foo(&binky);
}
void foo(int *tinky)
{
    int slinky = 5;
    tinky = &slinky; *tinky = slinky;
}
```

```
void main() /* 输出结果是什么? */
{
    int* pinky;
    pinky = bar();
    cout << *pinky;
}
int* bar()
{
    int winky = 5;
    return(&winky);
}
```

*thinking...*

# 第六章 指针

- 
- ① 基本概念
  - ② 指针变量
  - ③ 指针与数组
  - ④ 引用类型

## 6.3.1 数组元素的访问

数组的定义: `int a[5];`

空间的分配: `a`

--	--	--	--	--

The diagram shows a horizontal array of five empty cells, each representing an element of the array `a`. The labels `a[0]`, `a[1]`, `a[2]`, `a[3]`, and `a[4]` are positioned above the corresponding cells.

元素的访问:

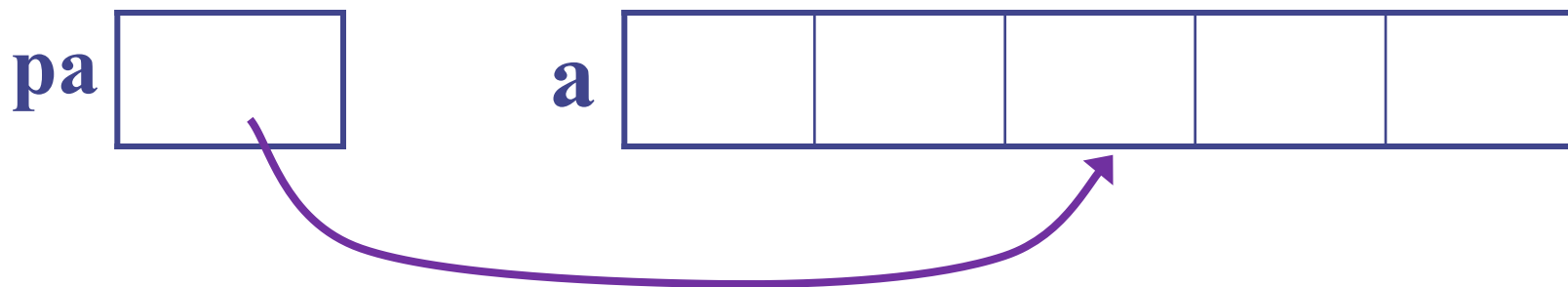
```
for (i = 0; i < 5; i++)  
{  
    a[i] = 0;  
}
```



# 通过指针来访问数组元素

指针

数组



`pa = &a[2]`

# 指针的算术和关系运算

指针加一个整数:  $\text{ptr} + k$

指针减一个整数:  $\text{ptr} - k$

两个指针相减:  $p1 - p2$

指针的关系运算:  $>$ 、 $\geq$ 、 $<$   
 $\leq$ 、 $==$ 、 $!=$

- 指针的算术运算是以数据元素为单元;
- `int *p;`  
`p = (int *)1000;`  
`p = p + 2;`
- `p+k`的计算方法是: **`p+k*sizeof(int)`**  
基类型长度的补偿由系统自动完成。

```
int a[5], *pa;
```

想做的事情	编写的代码
把a的起始地址放入pa	<pre>pa = a; pa = &amp;a[0];</pre>
pa指向第i个数组元素	<pre>pa = &amp;a[i]; pa = a + i;</pre>
访问第i个数组元素	<pre>*(pa + i) pa[i]</pre>

```

void main( )
{
    int salary, nCars, nHouses;
    salary = 6000;
    nCars = 0;
    nHouses = 0;
    DayDreaming(salary, nCars, nHouses);
    cout<<salary<<' '<<nCars<<' '<<nHouses;
}
void DayDreaming(int salary, int cars, int houses)
{
    salary = salary * 3;
    cars += 2;
    houses ++;
}

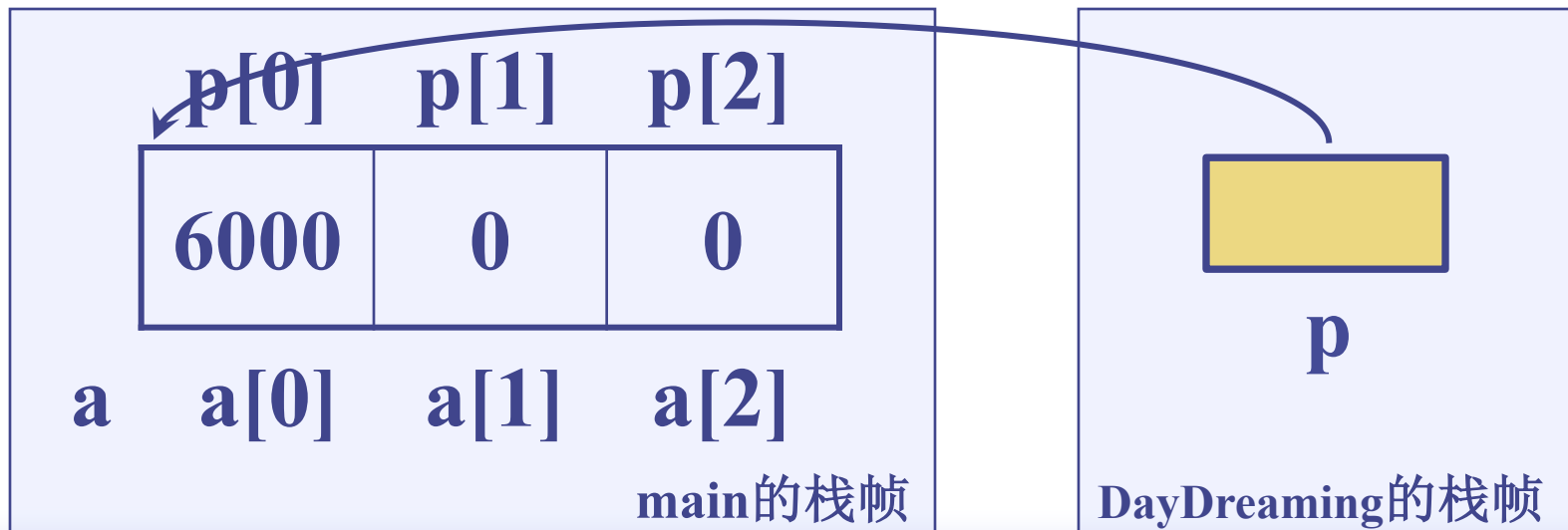
```

```

void main( )
{
    int a[3];
    a[0] = 6000;
    a[1] = 0;
    a[2] = 0;
    DayDreaming(a);
    cout<<a[0]<<' '<<a[1]<<' '<<a[2];
}
void DayDreaming(int p[3])
{
    p[0] = p[0] * 3;
    p[1] += 2;
    p[2] ++;
}

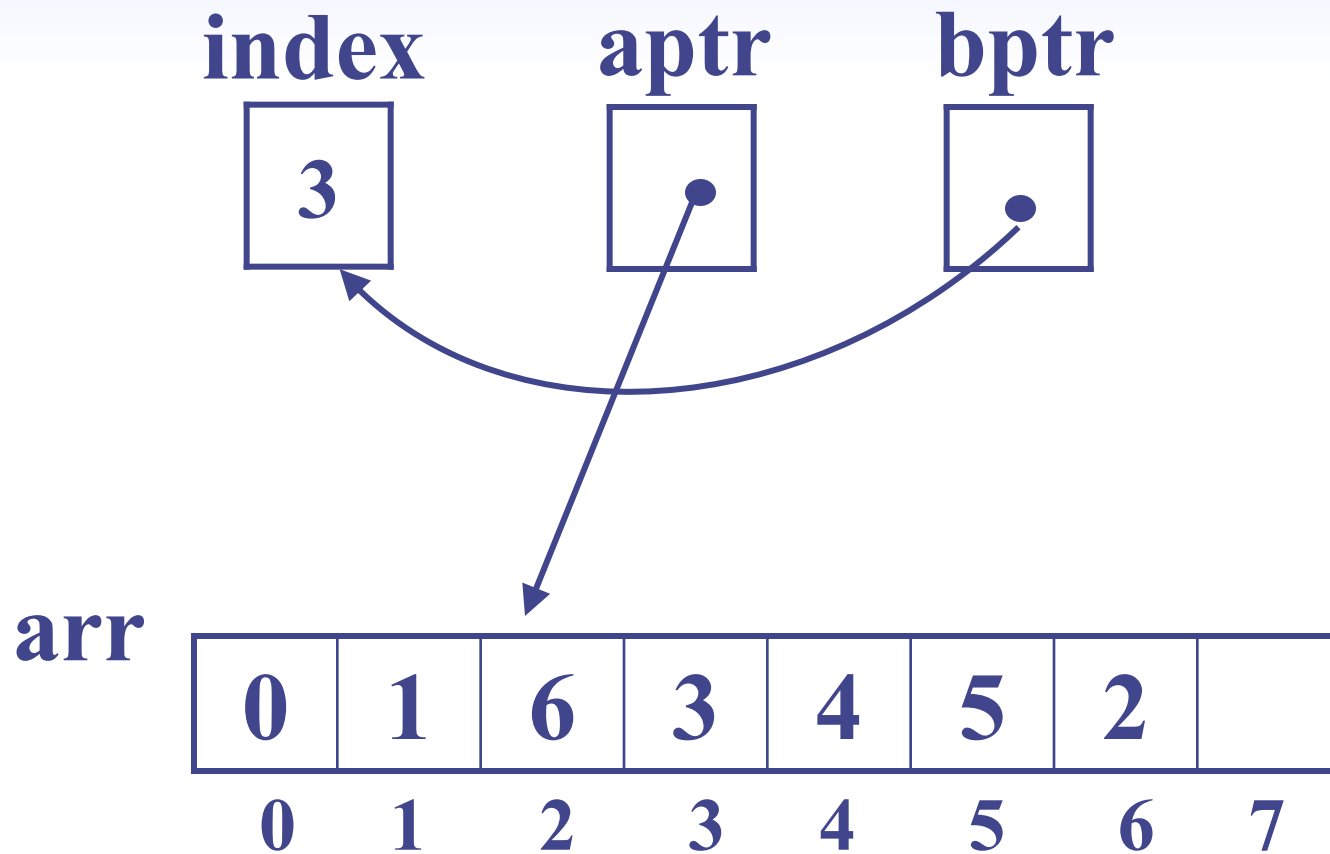
```

在函数调用时，传地址而不传值



# 代码分析

```
void main()  
{  
    int index, *aptr, *bptr;  
    int arr[8];  
    aptr = &index;  
    for(index=0; index<6; index++)  
        arr[index] = index;  
    bptr = aptr;  
    aptr = &arr[2];  
    arr[*bptr] = *aptr;  
    *aptr = *bptr;  
    *bptr = *(aptr + 1);  
    画出此时内存状态（变量的取值）  
}
```



## 6.3.2 动态数组

在定义一个数组时，必须事先指定其长度。

例如：

```
int a[6];
```

或者：

```
#define MAX_SIZE 100;  
int scores[MAX_SIZE];
```



# 如果在编程时并不知道数组的确切长度？

☺ 太小了：装不下

☺ 太大了：浪费内存空间



有时，数组的长度只有当程序运行以后才知道。因此，我们希望能这样定义数组：

```
int  n;  
cout << "请输入学生人数: ";  
cin  >> n;  
int  scores[n];
```

但是在C++语言中，这是**不可能**的。

我们能做的事情是：动态地为该数组分配所需的内存空间，即在程序运行时分配。具体做法是：定义一个指针，然后把动态分配的内存空间的起始地址保存在该指针中，如：

```
int *scores;
```

```
scores = 动态分配的内存空间的起始地址;
```

# 动态空间

## **new** 运算符

功能：申请一块动态内存空间

使用方法：可以是单个变量或数组

返回值：一个指向该内存的指针

```
int *p;
```

```
p = new int; // int是类型参数，用来确定空间大小
```

# 释放空间

## delete 运算符

功能：释放一块动态内存空间

参数：一个指向一块动态空间的指针

```
int *p;
```

```
p = new int;
```

```
*p = 2;    // 使用动态空间
```

```
delete p;  // 释放动态空间，但不能重复释放
```

# 动态数组

```
int *scores, N;
```

```
cin >> N;
```

```
scores = new int[N]; // N是变量
```

```
scores[3] = 99;
```



等价于: `*(scores + 3) = 99;`

```
delete[] scores;
```

作为局部变量的数组在函数调用结束后，其内存空间即被释放。而对于动态数组，即使在函数调用结束后，其内存空间依然存在。

# 程序 分析

```
#define    MAX_SIZE    10
void 4 4344416 1244916 4198895 1
{
    int a[5] = {1, -1, 2, -2, 0};
    int b[5] = {3, 1, -2, 4, 1};
    int *c, i;
    c = Add(a, b, 5);
    for(i = 0; i < 5; i++)
        cout << c[i] << ' ';
}
int *Add(int a[], int b[], int num)
{
    int i, c[MAX_SIZE];
    for (i = 0; i < num; i++)
        c[i] = a[i] + b[i];
    return c;
}
```

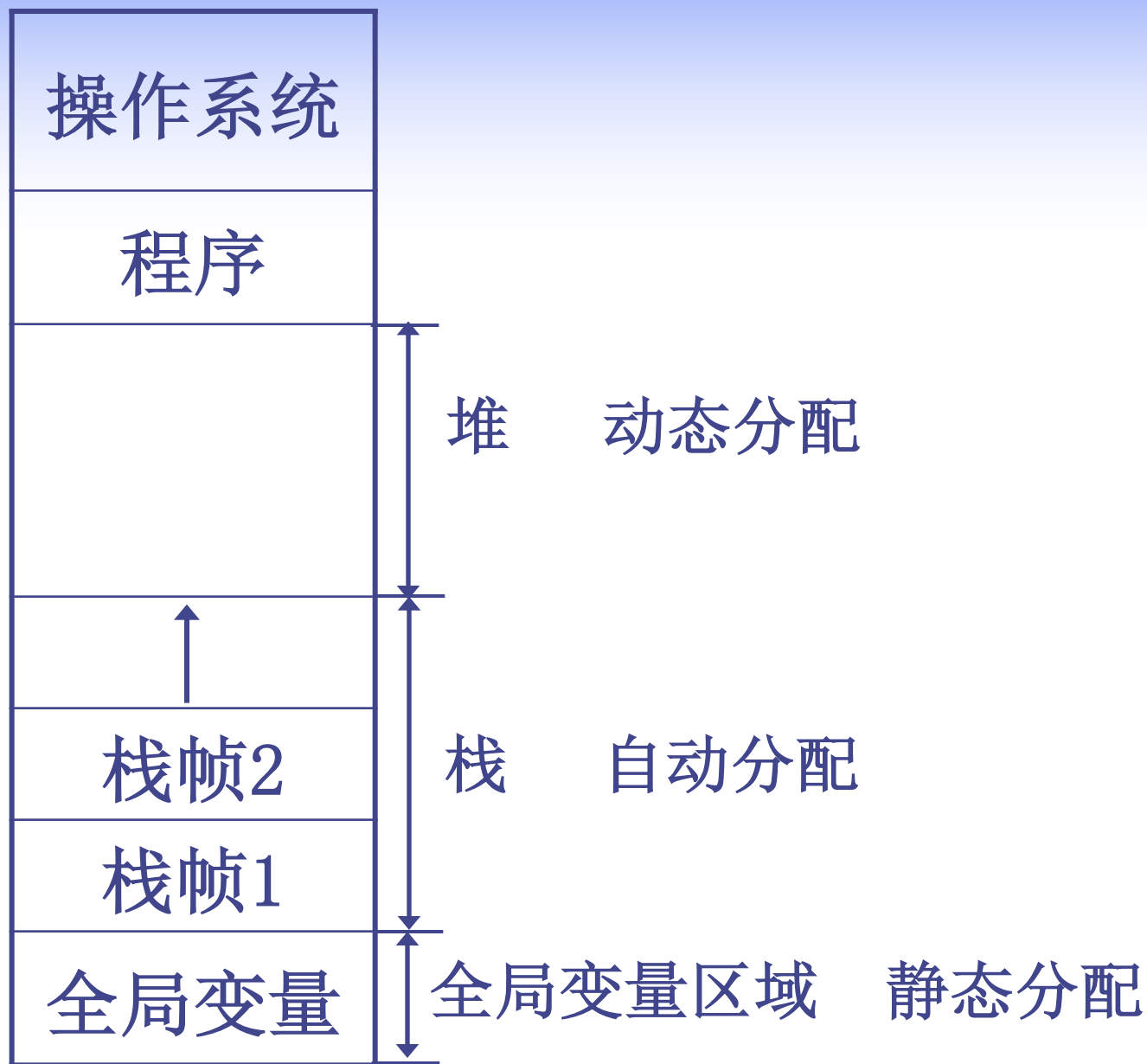


4 0 0 2 1

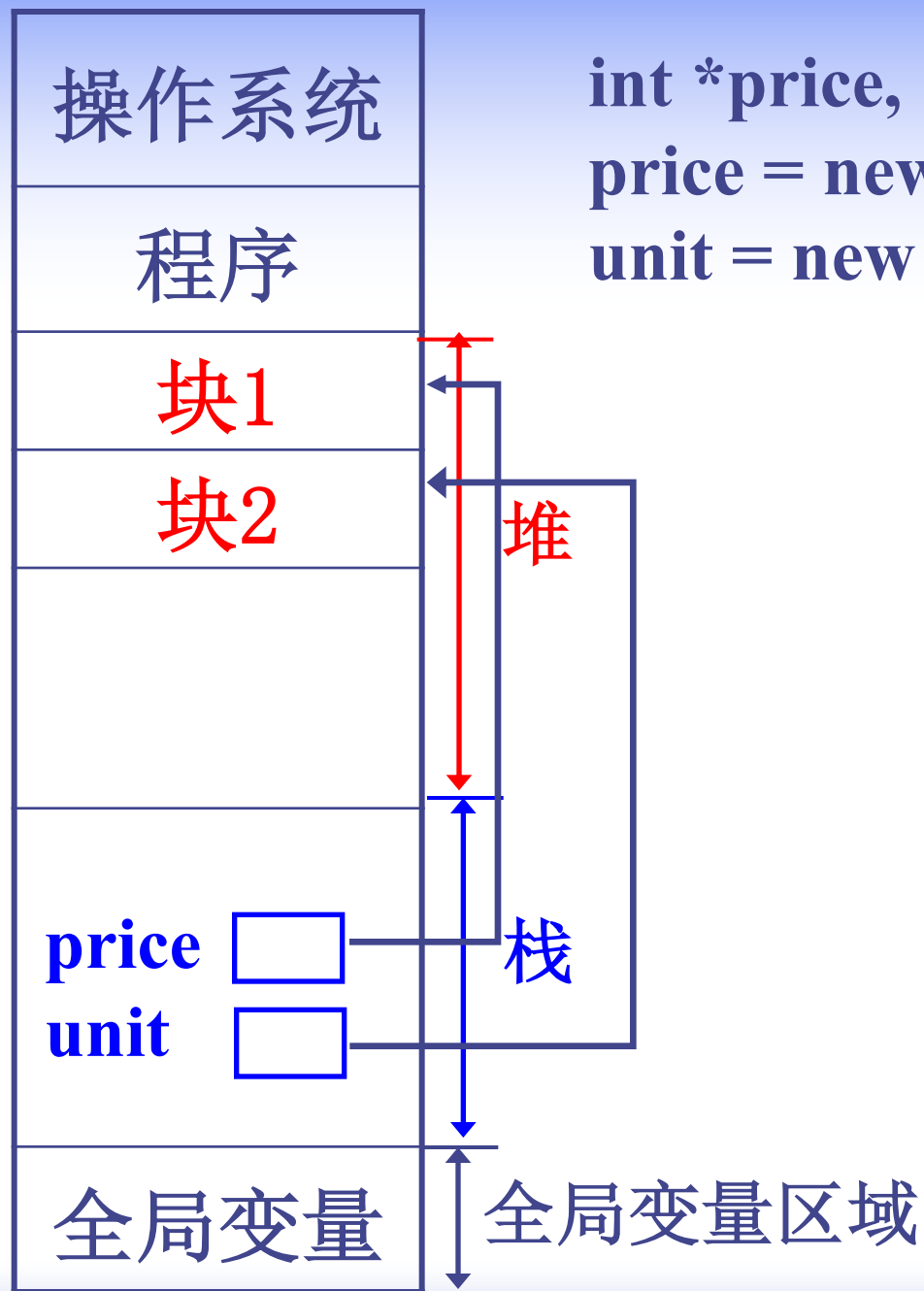
```
int main( )
{
    int a[5] = {1, -1, 2, -2, 0};
    int b[5] = {3, 1, -2, 4, 1};
    int *c, i;
    c = Add(a, b, 5);
    for(i=0; i<5; i++) cout << c[i] << ' ';
    delete[] c;
    return 0;
}

int *Add(int a[], int b[], int num)
{
    int i, *c;
    c = new int[num];
    for (i = 0; i < num; i++)
        c[i] = a[i] + b[i];
    return c;
}
```

# 内存分布状况



# 内存分布状况



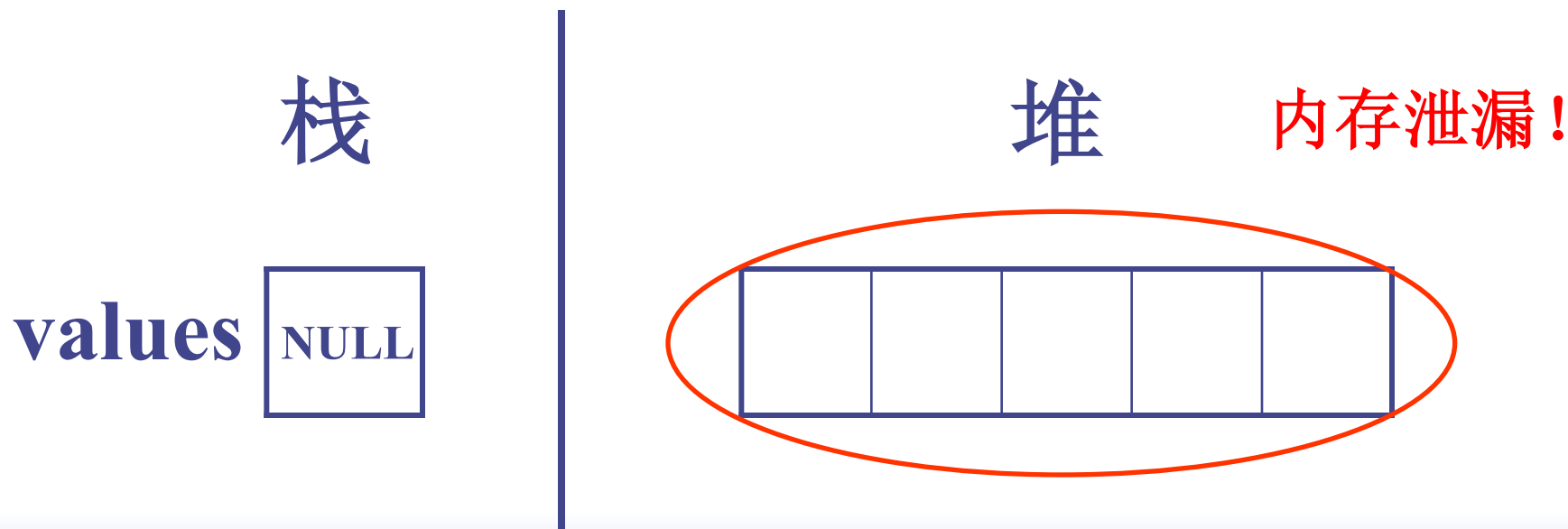
# 内存泄漏

```
values = new int[5];
```

```
delete[] values;
```

```
values = NULL;
```

values原来指向的内存单元现在无法访问。



# 代码分析

```
void main()  
{  
    int i, *a, *b, list[4];  
    a = &i;  
    for (i = 0; i < 4; i++)  
        list[i] = *a;  
    b = new int[4];  
    a = list + 2;  
    *b = *a;  
    b++;  
    画出此时内存状态（栈、堆、变量的取值）  
}
```

main

栈

i

4

a

b

list

0

1

2

3

堆

2

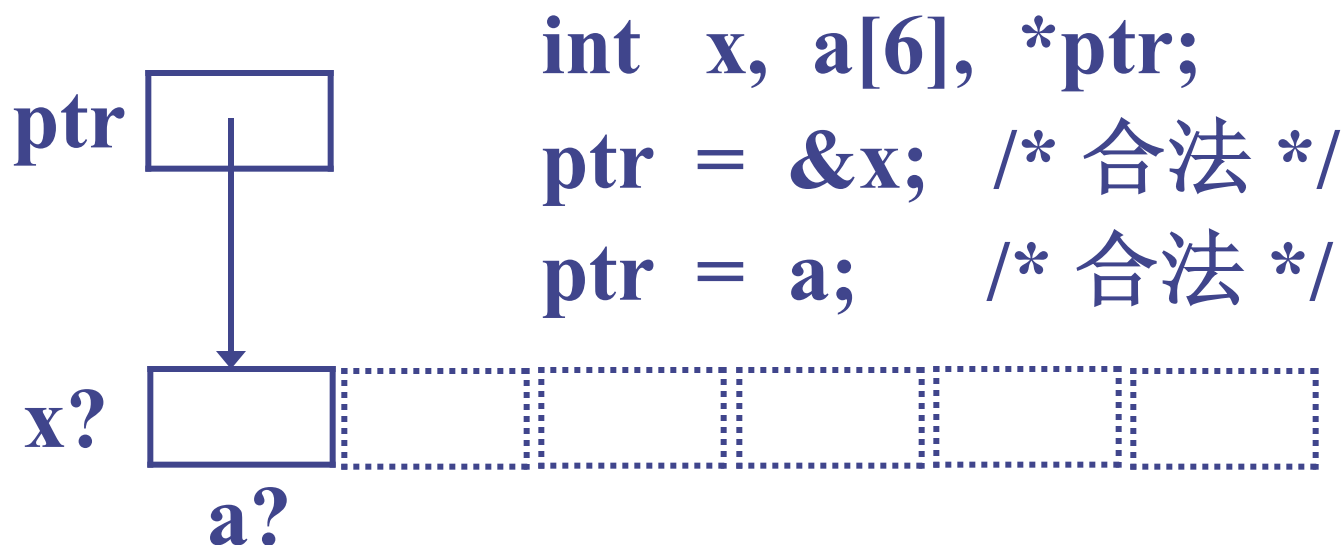
?

?

?

## 6.3.3 指针的两种用法

一个指针可能是指向单一的一个变量，也可能是指向一组相同类型的数组元素，从外观上，两种指针的形式完全一样。



# 分析结果

```
void bar(int p2[ ])  
{  
    p2[1] = 15;  
}
```

```
void foo(int p1[ ])  
{  
    *p1 += 5;  
}
```

1 15 5

2 9 15

```
void main( )  
{  
    int a[ ] = {1,3,5};  
    int b[ ] = {2,4,6};  
    int *p;  
    p = &a[0];  
    bar(p);  
    cout<<a[0]<<' '<<a[1]<<' '<<a[2]<<endl;  
    p = &b[0];  
    p ++;  
    foo(p);  
    bar(p);  
    cout<<b[0]<<' '<<b[1]<<' '<<b[2]<<endl;  
}
```



## 6.3.4 指针数组

一个数组，其元素均为指针类型变量，称为**指针数组**。即数组中的每一个元素都是一个指针。

定义形式：类型名 \*数组名[数组长度]；




例如：

```
int *pa[4];
```

指针数组作为**main**函数的形参

**main(int argc, char \*argv[ ]);**

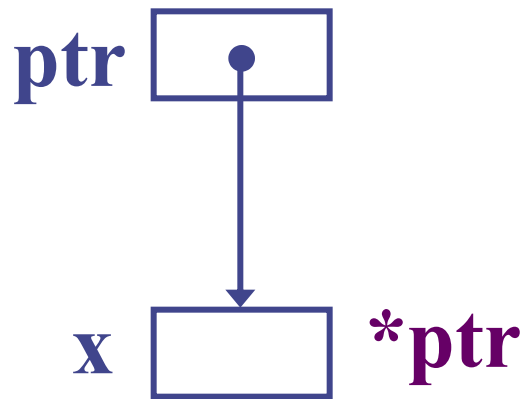
例如，假设程序名为**sort**，在运行时命令行的情况如下：

<b>sort</b>	<b>source.txt</b>	<b>destination.txt</b>
		
<b>argv[0]</b>	<b>argv[1]</b>	<b>argv[2]</b>
<b>argc = 3</b>		

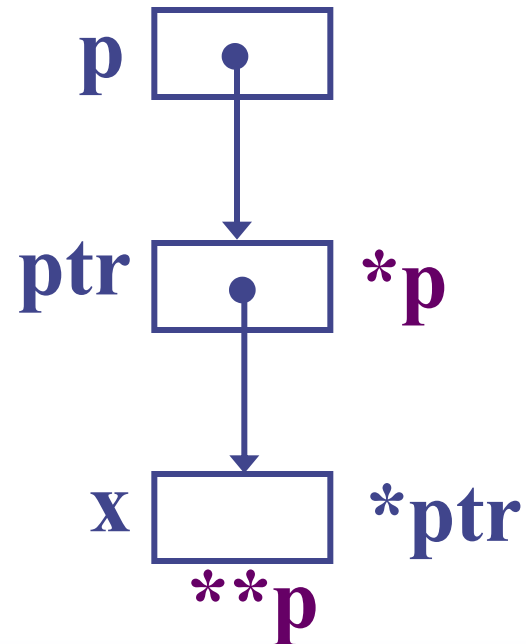
## 6.3.5 指向指针的指针

一种特殊的指针，其基类型也为指针类型。  
如 `int **p`。

通常的指针：



指向指针的指针：



已有的三种指针类型：

**int \***       “pointer to int”

**double \***   “pointer to double”

**char \***     “pointer to char”

现在又可以增加三种新的数据类型：

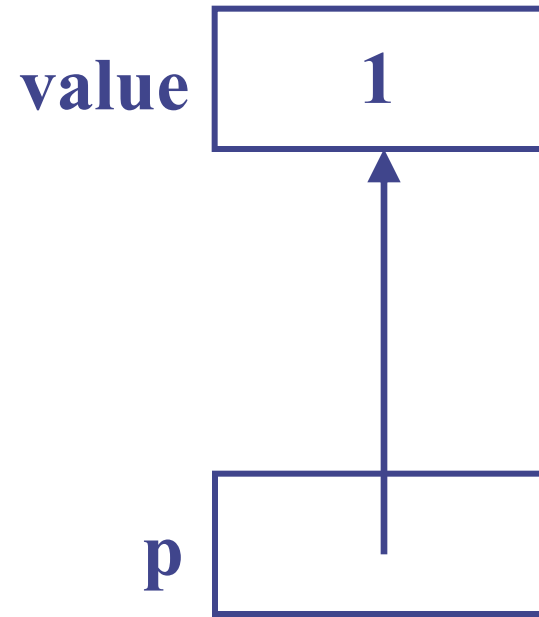
**int \*\***       “pointer to pointer to int”

**double \*\***   “pointer to pointer to double”

**char \*\***     “pointer to pointer to char”

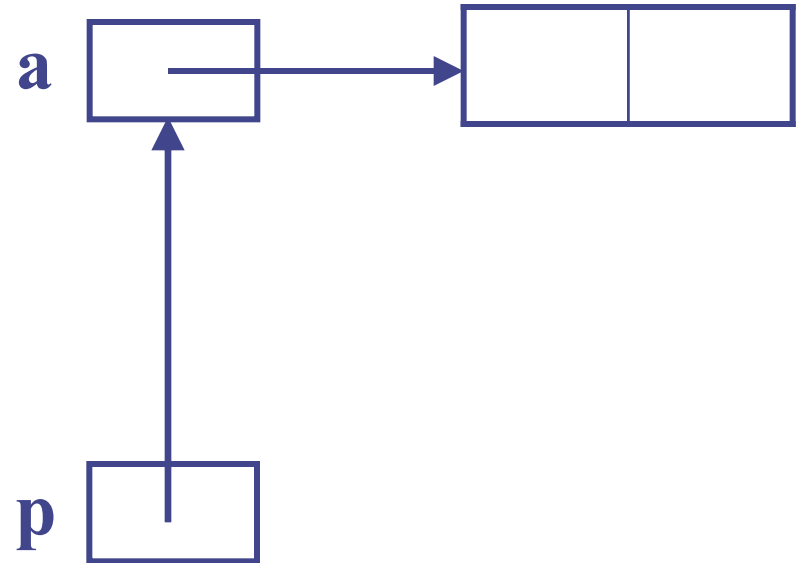
# 在foo函数中修改主函数中的变量

```
void main()  
{  
    int value;  
    foo(???);  
}  
  
void foo(???) //p  
{  
    ??? = 1;  
}
```



# 如果主函数中的变量是一个指针呢？

```
void main()  
{  
    int *a;  
    foo(???);  
}  
  
void foo(???) //p  
{  
    ??? = new int[2];  
}
```



```
void main( )
{
    int homer, *lisa;
    int *bart[3];
    lisa = &homer;
    bart[0] = lisa;
    *(bart + 2) = new int[2];
    Test(lisa, &bart[1]);
}

void Test(int *marge, int **maggie)
{
    *maggie++ = marge;
    **maggie = 1;
}
```

430430	1	
	...	
12ff7c		<b>homer</b> (int )
12ff78	① 0012ff7c	<b>lisa</b> (int *)
12ff74	③00430430	<b>bart[2]</b> (int *)
12ff70	⑥ 0012ff7c	<b>bart[1]</b> (int *)
12ff6c	② 0012ff7c	<b>bart[0]</b> (int *)
	...	
12ff1c	⑦ 0012ff74	<b>maggie</b> (int **)
12ff18	④ 0012ff7c	<b>marge</b> (int *) <sup>71</sup>

# 第六章 指针

- 
- ① 基本概念
  - ② 指针变量
  - ③ 指针与数组
  - ④ 引用类型



- “引用”（reference）的作用是为一个变量起一个别名。

```
int  LiBai;
```

```
int  &LiTaiBai = LiBai; //必须初始化
```

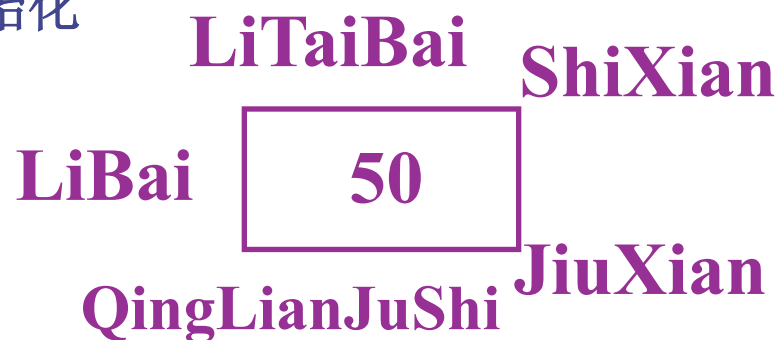
```
int  &QingLianJuShi = LiBai;
```

```
int  &ShiXian = LiBai;
```

```
int  &JiuXian = LiBai;
```

```
LiBai = 49;
```

```
LiTaiBai ++;
```



# 引用作为函数参数

- 使得在被调用的函数中可以修改作为实参的变量的值。

```
void main( )
{
    int salary, nCars, nHouses;

    salary = 6000;
    nCars = 0;
    nHouses = 0;
    DayDreaming(salary, nCars, nHouses);
    cout << salary << " " << nCars << " " << nHouses;
}

void DayDreaming(int &salary, int &cars, int &houses)
{
    salary = salary * 3;
    cars += 2;
    houses ++;
}
```

# 引用即特殊的指针

- 无需取地址&、自动间接访问\*、其指向的对象不能变更。当别名使用，有时易困惑。

```
void main( ) // 假想的实现过程
{
    int salary, nCars, nHouses;

    salary = 6000;
    nCars = 0;
    nHouses = 0;
    DayDreaming(salary, nCars, nHouses);
    cout << salary << " " << nCars << " " << nHouses;
}

void DayDreaming(int *salary, int *cars, int *houses)
{
    *salary = (*salary) * 3;
    *cars += 2;
    (*houses) ++;
}
```

# 本讲小结

- ◆ 指针
- ◆ 指针与数组的关系
- ◆ 动态内存管理
- ◆ 引用