# 第7章 指针

# 本章内容导读

- 本章主要介绍指针和指针变量，以及使用指针变量引用变量、字符串、一维数组元素的方法。同时介绍指针数组的定义和指针数组元素的使用。通过学习本章，读者应掌握以下内容：
  - 掌握地址、指针和指针变量的概念；
  - 掌握指针变量指向数组元素、数组或字符串的方法；
  - 掌握指针变量引用一维数组元素、二维数组元素或字符串中字符的方法；
  - 了解指针数组的概念和使用方法。

## 7 量

### 1

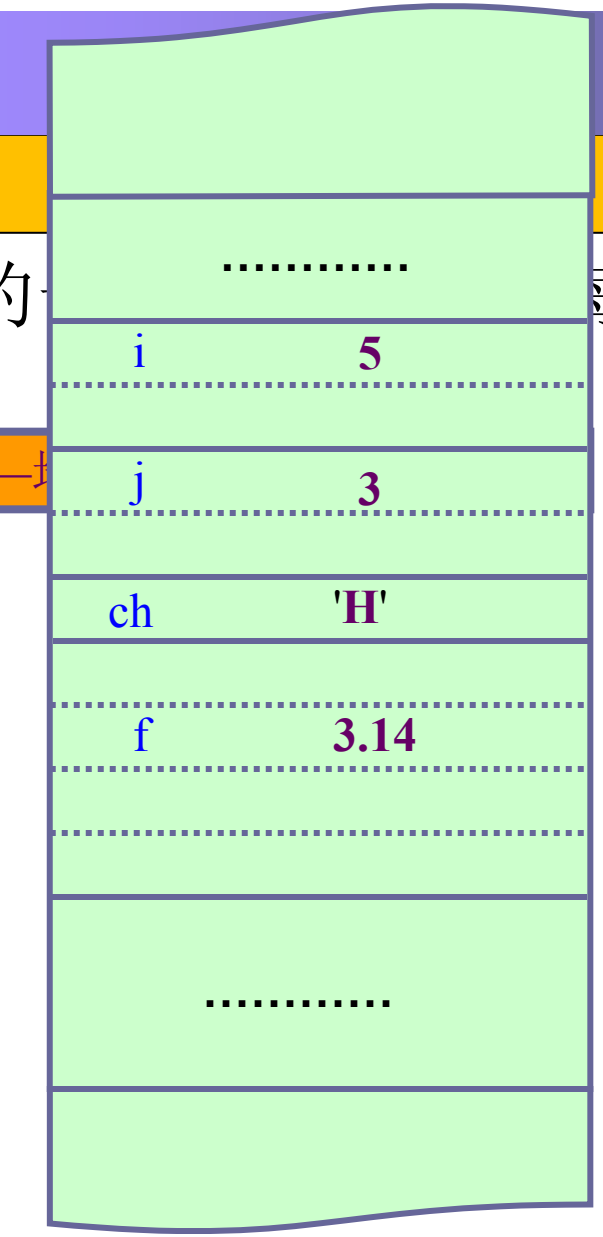
```
int i,j; char ch; float f;
i=5; j=3; ch='H'; f=3.14;
```

地址：一个变量名代表内存中的一个或多个存储单元，每个存储单元都有一个编号，即“地址”。

编译或函数调用时为变量分配内存单元

房间号—内存单元　房客—变量值　房门号—地址

内存中每个字节均有一个编号——地址



# 7.1 指针和指针变量

## 1、指针

例如 int main(void)

```
{   int num;
    scanf("%d",&num);
    printf("num=%d\n", num);
}
```

C编译程序编译到变量定义语句时，将变量num登录到“符号表”中。符号表的关键属性有两个：一是“标识符名（id）”，二是该标识符在内存空间中的“地址（addr）”。

为描述方便，假设系统分配给变量num的2字节存储单元为3000和3001，则起始地址3000就是变量num在内存中的地址。

- **变量值的存取——**通过变量在内存中的地址进行

系统执行scanf("%d",&num);和printf("num=%d\n", num);时，存取变量num值的方式有两种：

# 7.1 指针和指针变量

## 1、指针

- **直接访问——**直接利用变量的地址进行存取

（1）上例中scanf("%d",&num);的执行过程：用变量名num作为索引值，检索符号表，找到变量num的起始地址3000；然后将键盘输入的值（假设为３）送到内存单元3000和3001中。此时，变量num在内存中的地址和值。

（2）printf("num=%d\n",num);的执行过程与scanf相似. 首先找到变量num的起始地址3000，然后从3000和3001中取出其值，最后将它输出。

- **间接访问——**通过另一变量访问该变量的值

# 7.1 指针和指针变量

## 1、指针

例如，假设定义了一个指针变量num_pointer，它被分配到4000、4001单元，其值可通过赋值语句num_pointer=&num；得到。此时，指针变量num_pointer的值就是变量num在内存中的起始地址3000。

通过指针变量num_pointer存取变量num值的过程：首先找到指针变量num_pointer的地址（4000），取出其值3000（正好是变量num 的起始地址）； 然后从3000、3001中取出变量num的值（3）。

- 比较

两种访问方式之间的关系，可以用某人甲（系统）要找某人乙（变量）类比。

一种情况是，甲知道乙在何处，直接去找就是（即直接访问）。

另一种情况是，甲不知道乙在哪，但丙（指针变量）知道，此时甲可以这么做：先找丙，从丙处获得乙的去向，然后再找乙（即间接访问）。

# 7.1 指针和指针变量

## 2、指针变量的定义和初始化

- 定义格式

[存储类型] 数据类型符 *指针变量名[=初始地址值],…;

例
```
int i;
int *p=&i;
```

变量必须已说明过
类型应一致

例
```
int i,*p=&i;
int *p=&i,i;(×)
int *q=p;
```

用已初始化指针变量作初值

# 7.1　指针和指针变量

## 2、指针变量的定义和初始化

➢ **说明**

（1）“*”表示定义一个指针变量。指针变量的前面必须有“*”号。

（2）在定义指针变量的同时也可以定义普通变量或数组等。

（3）“数据类型符”是指针变量所指向变量的数据类型，可以是任何基本数据类型，也可以是其它数据类型。这个数据类型符不是指针变量中存放的数据类型，而是指针变量所指向的变量或数组的数据类型。

（4）“初始地址值”通常是&变量名、&数组元素或一维数组名，这里的变量或数组必须是已经定义过的。

（5）在定义指针变量时，可以只给部分指针变量赋初值。

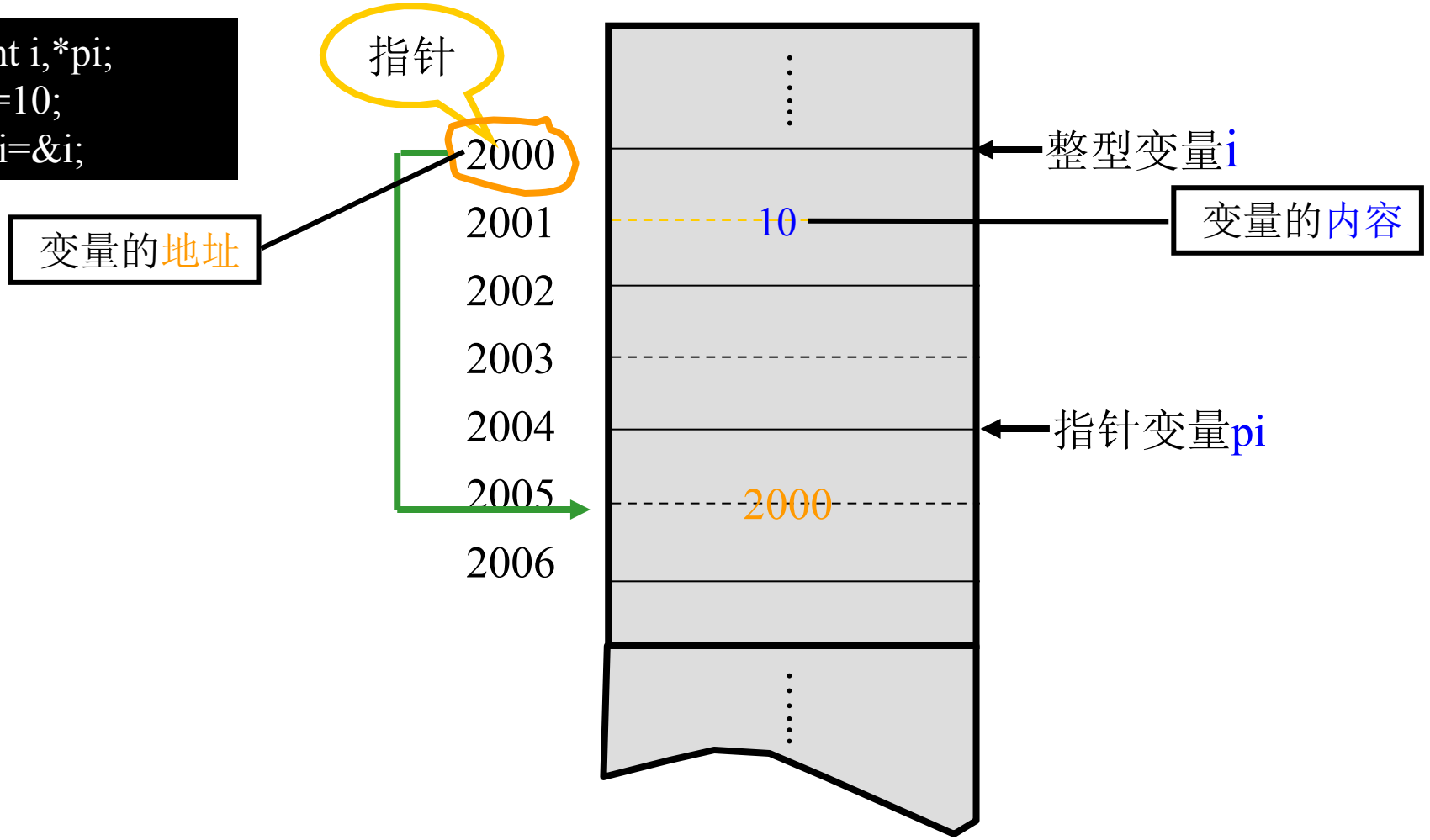
（6）指针变量的初始化，除了可以是已定义变量的地址，也可以是已初始化的同类型的指针变量，也可以是NULL（空指针）。

（7）指针变量初始化时，指针变量的“数据类型符”必须与其“初始地址值”是同一类型。

# 7.1 指针和指针变量

## 2、指针变量的定义和初始化

```
int i,*pi;
i=10;
pi=&i;
```



# 7.1　指针和指针变量

## 2、指针变量的定义和初始化

- **内存地址——**内存中存储单元的编号

（1）计算机系统的内存中，拥有大量的存储单元（容量为1字节）。为了方便管理，必须为每一个存储单元编号，这个编号就是存储单元的“地址”。每个存储单元都有一个惟一的地址。

（2）在地址所标识的存储单元中存放数据。

***注意：****内存单元的地址与内存单元中的数据是两个完全不同的概念。*

- **变量地址——**系统分配给变量的内存单元的起始地址

# 7.1　指针和指针变量

## 3、指针变量的一般使用

- 给指针变量赋值

格式：指针变量=地址型表达式

- 直接使用指针变量

格式：指针变量名

- 通过指针变量引用所指向的变量

格式：*指针变量名

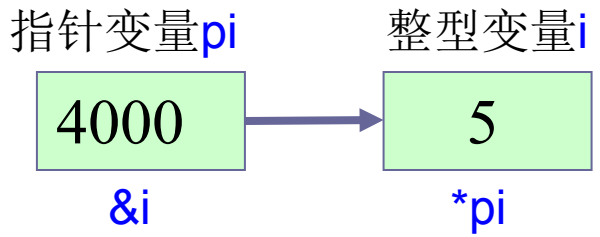
# 7.1 指针和指针变量

## 4、指针的基本运算

- 取地址运算符&：取变量的地址，单目运算符、右结合性。

例：int i=5, j, *pi;

pi=&i;



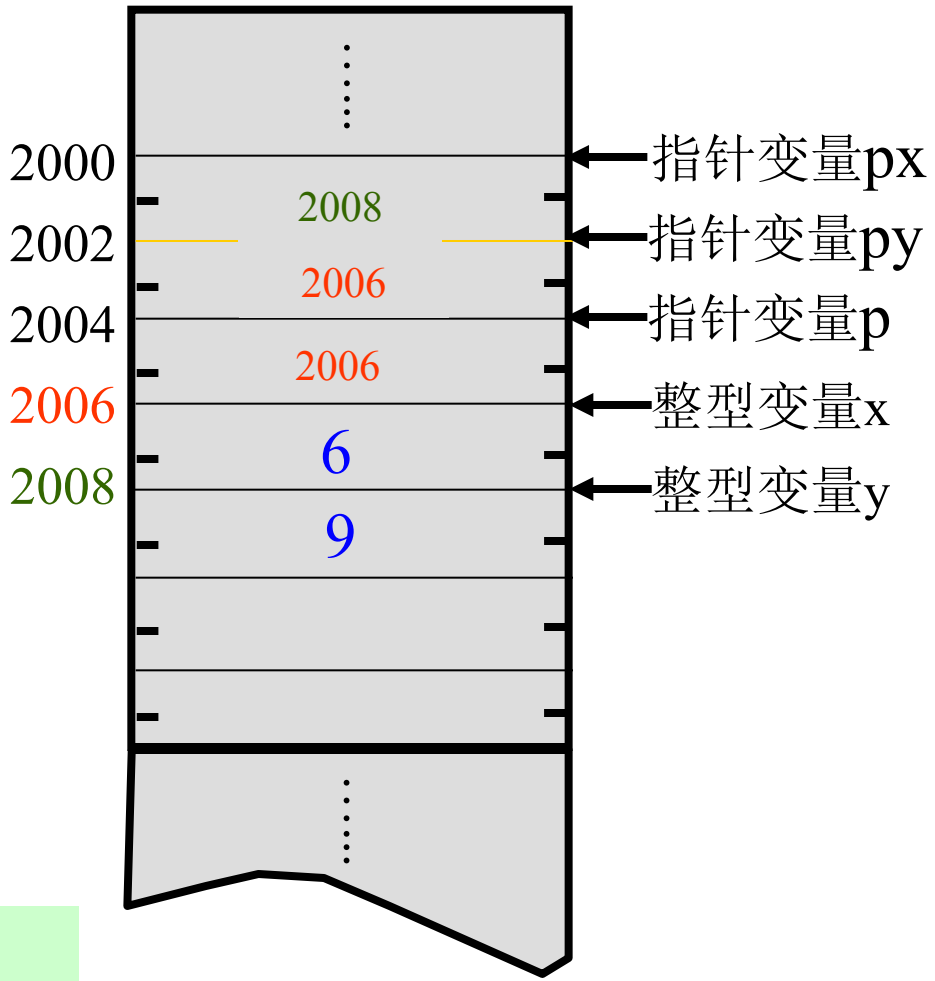

- 指针运算符*：取指针变量所向指地址中的内容，与&为互逆运算。单目运算符、右结合性。

例：j=*pi; 相当于j=i;

pi=&i = &(*pi)
i =*pi = *(&i)

## 【例7-1】输入x、y两个整数，按先大后小的顺序输出x、y。

```
#include <stdio.h>
int main(void)
{
    int x,y,*px,*py,*p;
    scanf("%d%d",&x,&y);
    px=&x;py=&y;
    if(x<y)
   { p=px;
     px=py;
     py=p;
    }
    printf("x=%d,y=%d\n",x,y);
    printf("MAX=%d,MIN=%d\n"
    ,*px,*py);
    return 0;
}
```




运行输入：6 9

输出结果：x=6,y=9
MAX=9,MIN=6

# 7.2 指针和数组

## 1、指针和一维数组

- **指针变量指向一维数组的方式**
  - 在数据定义语句中用赋初值的方式　　*指针变量=数组名;
  - 在程序中用赋值的方式　指针变量=数组名;
- **将指针变量指向一维数组元素的方式**
  - 在数据定义语句中用赋初值的方式 *指针变量=&数组名[下标];
  - 在程序中用赋值的方式　指针变量=&数组名[下标];
- **当指针变量指向一维数组时，引用下标为i的一维数组元素的方法**

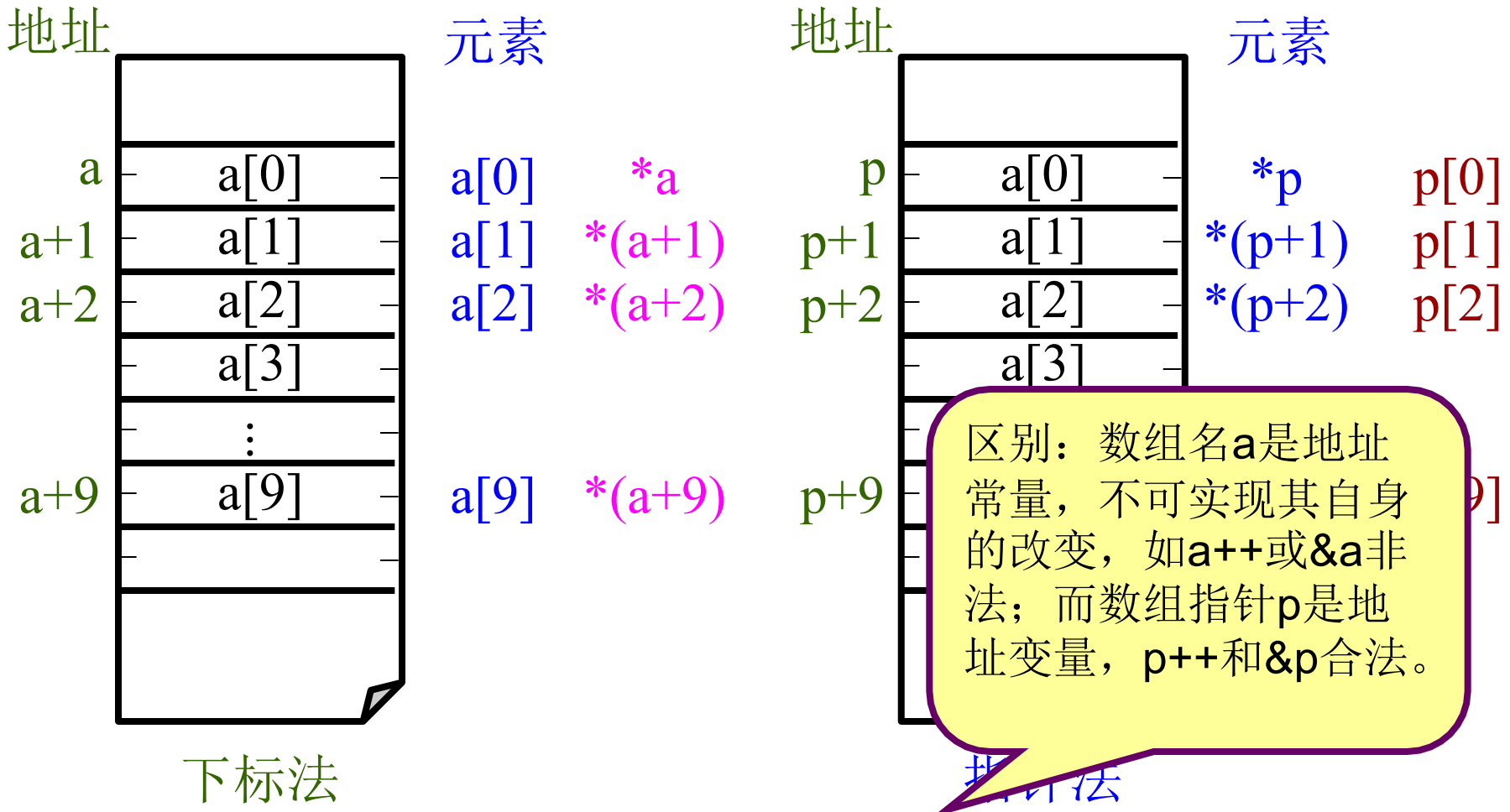
  *(指针变量+i)

  *(数组名+i)

  指针变量[i]

  数组名[i]

## ➢ 数组元素的四种表示方法

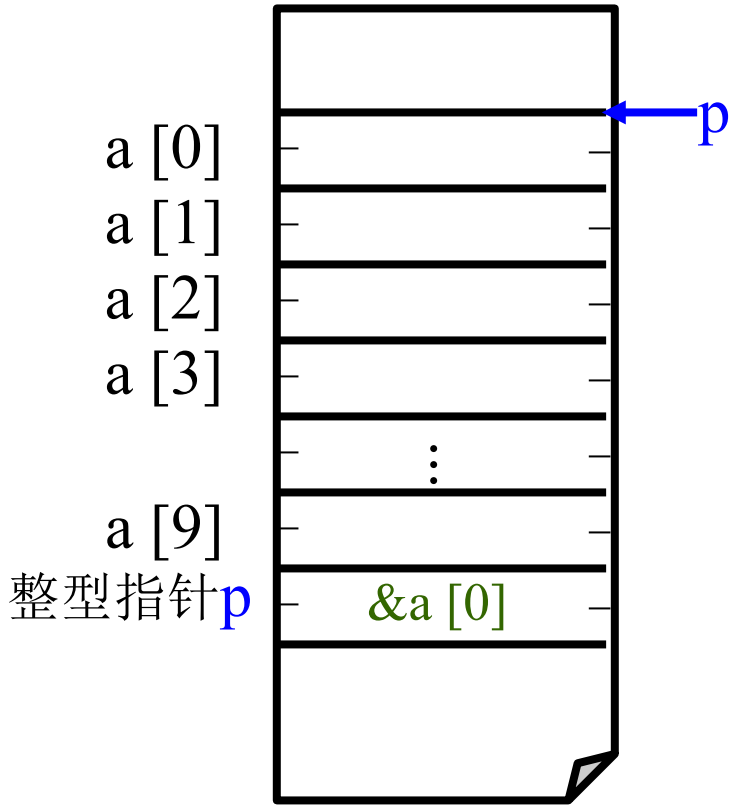


# 7.2 指针和数组

## 1、指针和一维数组

例 int a [10], *p;
p=a; //⇔ p=&a [0];

或 int *p=&a [0];
或 int *p=a;



# 7.2 指针和数组

## 2、指针变量的运算

### ❑ 赋值运算

例
```
pi=&i;          //将变量i地址⇒p
i=pi;           //(×)不能把pi的值⇒整型变量
p=array;        //将数组array首地址⇒p
p=&array[i];    //将数组元素地址⇒p
p1=p2;          //指针变量p2值⇒p1
pj=0;           //空指针，相当于pj=NULL;
```

## ❑ 算术运算

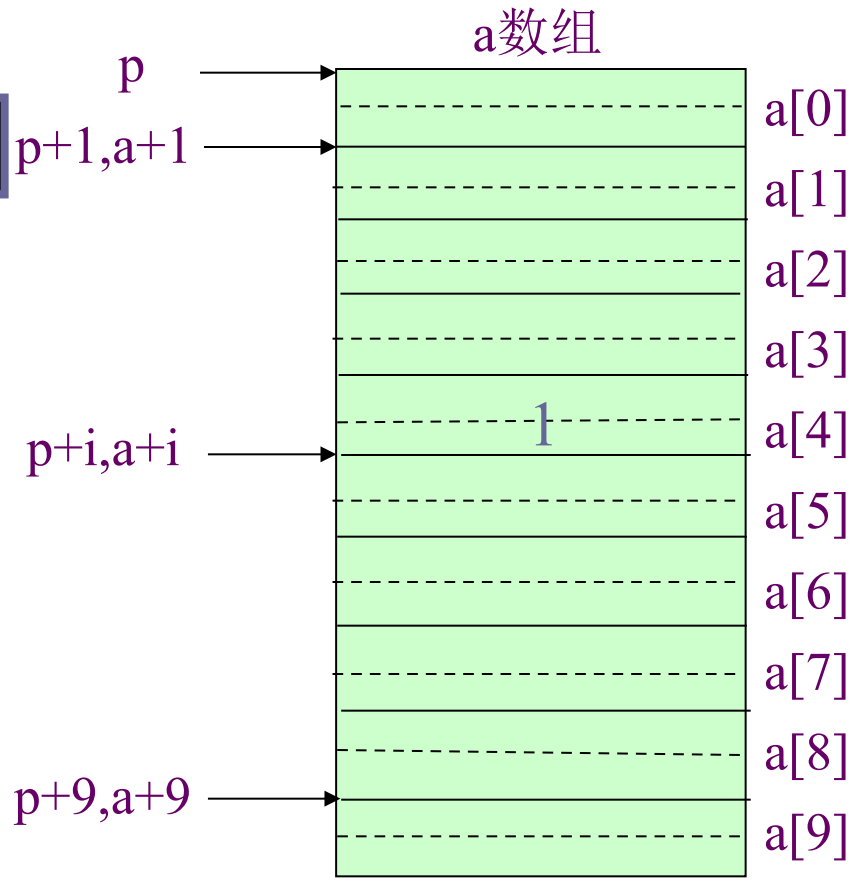
- 若p=&i;　则 p±n⇔ &i ±sizeof(i)×n
- p++, p--, p+n, p-n, p+=n, p-=n的含义
- 若p1与p2指向同一数组，则p1-p2=两指针间元素个数⇔(p1-p2)/sizeof(i)
- p1+p2 无意义

例　p指向float数，则 p+1 ⇔ p+1×4

例　p指向int型数组，且p=&a[0];
　　则p+1 指向a[1]

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

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



# 7.2 指针和数组

## 2、指针变量的运算

- 关系运算：表示两个指针变量所指向地址位置的前后关系。
  - 若px和py指向同一数组，则

    px<py　　表示p1指的元素在前

    px>py　　表示p1指的元素在后

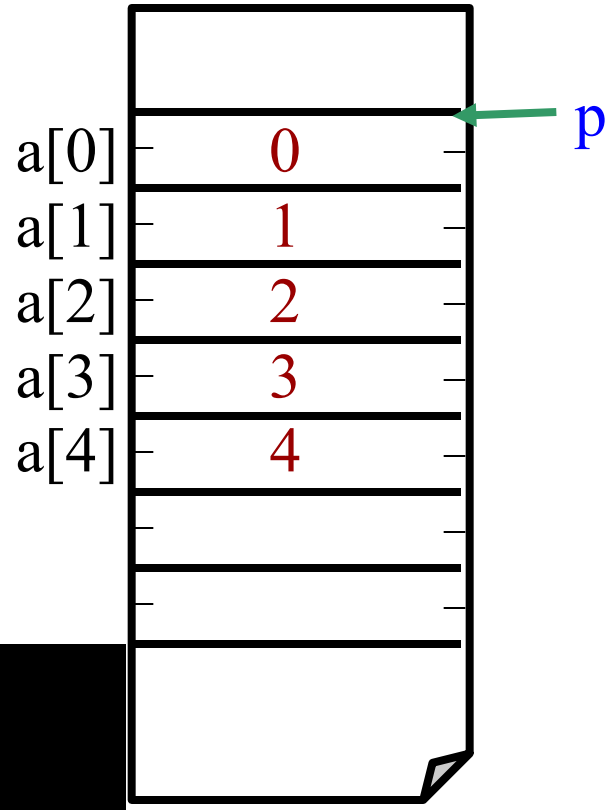
    px==py　　表示p1与p2指向同一存储单元

    px==0　　表示px为空指针⇔p==NULL
  - 若p1与p2不指向同一数组，比较无意义

## 【例7-2】使用指针变量。

```
#include <stdio.h>
int a[]={0,1,2,3,4};
int main(void)
{
    int i,*p;
    for(i=0;i<=4;i++) printf("%d\t",a[i]);
    putchar('\n');
    for(p=&a[0];p<=&a[4];p++) printf("%d\t",*p);
    putchar('\n');
    for(p=&a[0],i=1;i<=5;i++) printf("%d\t",p[i]);
    putchar('\n');
    for(p=a,i=0;p+i<=a+4;p++,i++)
      printf("%d\t",*(p+i));
     putchar('\n');
     return 0;
}
```

```
0   1   2   3   4
0   1   2   3   4
1   2   3   4
25637
0   2   4
```



# 7.3 指针和字符串

## ❑指向字符串的指针变量

- 在数据定义语句中用赋初值的方式

***指针变量=字符串**

- 在程序中用赋值的方式

**指针变量=字符串；**

## ❑指向字符数组的指针变量

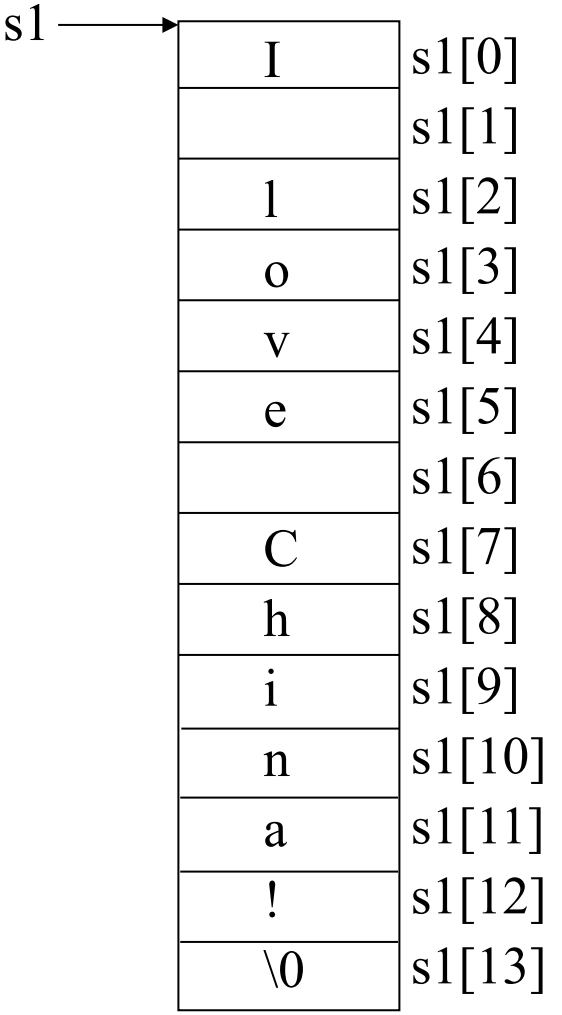
可以将一个字符串赋值给一个字符数组，也可以赋值给一个字符指针变量。

# 7.3 指针和字符串

- **字符串表示形式**
  - 字符数组方式

```
#include <stdio.h>
int main(void)
{
    static char s1[]="I love China!";
    printf("%s\n%c\t%c\n",s1,s1[0],*(s1+3));
    return 0;
}
```

```
I love China!
I       o
```

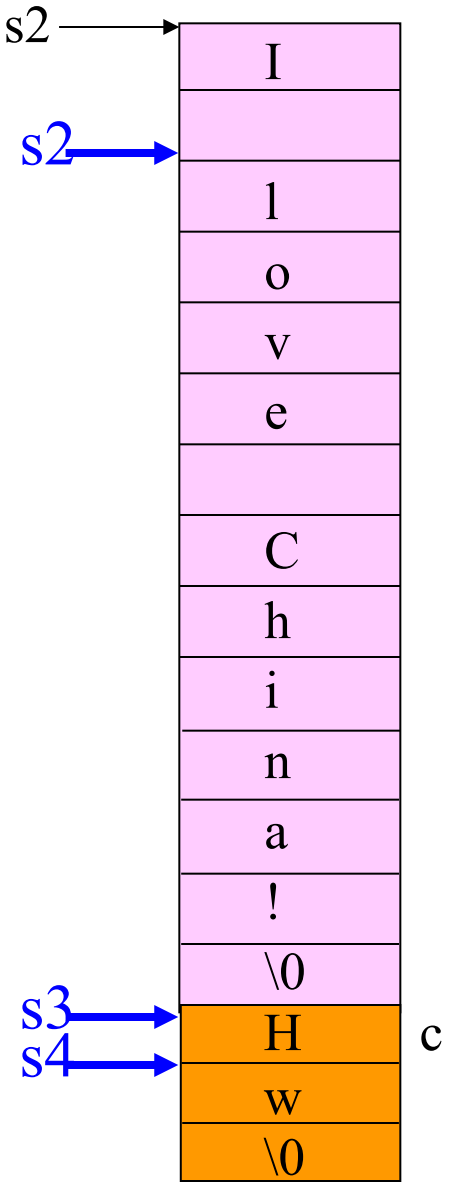


## ❑字符指针方式

字符指针初始化：把字符串首地址赋给string
⇔ char *s2;
s2="I love China!";

```
char *s2="I love China!";
char *s3,c;
char *s4="w";
s3=&c;
*s3='H';
s2=s2+2;
printf("%s\t%c\t%s\n",s2,*s3,s4);
return0;
}
```

Love China!　　H　　w



# 7.3 指针和字符串

- **字符数组与字符指针的区别**
  - 对字符数组整体赋值只能在初始化时进行；而对指针变量赋值，既可在初始化时进行，又可在其他地方赋值。

```
char  *cp; char str[20];
char  str[20];     str="I love China!";    (×)
char   *cp;        cp="I love China!";     (✓)
```

  - str是地址常量；cp是地址变量，cp接受键入字符串时，必须先开辟存储空间。

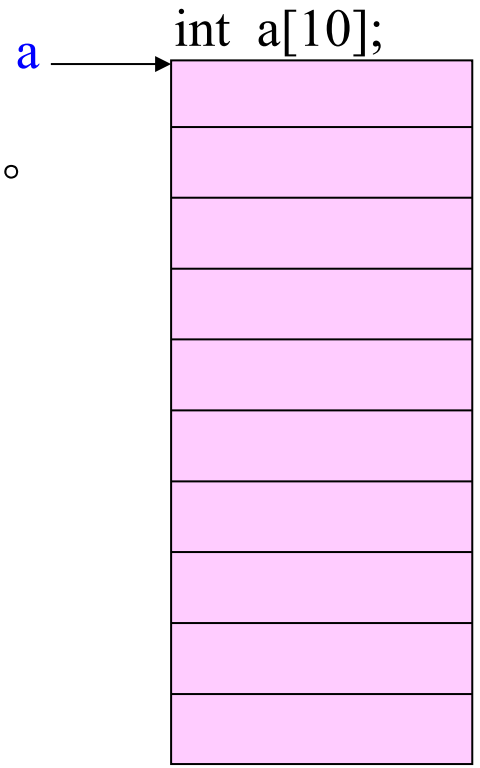
```
例   char  str[10];
     scanf("%s",str);    (✓)
而   char  *cp;
     scanf("%s",  cp);    (×)
```

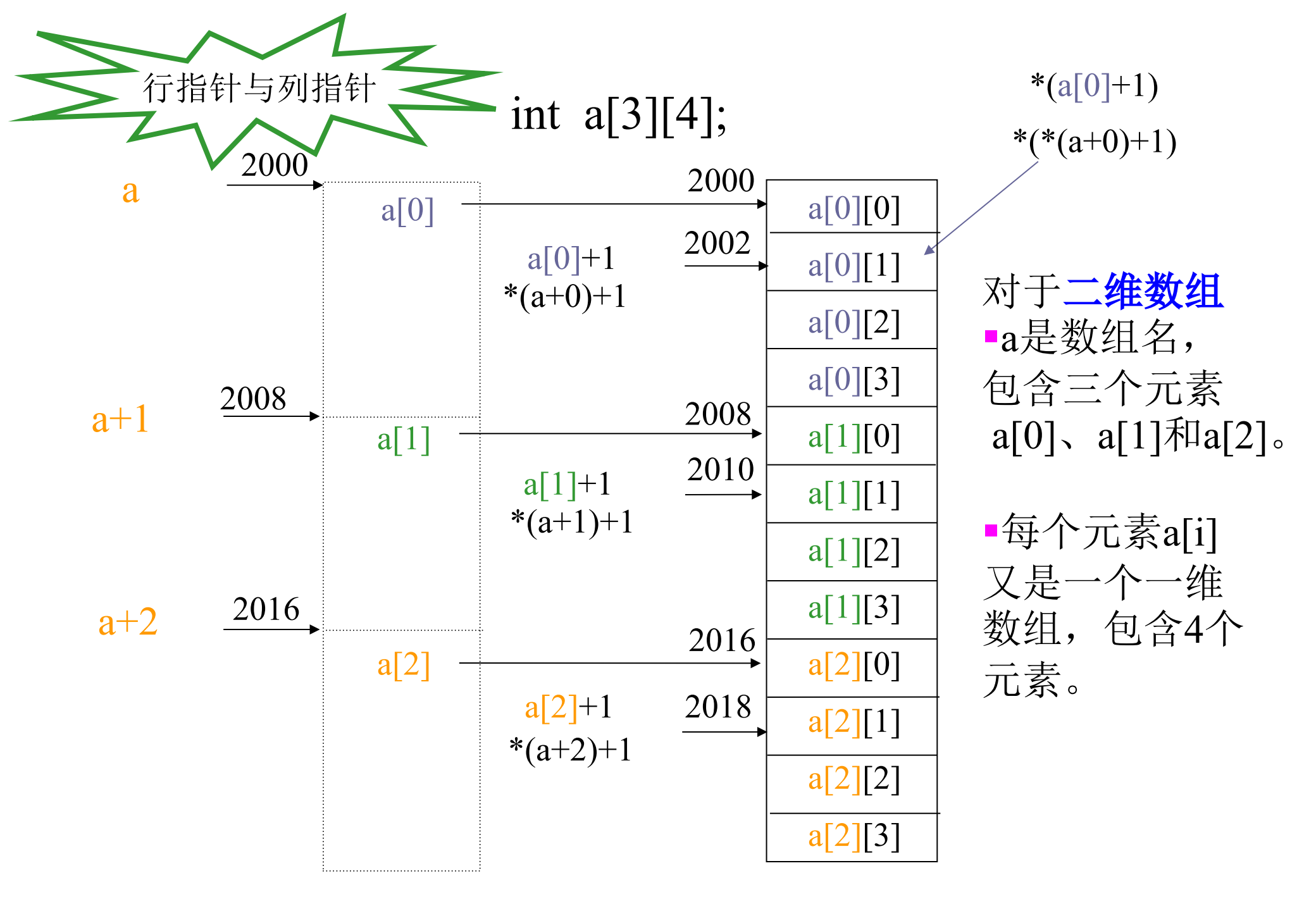
```
改为:  char *cp,str[10];
       cp=str;
       scanf("%s",cp);      (✓)
```

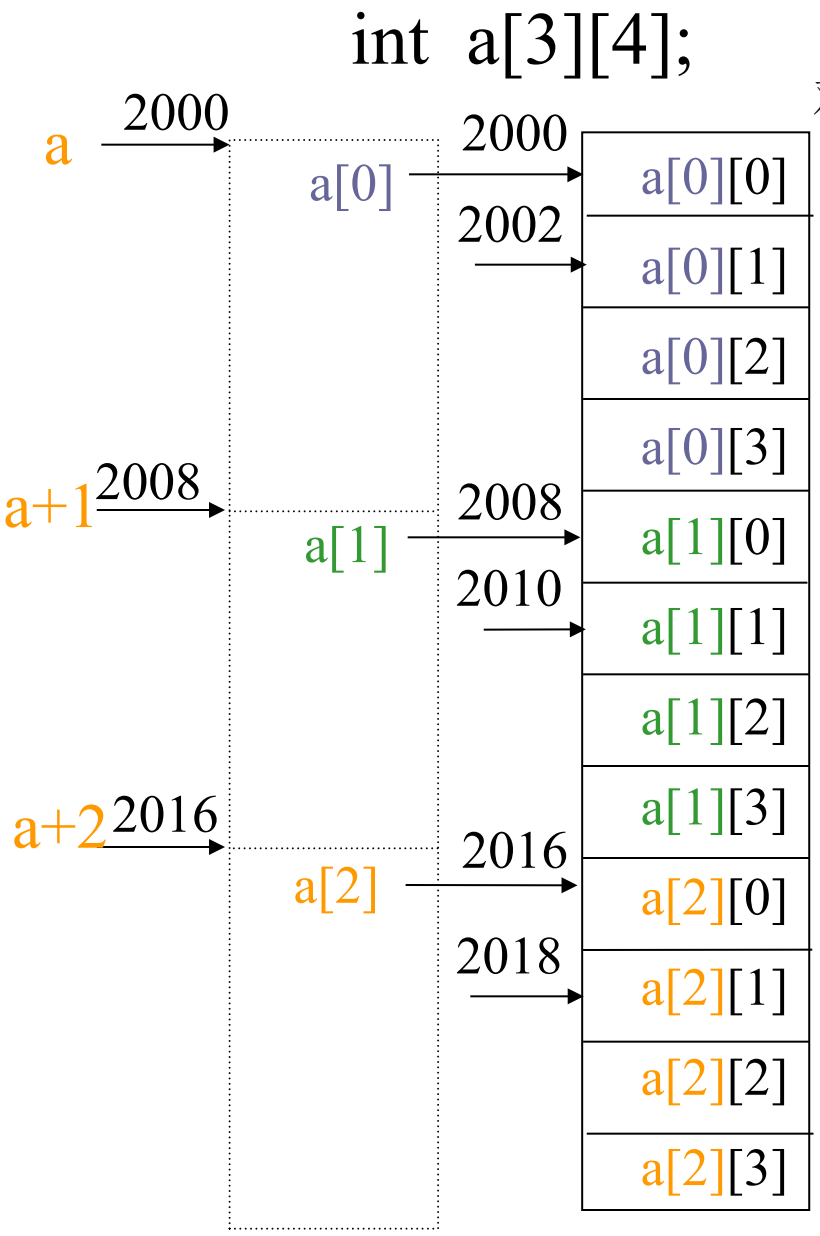
# 7.3 指针和字符串

对于一维数组

- 数组名a表示数组的首地址，即&a[0]。
- 数组名a是地址常量。
- a+i是元素a[i]的地址，即&a[i]。
- a[i] ⇔ *(a+i)




行指针与列指针
int a[3][4];
*(a[0]+1)
*(*(a+0)+1)
2000
a
a[0]
2000
a[0][0]
a[0]+1
*(a+0)+1
2002
a[0][1]
a[0][2]
a[0][3]
2008
a+1
a[1]
2008
a[1][0]
a[1]+1
*(a+1)+1
2010
a[1][1]
a[1][2]
a[1][3]
2016
a+2
a[2]
2016
a[2][0]
a[2]+1
*(a+2)+1
2018
a[2][1]
a[2][2]
a[2][3]
对于二维数组
a是数组名，包含三个元素a[0]、a[1]和a[2]。
每个元素a[i]又是一个一维数组，包含4个元素。




对于二维数组int a[3][4]：

a——二维数组的首地址，即第0行的首地址

a+i——第i行的首地址

a[i] ⇔ *(a+i)——第i行第0列的元素地址

a[i]+j ⇔ *(a+i)+j——第i行第j列的元素地址

*(a[i]+j) ⇔ *(*(a+i)+j) ⇔ a[i][j]

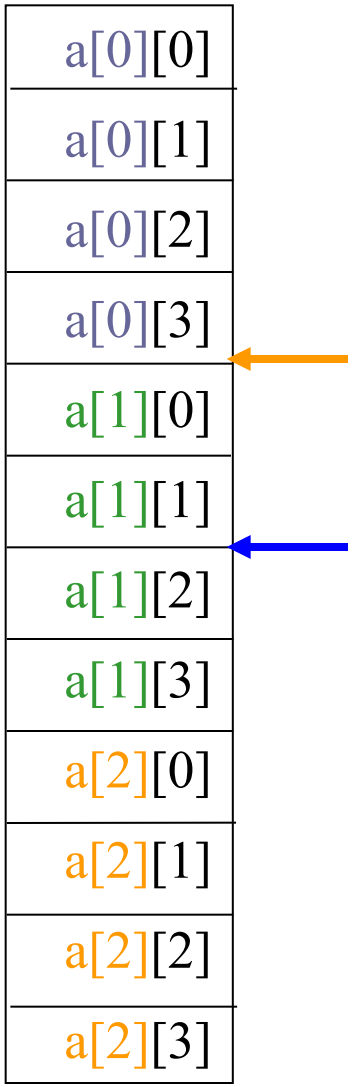
a+i=&a[i]=a[i]=*(a+i) =&a[i][0]，值相等，含义不同

- a+i ⇔ &a[i],表示第i行首地址，指向行
- a[i] ⇔ *(a+i) ⇔ &a[i][0]，表示第i行第0列元素地址，指向列

# 7.3 指针和字符串

| 表示形式 | 含义 | 地址 |
| --- | --- | --- |
| a | 二维数组名，数组首地址 | 2000 |
| a[0],*(a+0),*a | 第0行第0列元素地址 | 2000 |
| a+1 | 第1行首地址 | 2008 |
| a[1],*(a+1) | 第1行第0列元素地址 | 2008 |
| a[1]+2,*(a+1)+2,&a[1][2] | 第1行第2列元素地址 | 2012 |
| *(a[1]+2),*(*(a+1)+2),a[1][2] | 第1行第2列元素值 | 13 |

int a[3][4];




地址表示：

（1）a+1 ← 行指针

（2）&a[1][0]

（3）a[1]

（4） *(a+1) ← 列指针

（5）(int*) (a+1)

地址表示形式：

（1）&a[1][2]

（2）a[1]+2

（3）*(a+1)+2

（4）&a[0][0]+1*4+2

二维数组元素表示形式：

（1）a[1][2]

（2）*(a[1]+2)

（3）*(*(a+1)+2)

（4）*(&a[0][0]+1*4+2)

## 【例7-3】将字符串t复制到s中，并返回被复制的字符个数。

```
#include <stdio.h>
int strcpy(s,t)
  char *s,*t;
  {  int i;  i=0;
     while((*s=*t)!='\0')
     {  s++;
        t++;
        i++;  }
      return(i);  }
  int main(void)
  { int n;
    char *ss,*tt;
    gets(ss); /*scanf("%s",ss);*/
    n=strcpy(*ss,*tt);
    printf("%d,%s",n,tt);
    return 0;
 }
```

```
while(*s++=*t++) i++;
```

# 7.4　指针和函数

## 1、指针作为函数参数

函数的参数不仅可以是整型、实型、字符型等数据类型，还可以是指针类型，因此可以像普通变量一样在函数间传递指针变量的值。

➢ **指针变量作为函数参数**

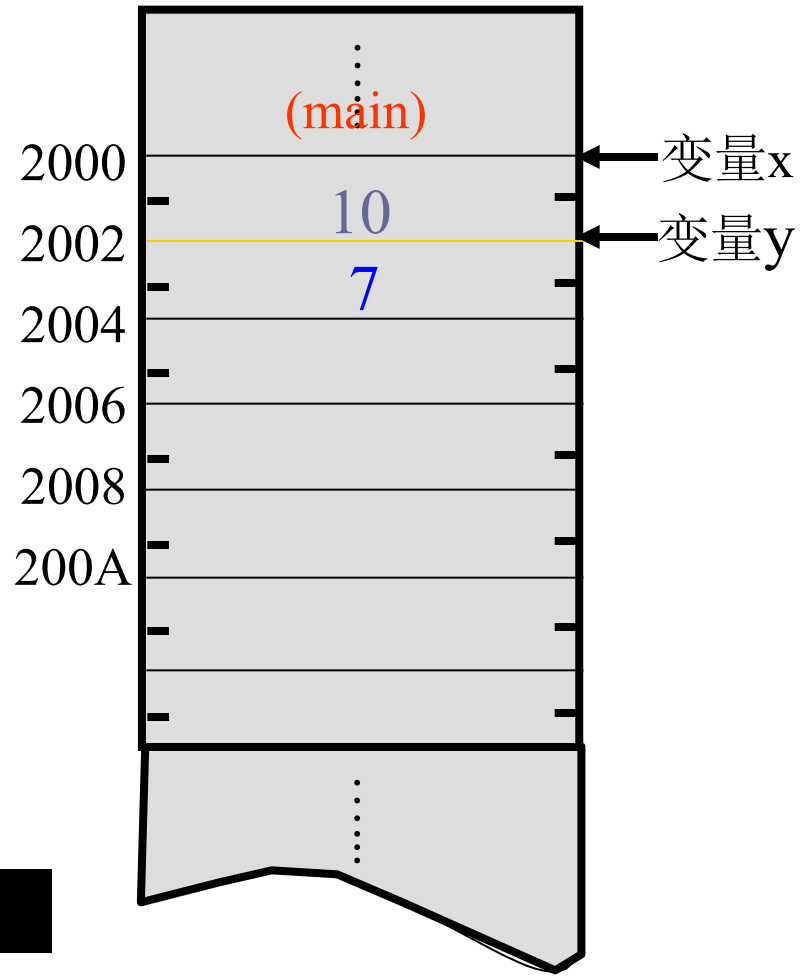
- 指针变量，可以作为函数的形参，也可以作函数的实参。
- 指针变量作实参时，形参和实参之间的数据传递方式本质上是“值传递”，即将指针变量的值（一个地址）传递给被调用函数的形参（必须是一个指针变量）。

***注意:*** *被调用函数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的变量的值。*

## 【例7-4】将数从大到小输出（普通参数）。

```
#include <stdio.h>
int main(void)
{   int x,y;
    scanf("%d,%d",&x,&y);
     if(x<y)  swap(x,y);
     printf("\n%d\t%d\n",x,y);
}
swap(int  x1,int y1)
{   int  temp;
    temp=x1;
    x1=y1;
    y1=temp;
    return 0;
}
```

值传递




```
7   10
```