



# 第5章 指针

**5.1 寻找保险箱密码**

**5.2 角色互换**

**5.3 冒泡排序**

**5.4 电码加密**

# 本章要点

- 如何定义指针变量，怎样才能使用指针变量？
- 什么是指针变量的初始化？
- 指针变量的基本运算有哪些？如何使用指针操作所指向的变量？
- 如何使用指针实现函数调用返回多个值？

## 5.1 寻找保险箱密码

一个关于国安局特工寻找保险箱密码的故事...

### 关键点分析

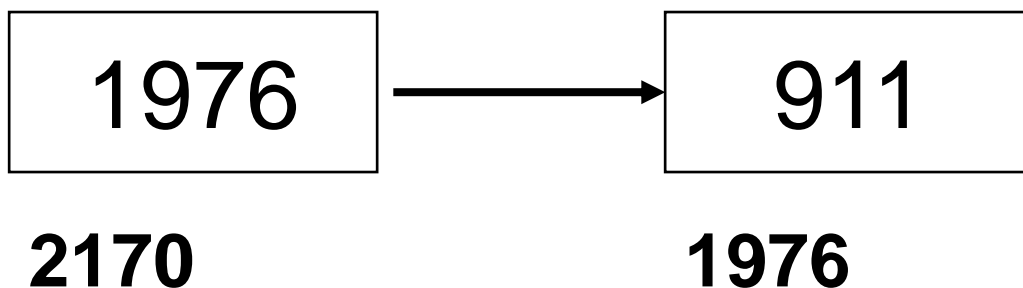
- 得到线索：地址为2170的房间内有线索
- 提示地址：1976
- 找到目标：地址为1976的房间
- 取出内容：911

## 5.1 寻找密码的途径分析

- 密码存放需要一定的存储空间作为存放地，每个存放地都会有地址
- 如果知道了存放地的名字，当然能够找到密码。但并不是每个存储空间都有名字
- 如果不知道存放地的名字，知道该存放地的地址也能够取出密码
- 如果连存放地的地址也不知道，但是有另外一个地方存放了该密码存放地的地址，那么找到这个地方，就能顺藤摸瓜，间接找到密码

## 5.1 密码存放示意图

**P**



名字	P	
地址	2170	1976
内容	1976	911

# 例5-1 利用指针模拟寻找保险箱密码的过程

获取密码的两种方法

int main(void)

```
{ int key = 911;    /* 变量key用于存放密码值911 */
```

```
    int *p_addr = NULL; /* 变量p_addr是整型指针变量 */
```

```
    p_addr = &key;      /* 将key的地址赋给p_addr */
```

```
                        /* 通过变量key输出密码值*/
```

```
    printf("The key is: %d\n", key);
```

```
                        /* 通过变量名key输出密码值*/
```

```
    printf("If I know the name of the variable, I can get it's value by name: %d\n ", key);
```

```
                        /* 通过变量key的地址来输出密码值 */
```

```
    printf("If I know the address of the variable is: %x, then I also can get it's value by address: %d\n",p_addr, *p_addr);
```

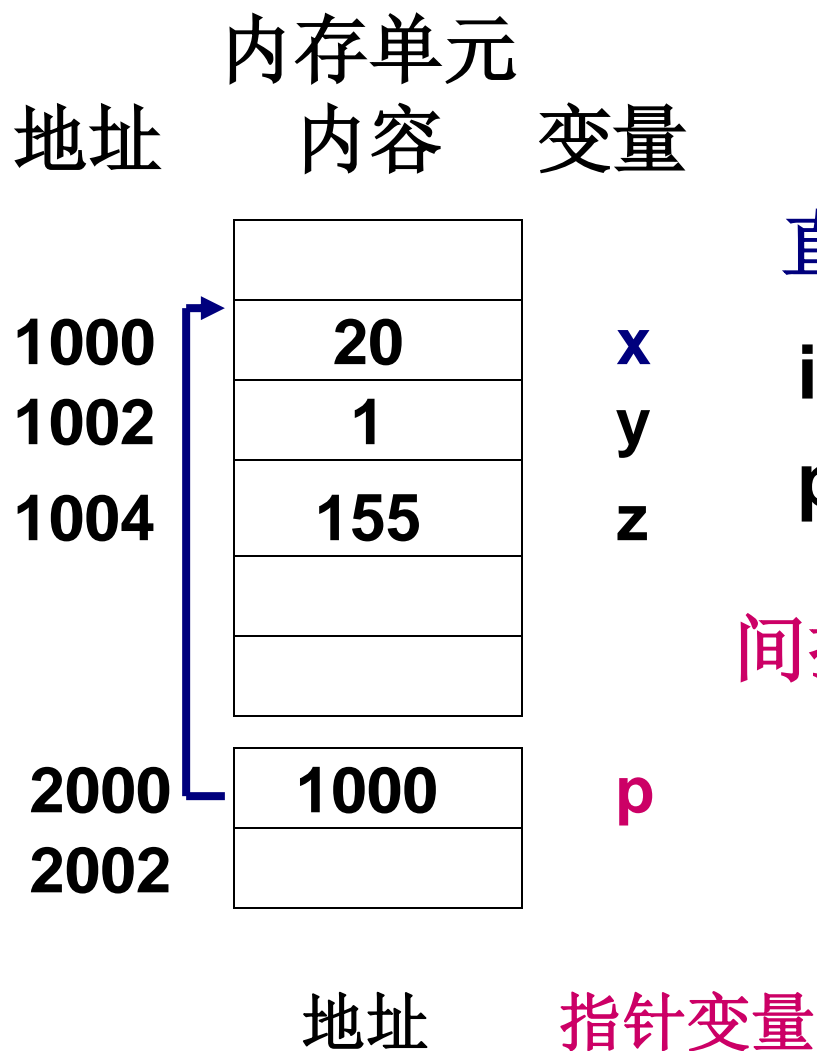
```
    return 0;
```

```
}
```

If I know the name of the variable, I can get it's value by name: **911**

If I know the address of the variable is:**12ff7c**, then I also can get it's value by address: **911**

## 5.1.2 地址和指针—指针的概念



**直接访问：**通过变量名访问

```
int x = 20, y = 1, z = 155;  
printf("%d", x);
```

**间接访问：**通过另一个变量访问  
把变量的地址放到另一变量中  
使用时先找到后者  
再从中取出前者的地址

# 指针

内存单元  
地址 内容 变量

1000	20
1002	1
1004	155
2000	1000
2002	

地址

变量

指针变量

```
int x = 20, y = 1, z = 155;  
printf("%d", x);
```

x

y

z

p

某个变量的地址

指向

指针变量：存放地址的变量



## 5.1.3 指针变量的定义

类型名 \* 指针变量名



指针声明符

指针变量所指向的变量的类型

```
int *p;
```

p 是整型指针，指向整型变量

```
float *fp;
```

fp 是浮点型指针，指向浮点型变量

```
char *cp;
```

cp 是字符型指针，指向字符型变量

# 指针变量的定义

类型名 \* 指针变量名

**int \* p;**

- 指针变量名是 **p**，不是 **\*p**
- **\*** 是指针声明符

**int k, \*p1, \*p2;**

等价于：

**int k;**

**int \*p1;**

**int \*p2;**

- 定义多个指针变量时，每一个指针变量前面都必须加上\*

## 5.1.4 指针的基本运算

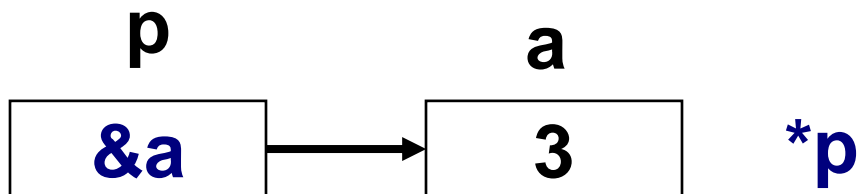
如果指针的值是某个变量的地址，通过指针就能**间接访问**那个变量。

### 1、取地址运算和间接访问运算

**&** 取地址运算符，给出变量的地址

`int *p, a = 3;` **指针变量的类型和它所指向变量的类型相同**

`p = &a;` 把 `a` 的地址赋给 `p`，即 `p` 指向 `a`



**\*** 间接访问运算符，访问指针所指向的变量

**\*p**: 指针变量 `p` 所指向的变量

# 例5-2指针取地址运算和间接访问运算

```
# include <stdio.h>
```

```
int main (void)
```

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

```
  p = &a;
```

```
  printf ("a=%d, *p=%d\n", a, *p);
```

```
  *p = 10;
```

```
  printf("a=%d, *p=%d\n", a, *p);
```

```
  printf("Enter a: ");
```

```
  scanf("%d", &a);
```

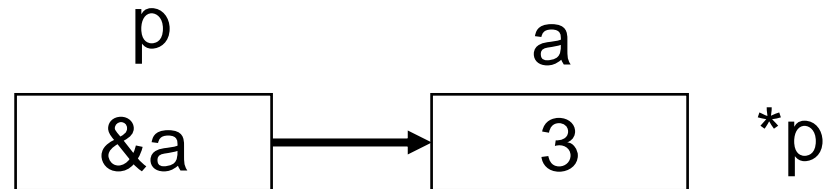
```
  printf("a=%d, *p=%d\n", a, *p);
```

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

```
  printf("a=%d, *p=%d\n", a, *p);
```

```
  return 0;
```

```
}
```



a = 3, \*p = 3

a = 10, \*p = 10

Enter a: 5

a = 5, \*p = 5

a = 6, \*p = 6

# 说明

(1) 当  $p = \&a$  后,  $*p$  与  $a$  相同

(2)  $\text{int } *p$ ; 定义指针变量  $p$

$*p = 10$ ; 指针  $p$  所指向的变量, 即  $a$

(3)  $\&*p$  与  $\&a$  相同, 是地址

$*\&a$  与  $a$  相同, 是变量

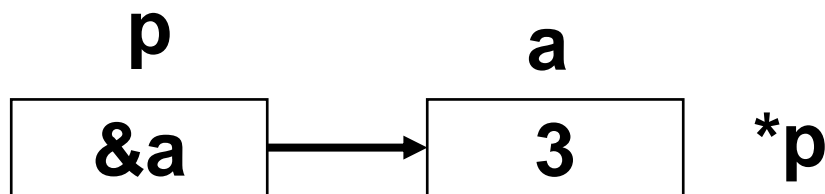
(4)  $(*p)++$  等价于  $a++$

将  $p$  所指向的变量值加1

$*p++$  等价于  $*(p++)$

先取  $*p$ , 然后  $p$  自加, 此时  $p$  不再指向  $a$

```
int a = 1, x, *p;  
p = &a;  
x = *p++;
```



## 2、赋值运算

```
int a = 3, *p1, *p2;
```

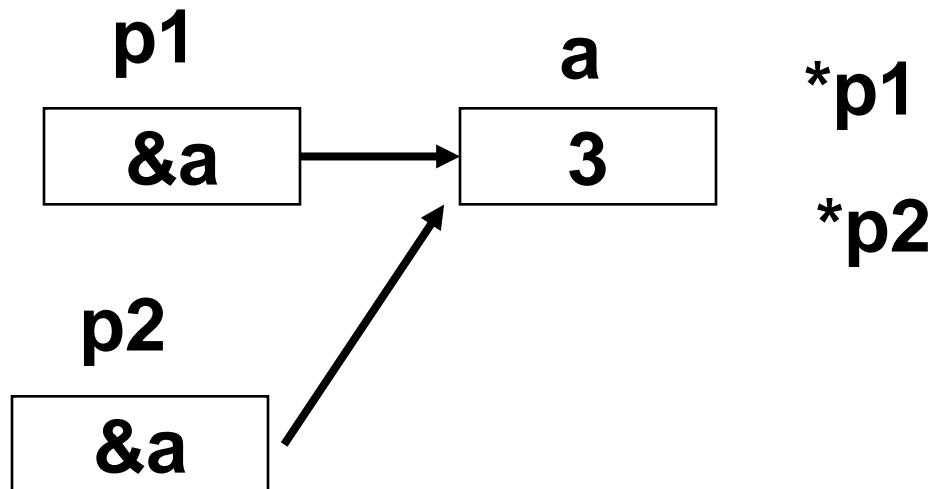
```
p1 = &a;
```

把 a 的地址赋给 p1，即 p1 指向 a

```
p2 = p1;
```

p2 也指向 a

**相同类型的指针才能相互赋值**



## 5.1.5 指针变量的初始化

- 1) 指针变量在定义后也要先赋值再引用
- 2) 在定义指针变量时，可以同时对它赋初值

```
int a;  
int *p1 = &a;  
int *p2 = p1;
```

- 3) 不能用数值作为指针变量的初值，但可以将一个指针变量初始化为一个空指针

```
int *p=1000;  
p = 0;  
p = NULL;  
p = (int*)1732;
```

使用强制类型转换 (**int\***) 来避免编译错误，不提倡

## 5.2 角色互换

如何通过函数调用实现代表2个角色的变量互相...

### 三套方案

- `swap1()`
- `swap2()`
- `swap3()`

哪个方案能成功？



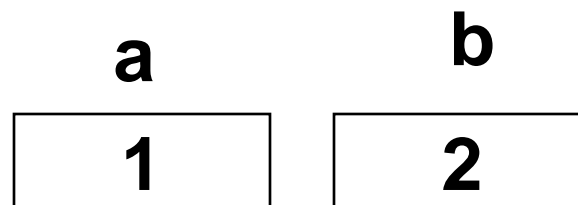
## 例5-3 指针作为函数参数模拟角色互换

**调用哪个函数，可以交换main ()  
中变量a和b的值？**

```
int main (void)
{   int a = 1, b = 2;
    int *pa = &a, *pb = &b;
    void swap1(int x, int y), swap2( int *px, int *py ), swap3 (int *px, int *py);
    swap1 (a, b);
    printf ("After calling swap1: a=%d b=%d\n", a, b);
    a = 1;  b = 2;
    swap2(pa, pb);
    printf ("After calling swap2: a=%d b=%d\n", a, b);
    a = 1;  b = 2;
    swap3(pa, pb);
    printf ("After calling swap3: a=%d b=%d\n", a, b);
    return 0;
}
```

## 例5-3 swap1()

**swap1 (a, b);**



**void swap1 (int x, int y)**

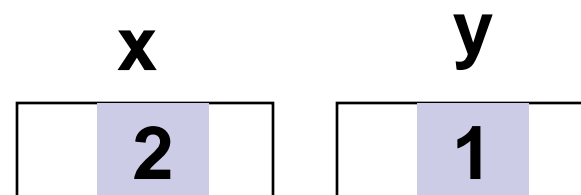
**{ int t;**

**t = x;**

**x = y;**

**y = t;**

**}**



在**swap1()**函数中改变了形参**x,y**的值  
但不会反过来影响到实参的值

**swap1()**不能改变**main()**函数中实参**a**和**b**的值

## 例5-3 swap2()

```
swap2 (&a, &b);
```

```
void swap2 (int *px, int *py)
```

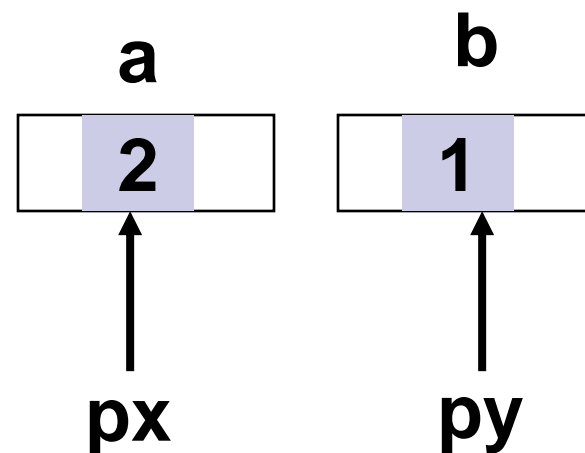
```
{  int t;
```

```
    t = *px;
```

```
    *px = *py;
```

```
    *py = t;
```

```
}
```



在swap2()函数中交换\*px和\*py的值，主调函数中a和b的值也相应交换了

值传递，地址未变，  
但存放的变量的值改变了

## 例5-3 swap3()

```
swap3 (&a, &b);
```

```
void swap3 (int *px, int *py)
```

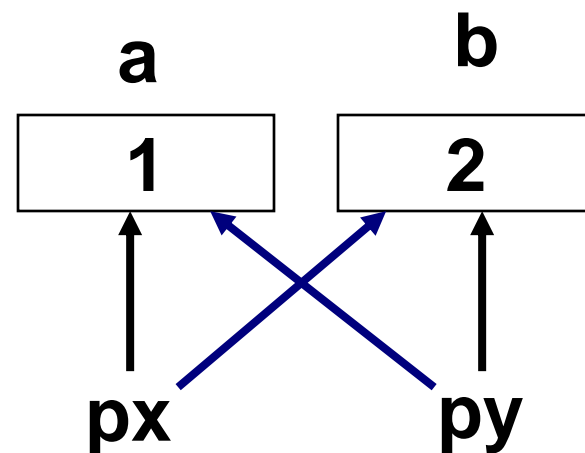
```
{  int *pt;
```

```
    pt = px;
```

```
    px = py;
```

```
    py = pt;
```

```
}
```



**swap3()中直接交换了形参指针px和py的值**

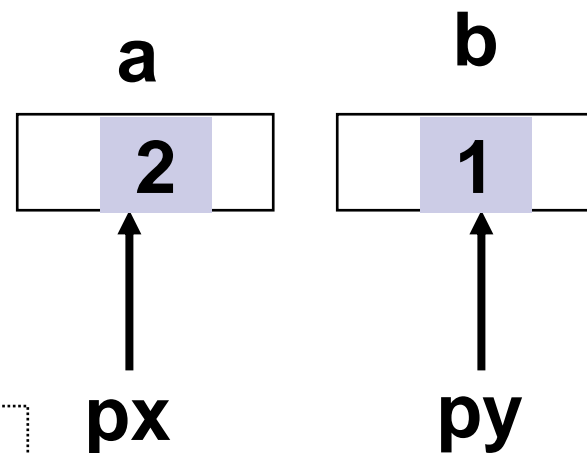
**值传递，形参指针的改变  
不会影响实参**

# swap2 (&a, &b); 指针作为函数参数的应用

```
void swap2 (int *px, int *py)
```

```
{  int t;  
    t = *px;  
    *px = *py;  
    *py = t;  
}
```

After calling swap1: a=1, b=2  
After calling swap2: a=2, b=1  
After calling swap3: a=1, b=2



要通过函数调用来改变主调函数中某个变量的值:

- (1) 在主调函数中, 将该变量的地址或者指向该变量的指针作为实参
- (2) 在被调函数中, 用指针类型形参接受该变量的地址
- (3) 在被调函数中, 改变形参所指向变量的值

## 5.2.2 指针作为函数参数

- 函数参数包括实参和形参，两者的类型要一致，可以是指针类型
- 如果实参是某个变量的地址，相应的形参就是指针
- 在C语言中实参和形参之间的数据传递是单向的“值传递”方式

## 5.3.2 数组和地址间的关系

```
int a[100], *p;
```

数组名代表一个地址，它的值是数组首元素的地址（基地址）

**a+i** 是距数组a的基地址的第i个偏移

	地址	内容	数组元素
<b>a</b>	3000		<b>a[0]</b>
<b>a+1</b>	3002		<b>a[1]</b>
<b>a+i</b>			<b>a[i]</b>
<b>a+99</b>	3198		<b>a[99]</b>

**&a[i]** **\*(a+i)**

```
sum = 0;
```

```
for(i = 0; i < 100; i++)
```

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

下标运算符[ ]的含义

# 指针和数组的关系

任何由数组下标来实现的操作都能用指针来完成

```
int a[100], *p;
```

```
p = a;
```

或

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

**p**

**a**

**p+1**

**a+1**

**p+i**

**a+i**

**p+99**

**a+99**

地址

内容

数组元素

3000

3002

3198


**a[0]**

**a[1]**

**a[i]**

**a[99]**

```
p = a;
```

```
sum = 0;
```

```
for(i = 0; i < 100; i++)
```

```
    sum = sum + p[i];
```

**&a[i]**

**a+i**

**p+i**

**&p[i]**

**等价**

**等价**

**a[i]**

**\*(a+i)**

**\*(p+i)**

**p[i]**



# 用指针完成对数组的操作

```
int a[100], *p;
```

移动指针

		地址	内容	数组元素
<b>p</b>	<b>a</b>	<b>3000</b>		<b>a[0]</b>
<b>p</b>	<b>a+1</b>	<b>3002</b>		<b>a[1]</b>
<b>p</b>	<b>a+i</b>			<b>a[i]</b>
<b>p</b>	<b>a+99</b>	<b>3198</b>		<b>a[99]</b>

```
sum = 0;
```

```
for(p = a; p <= &a[99]; p++)
```

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

## 例8-6 使用指针计算数组元素个数和数组元素的存储单元数

```
# include <stdio.h>
```

```
int main (void)
```

```
{  double a[2], *p, *q;
```

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

```
    q = p + 1;
```

```
    printf ("%d\n", q - p);
```

```
    printf ("%d\n", (int) q - (int) p);
```

```
    return 0;
```

```
}
```

		地址	内容	数组元素
<b>p</b>	<b>a</b>	3000		a[0]
<b>q</b>	<b>a+1</b>	3008		a[1]

1

8

指针p和q之间元素的个数

指针p和q之间的字节数

地址值

# 指针的算术运算和比较运算

			地址	内容	数组元素
double *p, *q;	p	a	3000		a[0]
	q	a+1	3008		a[1]

## ■ $q - p$

两个相同类型的指针相减，表示它们之间相隔的存储单元的数目

## ■ $p + 1 / p - 1$

指向下一个存储单元 / 指向上一个存储单元

## ■ 其他操作都是非法的

指针相加、相乘和相除，或指针加上和减去一个浮点数

## ■ $p < q$

两个相同类型指针可以用关系运算符比较大小

## 例5-7 分别使用数组和指针计算数组元素之和

			地址	内容	数组元素
<pre>int main(void) {  int i, a[10], *p;   long sum = 0;   printf("Enter 10 integers: ");   for(i = 0; i &lt; 10; i++)     scanf("%d", &amp;a[i]);   for ( i = 0; i &lt; 10; i++)     sum = sum + a[i];   printf("calculated by array,     sum=%ld \n", sum);    sum=0;   for(p = a; p &lt;= a+9; p++)     sum = sum + *p;   printf("calculated by pointer,     sum=%ld \n", sum);   return 0; }</pre>	<b>p</b>	<b>a</b>	<b>3000</b>		<b>a[0]</b>
	<b>p</b>	<b>a+1</b>	<b>3002</b>		<b>a[1]</b>
	<b>p</b>	<b>a+i</b>			<b>a[i]</b>
	<b>p</b>	<b>a+9</b>	<b>3018</b>		<b>a[9]</b>

Enter 10 integers: **10 9 8 7 6 5 4**  
**3 2 1**

calculated by array, sum=**55**

calculated by pointer, sum=**55**

写出程序运行结果:

```
#include <stdio.h>
```

```
int main( void)
```

```
{ int a[]={2,4,6,8,10};
```

```
    int s=0, i, *p;
```

```
    p=&a[1];
```

```
    for (i=1; i<4; i++ )
```

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

```
    printf("%d\n",s);
```

```
    return 0;
```

```
}
```

## 5.3.3 数组名作为函数的参数

数组元素作为函数实参时，函数形参为变量  
与变量作为函数实参相同，值传递

```
double fact (int n);
int main(void )
{
    int a[5]={1, 4, 5, 7, 9};
    int i, n = 5;
    double sum;
    sum = 0;
    for(i = 1; i <= n; i++ )
        sum = sum + fact(a[i-1]);
    printf("sum = %e\n", sum);
    return 0;
}
```

```
double fact (int n)
{
    int i;
    double result = 1;
    for (i = 1; i <= n; i++)
        result = result * i ;
    return result ;
}
```

**1!+4!+5!+7!+9!**

# 数组名作为函数的参数

- 数组名是指针常量，相当于指针作为函数的参数
- 数组名做为实参，形参是指针变量（数组）

(1) 实参是数组名

(2) 形参是指针变量

可以写成数组形式

int **a[ ]**

```
int sum (int *a, int n)
{  int i, s = 0;
   for(i=0; i<n; i++)
       s += a[i];
       *(a+i)
   return(s);
}
```

例

```
int main(void )
{  int i;
   int b[5] = {1, 4, 5, 7, 9};
   printf("%d\n", sum(b, 5));
   return 0; }
```

```

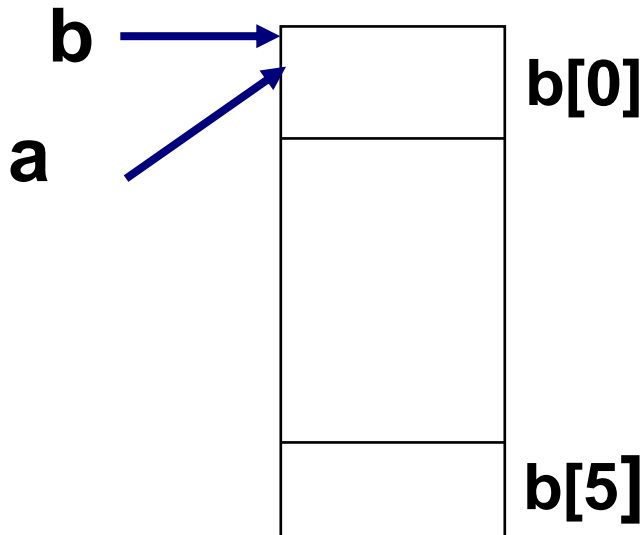
int sum (int *a, int n)
{
    int i, s = 0;
    for(i=0; i<n; i++)
        s += a[i];
    return(s);
}

```

```

int main(void )
{
    int i;
    int b[5] = {1, 4, 5, 7, 9};
    printf("%d\n", sum(b, 5));
    return 0;
}

```



`sum(b, 5)`      `b[0]+b[1]+...+b[4]`

`sum(b, 3)`      `b[0]+b[1]+b[2]`

`sum(b+1, 3)`      `b[1]+b[2]+b[3]`

`sum(&b[2], 3)`      `b[2]+b[3]+b[4]`



## 例5-8 将数组元素逆序存放

```
#include <stdio.h>
```

```
int main(void)
```

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

```
    void reverse(int p[ ], int n);
```

```
    printf("Enter n: ");
```

```
    scanf("%d", &n);
```

```
    printf("Enter %d integers: ", n);
```

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

```
        scanf("%d", &a[i]);
```

```
    reverse(a, n);
```

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

```
        printf("%d\t", a[i]);
```

```
    return 0;
```

```
}
```

Enter n: 10

Enter 10 integers: 10 9 8 7 6 5 4 3 2 1

1 2 3 4 5 6 7 8 9 10

```
void reverse(int p[ ], int n)
```

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

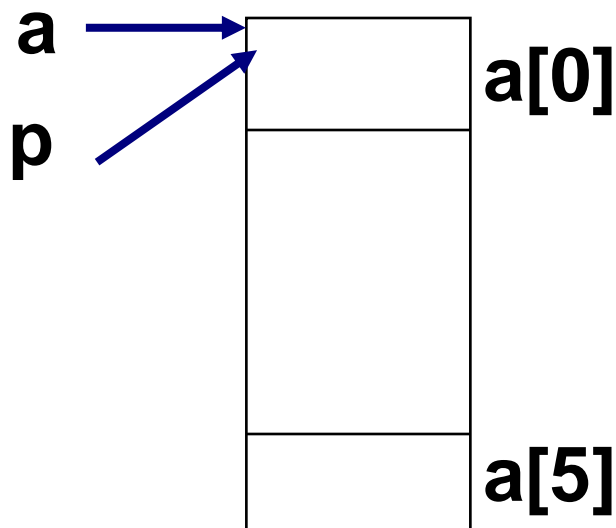
```
    for(i=0, j=n-1; i<j; i++, j--){
```

```
        t = p[i]; p[i] = p[j]; p[j] = t;
```

```
    }
```

```
}
```

- 数组名做为函数的参数，在函数调用时，将实参数组首元素的地址传给形参（指针变量），因此，形参也指向实参数组的首元素。如果改变形参所指向单元的值，就是改变实参数组首元素的值。
- 或：形参数组和实参数组共用同一段存贮空间，如果形参数组中元素的值发生变化，实参数组中元素的值也同时发生变化。



# 指针实现

```
#include <stdio.h>
int main(void)
{ int i, a[10], n;
  void reverse(int p[ ], int n);
```

```
  printf("Enter n: ");
  scanf("%d", &n);
  printf("Enter %d integers: ", n);
  for(i = 0; i < n; i++)
    scanf("%d", &a[i]);
  reverse(a, n);
  for(i = 0; i < n; i++)
    printf("%d\t", a[i]);
  return 0;
}
```



```
void reverse(int *p, int n)
{ int *pj, t;
  for(pj=p+n-1; p<pj; p++, pj--){
    t=*p; *p=*pj; *pj=t;
  }
}
```

## 5.3.4 冒泡排序算法分析

相邻两个数比较,小的调到前面,大的调到后面

9	8	8	8	8	8	5	4	4	0
8	9	5	5	5	5	4	5	0	4
5	5	9	4	4	4	6	0	5	
4	4	4	9	6	6	0	6		
6	6	6	6	9	0	8			
0	0	0	0	0	9				

9	8	8	8	8	8
8	9	5	5	5	5
5	5	9	4	4	4
4	4	4	9	6	6
6	6	6	6	9	0
0	0	0	0	0	9

5	4	4	0
---	---	---	---

4	5	0	4
---	---	---	---

6	0	5
---	---	---

0	6
---	---

8
---

$i=5$

$i=4$

$i=3$   $j=0$  to  $2$

$i=2$   $j=0$  to  $3$

$i=1$   $j=0$  to  $4$

$j=0$  to  $6-1-i$

$a[j]>a[j+1]$

```
int main(void )
{  int i, j, n, t, a[10];
    n = 6;
    for(i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(i = 1; i < n; i++)
        for(j = 0; j < n-i; j++)
            if(a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
    return 0;
}
```

**9 8 5 4 6 0**

**i=1**

**j=0: 8 9 5 4 6 0**

**j=1: 8 5 9 4 6 0**

**j=2: 8 5 4 9 6 0**

**j=3: 8 5 4 6 9 0**

**j=4: 8 5 4 6 0 9**

```
void sort(int *array, int n)
{
    int i, j, t;
    for(i=1; i<n; i++)
        for(j=0; j<n-i; j++)
            if(array[j]>array[j+1]){
                t = array[j];
                array[j] = array[j+1];
                array[j+1] = t;
            }
}
```

```
int main(void )
{ int i, a[10];

    for(i=0; i<10; i++)
        scanf("%d", &a[i]);
    sort(a, 10);
    for(i=0; i<10; i++)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

## 5.4 电码加密

字符串： 字符数组  
          字符指针

**5.4.1 程序解析**

**5.4.2 字符串和字符指针**

**5.4.3 常用的字符串处理函数**



## 5.4.1 程序解析—加密

```
# define MAXLINE 100
void encrypt(char *);
int main (void)
{ char line [MAXLINE];
```

```
    printf ("Input the string: ");
    gets(line);
    encrypt (line);
```

```
    printf ("%s%s\n", "After being encrypted: ", line);
```

```
    return 0;
}
```

```
void encrypt ( char *s)
{
    for ( ; *s != '\0'; s++)
        if (*s == 'z')
            *s = 'a';
        else
            *s = *s+1;
}
```

Input the string: **hello hangzhou**  
After being encrypted: **ifmmp!ibohaipv**

## 5.4.2 字符串和字符指针

### ■ 字符串常量


**"array"**

**"point"**

- 用一对双引号括起来的字符序列
- 被看做一个特殊的一维字符数组,在内存中连续存放
- 实质上是一个指向该字符串首字符的指针常量

**char sa[ ] = "array";**

**char \*sp = "point";**



```
char sa[ ] = "array";
```

```
char *sp = "point";
```

```
printf("%s ", sa);
```

```
printf("%s ", sp);
```

```
printf("%s\n", "string");
```

```
printf("%s ", sa+2);
```

```
printf("%s ", sp+3);
```

```
printf("%s\n", "string"+1);
```

array point string

ray nt tring

数组名**sa**、指针**sp**和字符串 **"string"** 的值都是  
地址

# 字符数组与字符指针的重要区别

```
char sa[ ] = "This is a string";
```


```
char *sp = "This is a string";
```

sa

T	h	i	s		i	s		a		s	t	r	i	n	g	\0
---	---	---	---	--	---	---	--	---	--	---	---	---	---	---	---	----

sp

--



T	h	i	s		i	s		a		s	t	r	i	n	g	\0
---	---	---	---	--	---	---	--	---	--	---	---	---	---	---	---	----

如果要改变数组**sa**所代表的字符串，只能改变数组元素的内容

如果要改变指针**sp**所代表的字符串，通常直接改变指针的值，让它指向新的字符串

## 示例

```
char sa[ ] = "This is a string";
```

```
char *sp = "This is a string";
```

```
strcpy (sa, "Hello");
```

```
sp = "Hello";
```

```
sa = "Hello"; 非法
```

数组名是常量，不能对它赋值

# 字符指针—先赋值，后引用

定义字符指针后，如果没有对它赋值，指针的值**不确定**。

```
char *s ;  
scanf("%s", s);
```

**不要引用未赋值的指针**

```
char *s, str[20];  
s = str;  
scanf("%s", s);
```

定义指针时，先将它的初值置为空

```
char *s = NULL
```

# 加密函数的两种实现

```
void encrypt ( char *s)
{
    for ( ; *s != '\0'; s++)
        if (*s == 'z')
            *s = 'a';
        else
            *s = *s+1;
}
```

```
void encrypt (char s[ ])
{
    int i;
    for(i = 0; s[i] != '\0'; i++)
        if (s[i] == 'z')
            s[i] = 'a';
        else
            s[i] = s[i]+1;
}
```

## 5.4.3 常用的字符串处理函数

- 函数原型在 **stdio.h** 或 **string.h** 中给出

### 1、字符串的输入和输出

- 输入字符串: **scanf( )**或**gets( )**
- 输出字符串: **printf( )**或**puts( )**
- **stdio.h**



# 字符串的输入

```
char str[80];  
i = 0;  
while((str[i] = getchar( )) != '\n')  
    i++;  
str[i] = '\0';
```

'\n'  
' '  
'\t'

## (1) scanf("%s", str)

输入参数：字符数组名,不加地址符

遇回车或空格输入结束，并自动将输入的一串字符和 **'\0'** 送入数组中

## (2) gets(str)

遇回车输入结束，自动将输入的一串字符和 **'\0'** 送入数组中

# 字符串的输出

```
char str[80];  
for(i = 0; str[i] != '\0'; i++)  
    putchar(str[i]);
```

```
(3) printf("%s", str)  
    printf("%s", "hello");
```

```
(4) puts(str)  
    puts("hello");  
    输出字符串后自动换行
```

输出参数可以是字符数组名或字符串常量，输出遇 **'\0'** 结束

# 例5-10 字符串输入输出函数示例

```
#include <stdio.h>
int main( )
{ char str[80];
  scanf("%s", str);
  printf("%s", str);
  printf("%s", "Hello");
  return 0;
}
```

**Programming**

**ProgrammingHello**

**Programming is fun!**

**ProgrammingHello**

```
#include <stdio.h>
int main( )
{ char str[80];
  gets(str);
  puts(str);
  puts("Hello");
  return 0;
}
```

**Programming**

**Programming**

**Hello**

**Programming is fun!**

**Programming is fun!**

**Hello**

## 2、字符串的复制、连接、比较、求字符串长度

- 字符串复制: **strcpy(str1, str2)**
- 字符串连接: **strcat(str1, str2)**
- 字符串比较: **strcmp(str1, str2)**
- 求字符串长度: **strlen(str)**
- string.h**

# 字符串复制函数strcpy()

**strcpy(str1, str2);**

将字符串 **str2** 复制到 **str1** 中

**char str1[20];**

**char str2[20] = "happy";**

\0					
----	--	--	--	--	--

h	a	p	p	y	\0
---	---	---	---	---	----

**strcpy(str1, str2);**

**str1 中**

h	a	p	p	y	\0
---	---	---	---	---	----

**strcpy(str1, "world");**

**str1 中:**

w	o	r	l	d	\0
---	---	---	---	---	----

# strcpy() 示例

```
# include "stdio.h"
# include "string.h"
int main(void )
{
    char str1[20], str2[20];
    gets(str2);
    strcpy(str1,str2);
    puts(str1);
    return 0;
}
```

1234

1234

# 字符串连接函数strcat

**strcat(str1, str2);**

连接两个字符串**str1**和**str2**, 并将结果放入**str1**中

```
# include "stdio.h"
# include "string.h"
int main(void)
{
    char str1[80], str2[20];
    gets(str1);
    gets(str2);
    strcat(str1, str2);
    puts(str1);
    return 0;
}
```

str1中: Let us \0

str2中: go.\0

str1中: Let us go.\0

str2中: go.\0

Let us  
go.  
Let us go.

**str1=str1+str2 非法!**

# 字符串比较函数**strcmp**

**strcmp(str1, str2)**

比较 两个字符串**str1**和**str2**的大小。

规则：按字典序(**ASCII**码序)

- 如果 **str1** 和 **str2** 相等，返回 **0**;
- 如果 **str1** 大于 **str2** ， 返回一个正整数;
- 如果 **str1** 小于 **str2** ， 返回一个负整数;

```
static char s1[20] = "sea";
```

```
strcmp(s1, "Sea");
```

```
strcmp("Sea", "Sea ");
```

```
strcmp("Sea", "Sea");
```

正整数

负整数

0



# strcmp() 示例

```
# include "stdio.h"
# include "string.h"
int main(void )
{ int res;
  char s1[20], s2[20];
  gets(s1);
  gets(s2);
  res = strcmp(s1, s2);
  printf("%d", res);
  return 0;
}
```

1234

2

-1

# 用strcmp()比较字符串

利用字符串比较函数比较字符串的大小

**strcmp(str1, str2);**

为什么定义这样的函数？

**str1 > str2**

**str1 < "hello"**

**str1 == str2**

**strcmp(str1, str2) > 0**

**strcmp(str1, "hello") < 0**

**strcmp(str1, str2) == 0**

**比较字符串首元素的地址**

**比较字符串的内容**

# 字符串长度函数strlen

## ■ strlen(str)

计算字符串的有效长度，不包括 ‘\0’ 。

**static char str[20]="How are you?"**

**strlen ("hello") 的值是: 5**

**strlen(str) 的值是: 12**

# 字符串处理函数小结

函数	功能	头文件
<b>puts(str)</b>	输出字符串	<b>stdio.h</b>
<b>gets(str)</b>	输入字符串（回车间隔）	
<b>strcpy(s1,s2)</b>	<b>s2 ==&gt; s1</b>	<b>string.h</b>
<b>strcat(s1,s2)</b>	<b>s1 “+” s2 ==&gt; s1</b>	
<b>strcmp(s1,s2)</b>	若 s1“ <b>==</b> ”s2, 函数值为 <b>0</b> 若 s1 “ <b>&gt;</b> ” s2, 函数值 <b>&gt;0</b> 若 s1 “ <b>&lt;</b> ” s2, 函数值 <b>&lt;0</b>	
<b>strlen(str)</b>	计算字符串的有效长度, 不包括 ‘\0’	

## 例5-11 求最小字符串

```
int main( )
{ int i;
  int x, min;
  scanf("%d", &x);
  min = x;
  for(i = 1; i < 5; i++){
    scanf("%d", &x);
    if(x < min)
      min = x;
  }
  printf("min is %d\n", min);
  return 0;
}
```

2 8 -1 99 0

min is -1

```
#include <string.h>
```

```
int main( )
{ int i;
  char sx[80], smin[80];
  scanf("%s", sx);
  strcpy(smin,sx);
  for(i = 1; i < 5; i++){
    scanf("%s", sx);
    if(strcmp(sx, smin)<0)
      strcpy(smin,sx);
  }
  printf("min is %s\n", smin);
  return 0;
}
```

tool key about zoo sea  
min is about