第9章 指针与数组

本章重点

- 指针的概念与指针赋值
- 指针变量对一维数组元素的引用方法
- 指针变量对字符串的引用
- 指针数组的概念

本章难点

- 如何用指针变量来表示数组元素及元素的地址
- 指针变量在一维数组中的移动
- 字符指针变量的赋值方法及运用
- 数组指针与指针数组的区别
- 数组指针的理解与应用

9.1 指针的基本概念

变量的值存放在内存中，而内存有一确定的地址。在C语言中，有一种变量用来存放内存的地址，这种变量称为指针变量。

1. 指针变量的定义

定义指针变量的一般形式如下：

类型名 * 指针变量名；

例如： int *p1 , *p2;

注意：类型是指针变量所指的地址上存储内容的类型。

2. 指针变量的赋值

一个指针变量可以通过赋值、初始化、分配内存空间来获得一个确定的地址值，从而指向一个具体的对象。

例如，若有以下定义：

```
int k=1,*q,*p;
q=&k;                    /*  指针 q 指向变量 k 的地址   */
p=q;          /*  指针 p 指向指针 q 所指的地址 , 指针变量 p 和 q 都指向了变量 k */
```

注意：当进行赋值运算时，赋值号两边指针变量的基类型必须相同。

3. 指针变量的间接寻址运算

从指针变量所指的地址上取内容可用运算符“ *”。“ *”称为间接寻址运算符，其操作数是一个指针变量。

间接寻址运算符形式如下：

*** 指针；** 它的功能是获取指针所指存储单元的值。

【例 9.1 】 定义一个整型变量与一个指向整型数的指针，让指针指向变量的地址，通过从键盘输入一个数给变量，输出指针所指的地址上的内容。

```
#include<stdio.h>
int main()
{
    int x;
    int *p;  /* 定义指针变量 p*/
    p=&x;    /* 指针变量指向变量 x 的地址 */
    scanf("%d",&x);
    printf("*p=%d\n",*p); /* *p 为指针变量 p 所指的内存上的内容 */
    return 0;
}
```

分析：在图 9.1 中，假定变量 x 分配在内存 2000H 地址上，指针变量 p 分配在内存 2500H 地址上，从键盘读入 100 给变量 x 。由于执行语句 p=&x ；它表示指针 p 指向变量 x 的地址，即指针 p 的值为 2000H ，而 2000H 上的内容就是变量 x 的值 100 。

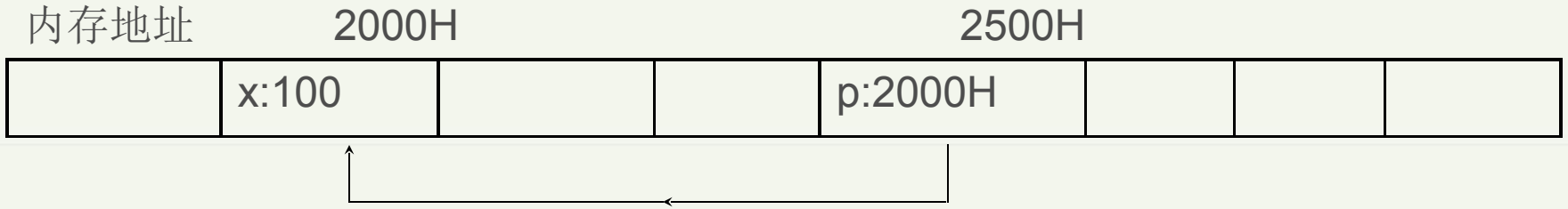

图 9.1　p=&x 的示意图

注意：

1 ）指针变量名是一个标识符，对它的命名要符合标识符的命名规则。

2 ）指针变量只能指向同一类型的变量。

3 ）指针定义时的“ *” 只是定义说明符，它并不是指针变量名的一部分。例如，在定义指针变量 int　*p ；时， p 为指针变量， *p 并不表示指针，而表示指针变量 p 所指地址上的内容。

思考

int x, *p=&x; 能否把此语句中的 *p 理解为指针 p 取内容运算。如果把此语句中的 *p=&x ；单独写出是否正确？

答：不正确，不能这样理解，此处 * 表示指针的定义符，而不是取内容运算符，如果写成 *p=&x 是不正确的，因为此时左边 * 为取内容运算符，右边是变量的地址，应改为 p=&x 才正确。

注意：

C 语言中有一个特殊的指针值 NULL，即全部二进制为 0 的值。引进 NULL 的目的是作为指针的异常事件标志。除了给指针变量赋地址值外，还可以给指针变量赋 NULL 值，例如：

```
p = NULL ;
```

NULL 是在 stdio.h 头文件中定义的预定义符，因此在使用 NULL 时，应该在程序的前面出现预定义行：#include<stdio.h>。NULL 的代码值为 0，当执行了以上的赋值语句后，称 p 为空指针。

上机操作与练习 1

分析下列程序的运行结果并上机调试：

```
#include<stdio.h>
void f(int *p)
{
    *p=*p+10;
}
int main()
{
    int x,i;
    FILE *fp;
    scanf("%d",&x);
    fp=fopen("k.txt","w");
    f(&x);
    for(i=0;i<x;i++)
        fprintf(fp,"%4d%c%c",i,' ','a'+(i+x)%26);
    fclose(fp);
    return 0;
}
```

程序运行后，先输入一个整数，确认后，查找文件 k.txt，检验你思考的结果与文件 k.txt 中的内容是否一致。

9.2 相同类型指针间的基本运算

指针指向内存中的一个地址，所以它是一个整数。但指针又不同于一般的整数，它必须是具体类型的特定变量地址所允许的整数。因此，指针的运算有很大的特殊性。对指针可以进行的运算有算术运算、关系运算和赋值运算。

【例9.2】 两个同类指针之间的算术减运算。

```
#include<stdio.h>
void main()
{
   int a[5];
   int *p1,*p2;
   p1=a;
   p2=&a[3];
   printf("%d \n",p2-p1);
}
```

【例9.3】 两个同类指针之间的算术运算。

```
#include<stdio.h>
void main()
{
   int a[10];
   int *p1,*p2;
   p1=a;
   p2=&a[3];
   p2++;
   p2=p2+2;
   printf("%d \n",p2-p1);
}
```

【例 9.4】　两个同类指针之间的算术减运算及指针与整型数之间的运算。

```
#include<stdio.h>
void main()
{
   int a[5],x,y;
   int *p1,*p2;
   p1=a;
   x=(int)p1;
   p2=&a[3];
   y=(int)p2;
   printf("%u \n",p1);
   printf("%u \n",p2);
   printf("%u \n",p2-p1);
   printf("%u \n",y-x);
}
```

【例 9.5】　两个同类指针之间的关系运算。

```
#include<stdio.h>
void main()
{
   int a[5];
   int *p1,*p2;
   p1=a;
   p2=&a[3];
   printf("%u \n",p2>p1);
}
```

9.3 一维数组与指针

指针是一个很重要的概念。在实际使用中，指针变量通常应用于数组，因为数组在内存中是连续存放的、指针应用于数组将会使程序的概念十分清楚、精练和高效。

【例 9.6】 写出以下程序执行后的结果。

分析： 如图 9.2 所示， p 所指的地址上的内容为 30 ，因而 *p+2 为 32 ，而 p+2 指在元素 a[4] 的地址上，所以 *（ p+2 ）的值为 50 。当 p-- 后，指针 p 指在元素 a[1] 的地址上，此时 *p 、 *p+2 、 *（ p+2 ）分别为 20 、 22 、 40 。

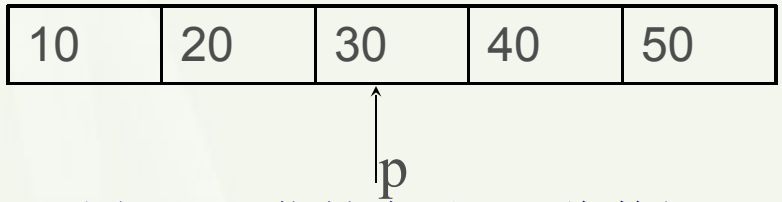

图 9.2 指针应用于一维数组

```
#include<stdio.h>
int main()
{
  int a[5]={10,20,30,40,50},*p;
  p=a+2;
  printf("%d  %d   %d\n",*p,*p+2,*(p+2));
  p--;
  printf("%d  %d   %d\n",*p,*p+2,*(p+2));
  return 0;
}
```

【例 9.7】　如有定义：

```
int a[10]={10,20,30,40,50,60,70,80,90,100}, *p;
p=a+5;
```

假定下列各问题都以指针 p 指向 a+5 的地址上为条件，问 p++ 、 *p++ 、 *++p 、 ++*p 的含义。

分析：

p++ ；表示向高地址移动指针，使指针变量 p 指向存储空间为 70 的地址上。

*p++ ；表示取 60 的值，然后使指针 p 指向存储单元为 70 的地址上。

*++p ；表示先使指针 p 指向存储单元为 70 的地址上，然后取值 70 。

++*p ；表示对 p 所指的值 60 增加 1 ，然后取值 61 。

思考

如有定义：

```
int a[10]={10,20,30,40,50,60,70,80,90,100}, *p;
p=a+5;
```

假定下列各问题都以指针 p 指向 a+5 的地址上为条件，问 p-- 、 *p-- 、 *--p 、 --*p 的含义。

表 9.1　一维数组的元素表示法

数组表示	指针表示
a[i]	p[i]
*（a+i）	*（p+i）

表 9.2　一维数组元素的地址表示法

数组表示	指针表示
&a[j]	p+j
a+j	

【例 9.8】　编写程序，定义一个具有 10 个元素的一维数组，数组元素从键盘输入，找出比相邻元素大的元素个数（将头、尾元素考虑在内）。

分析：如数组定义为 int a[10]; 数组的元素可表示为 *(a+i)、a[i]，其元素的地址可表示为 a+i、&a[i]。统计比相邻元素大的元素个数时可用如右语句：

```
k=0;
if(a[0]>a[1])
   k++;
if(a[9]>a[8])
   k++;
for(i=1;i<9;i++)
if(a[i]>a[i+1]&&a[i]>a[i-1])
     k++;
```

完整的程序代码如下：

```
#include<stdio.h>
void main( )
{
  int a[10],i,k;
  for(i=0;i<10;i++)
    scanf("%d",a+i);
  k=0;
  if(a[0]>a[1])
     k++;
  if(a[9]>a[8])
     k++;
  for(i=1;i<9;i++)
     if(a[i]>a[i+1]&&a[i]>a[i-1])
        k++;
  printf("k=%d\n",k);
}
```

【例 9.9 】　有一个 10 个元素的整型数组 a ，如果想把数组中的 10 个元素按反序存放，请用数组的方法及指针的方法实现。

（ 1 ）用数组的方法编写代码

分析：将 a[0] 与 a[n-1] 对换，再 a[1] 与 a[n-2] 对换，…，直到将 a[(n-1/2)] 与 a[n- (n-1)/2] 对换。现用循环处理此问题，设两个位置指示变量 i 和 j ，i 的初值为 0 ， j 的初值为 n-1 。将 a[i] 与 a[j] 交换，然后使 i 的值加 1 ， j 的值减 1 ，再将 a[i] 与 a[j] 交换，直到 i=(n-1)/2 为止，如图 9.3 所示。

图 9.3　数组交换前后的示意图

用数组元素的方法，程序设计如下：

```
#include<stdio.h>
void inv(int x[ ],int n)    /* 形参 x 是数组名 */
{
  int temp,i,j,m=(n-1)/2;
  for(i=0;i<=m;i++)
    {
      j=n-1-i;
      temp=x[i];
      x[i]=x[j];
      x[j]=temp;
    }
}
```

```
void main()
{
  int i,a[10];
  for(i=0;i<10;i++)
    scanf("%d",a+i);
  printf("The original array:\n");
  for(i=0;i<10;i++)
    printf("%d,",a[i]);
  printf("\n");
  inv(a,10);
  printf("The array has been inverted:\n");
  for(i=0;i<10;i++)
    printf("%d,",a[i]);
  printf("\n");
}
```

（2）用指针的方法编写代码

分析：先定义两个指针 p1 和 p2，让 p1 指向数组的首地址，p2 指向数组的最后一个元素，交换两个指针所指的元素值，然后 p1++ 前移，p2-- 后移，再交换两个指针所指的元素，循环直到 p1 大于 p2 为止。

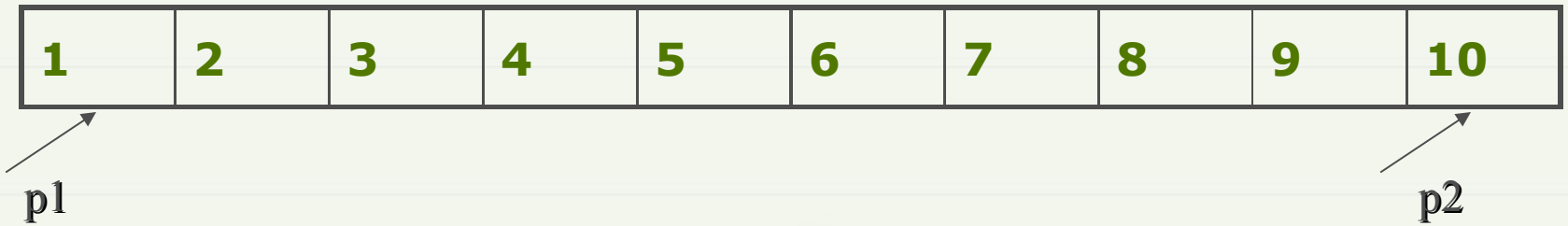

图 9.4　指针 p1 和 p2 分别指向数组首地址与尾地址

用指针的方法，程序设计如下页：

```
#include<stdio.h>
void inv(int *p,int n)
{
  int temp,*p1,*p2;
  p1=p;
  p2=p+n-1;
  for( ;p1<p2;p1++,p2--)
  {
    temp=*p1;
    *p1=*p2;
    *p2=temp;
  }
}
```

```
void main()
{
  int i,a[10],*p=a;
  for(i=0;i<10;i++)
      scanf("%d",a+i);
  printf(" 原数组为 :\n");
  for(i=0;i<10;i++,p++)
      printf("%d,",*p);
  printf("\n");
  p=a;
  inv(p,10);
  printf(" 反序后的数组为 :\n");
  for(i=0;i<10;i++,p++)
      printf("%d,",*p);
  printf("\n");
}
```

9.4　字符串与字符指针变量

9.4.1　字符数组与字符串

1. 字符串的初始化

在 C 语言中，字符串是借助于字符型一维数组来存放的，以字符 '\0' 作为字符串结束标志。 '\0' 的 ASCII 代码值为 0， '\0' 占用存储空间，但不输出，不计入字符串的实际长度。

可以用字符数组存放字符串，例如：

```
char s[10]={'H','e','l','l','o','!','\0'};
```

或

```
char s[10]="Hello!";
```

如果一个字符数组用来存储字符串，那么在定义该字符数组时，数组的大小就应该比它将要实际存放的字符串多一个字节，用来存放‘ \0’。

当用赋初值方式来定义字符数组大小时，这时定义应写成

```
char  str1[8]={'s','t','r','i','n','g','!','\0'};
```

在此定义了有 7 个字符的字符串，数组 str1 包含了 8 个元素的字符数组。可以直接用字符串常量给一维字符数组赋初值。例如：

```
char  str[10]={"string!"};
```

在这里由于“ string ！ ”是字符串常量，系统已自动在最后加入 '\0' ，所以不必人为加入。或写成

```
char   str[]="string";
```

若是定义成

```
char  str[7]="string!";
```

则是错误的，因为 7 个字节的空间不够用， '\0' 将占用下一个不属于 str 的存储单元。

1 ）以下给数组赋值是否合法？

```
char  s[10];
s="Hello!";
```

字符数组 s 可以存放多少个字符？

提示：表示字符串“ Hello ！”的是存储空间的首地址，而数组定义后，数组首地址也就确定了，即是常量，两个常量之间不能相互赋值。

2 ）如有以下定义：

```
char  str1[10]="computer" ,str2[20] ;
```

赋值语句 str2=str1; 是否正确？

2. 字符串的复制

在 C 语言中，字符串 str1 给字符串 str2 赋值，可以有两种方法。

（1）对字符逐个赋值

数组元素间赋值可逐个进行，最后人为加入字符串结束标志 '\0' 。例如：

```
for(j=0;str1[j]!='\0';j++)
    str2[j] = str1[j];
    str2[j] = '\0';      /*  人为加入字符串结束标志 '\0' */
```

上述 for 语句与下述①、②表示的语句等价吗？

① while(str2++=str1++);

② for(j=0;str1[j];j++)
  str2[j] = str1[j];

（2）利用库函数进行拷贝

```
strcpy(str2,str1);
```

调用库函数 strcpy 时应使用预处理命令： #include<string.h> 。

3．字符串的输入 / 输出

（1）字符串的输入

字符串的输入可使用函数 scanf 及 gets 。例如：

```
char  s1[80],s2[80];
scanf("%s  %s",s1,s2);
gets(s1);
```

注意： scanf 输入时以空格为分隔， gets 以回车分隔。

（2）字符串的输出

字符串的输出可使用函数 printf 及 puts 。例如：

```
printf("%s",s1);
puts(s1);
```

它表示输出以 s1 为首地址的内容，直到 '\0' 。

思考

下列程序输出的结果是什么？

```
#include<stdio.h>
void  main()
{
  int  i;
  char string[ ]="I love China!"
;
  for(i=0;string[i];i++)
    printf("%s\n",string+i);
}
```

9.4.2 指针变量与字符串

每一个字符串常量都分别占用内存中一串连续的存储空间，以存储空间的首地址表示字符串，因而可以用字符指针变量指向字符串的首地址。例如：

```
char   *sp ;
sp="Hello!";
```

或写成

```
char *sp="Hello!";
```

此赋值语句并不是把字符串的内容放入 sp 中，而只是把字符串在内存中所占的首地址赋予了 char 类型的指针变量 sp，使指针变量 sp 指向以上字符串的首地址。

注意： 当定义为 char *sp; 用语句 scanf("%s"， p); 或 gets(sp); 是错误的，因为指针 sp 还没有指向一个确定的地址。

思考

1）下列程序段是否正确？为什么？

```
char   *sp;
gets(sp);
```

2）下列程序段是否正确？为什么？

```
char   *sp ,str[80];
sp=str;
gets(sp);
```

3）下列三个函数的不同写法，分析其功能是否相同？

代码 1 ：

```
void strcpy(char *to,char *from)
{
  for(i=0;*(from+i)!='\0';i++)
       *(to+i)=*(from+i);
  *(to+i)='\0';
 }
```

代码 2 ：

```
void strcpy(char *to,char *from)
{
  while (*from!='\0')
  {
  *to=*from  ;
  to++  ;
  from++;
  }
}
```

代码 3 ：

```
void strcpy(char *to,char *from)
{
  while(*to++=*from++);
}
```

9.5　指向二维数组的指针

9.5.1　二维数组的表示及应用

如有二维数组定义 int a[3][4]; 那么数组的元素可表示为 a[i][j]，数组元素的地址可表示为 &a[i][j]。由于一个二维数组可看作多个一维数组，例如数组 a 可看作三个一维数组构成，而每个一维数组由 4 个元素构成，如图 9.5 所示。三个一维数组的首地址分别为 a[0]、 a[1] 和 a[2]。

例如 a[0] 数组，含有 a[0][0]、 a[0][1]、 a[0][2]、 a[0][3] 四个元素。有关二维数组的元素、元素的地址、行首地址的表示方法如表 9.3 所示。

表 9.3　二维数组的元素及地址的表示方法

二维数组的元素表示	二维数组元素的地址表示	二维数组首行地址表示
数组 [i][j]	& 数组 [i][j]	数组 [i]
((数组名 +i)+j)	*(数组名 +i)+j	数组 [i]
*(数组名 [i]+j)	数组名 [i]+j	*(数组 +i)

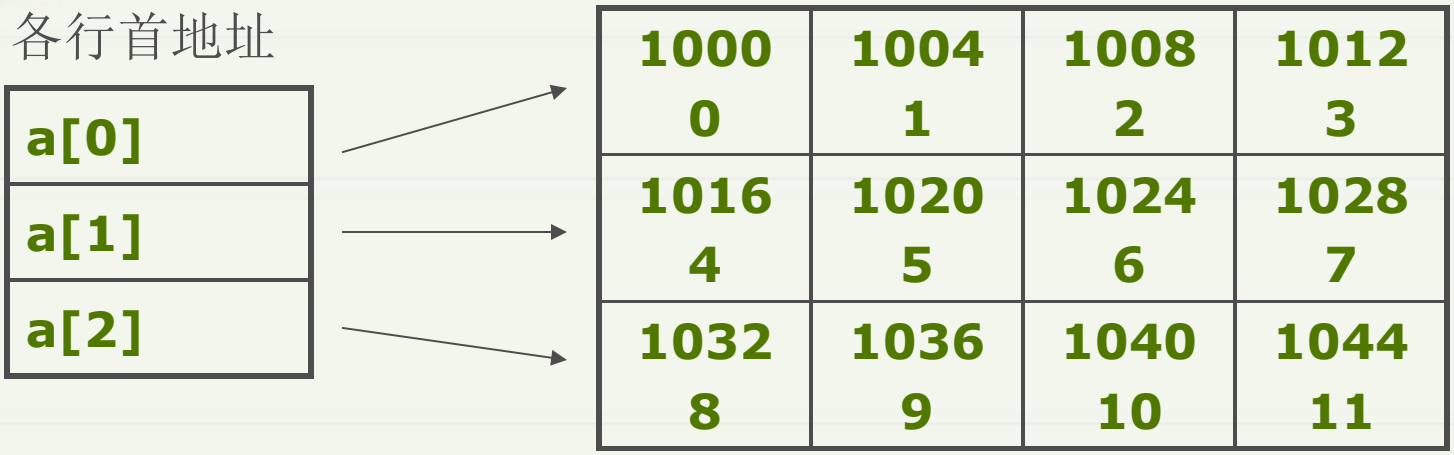

图 9.5　数组 a 可分解为三个一维数组

注意：

1）*(a+i) 表示第 i 行首地址，而 *(a+i)+j 表示元素 a[i][j] 的地址。

2）*(*(a+i)+j)，*(a[i]+j)，a[i][j] 等价，都表示数组的元素。

3）*(a+i)+j 与 *(a+i+j) 不等价。

【例 9. 10 】　定义一个 4×4 整型数组，数组元素从键盘输入，由此数组产生一个 4×6 的新数组，新数组的第 0 列为原数组主对角线，第 5 列为原数组辅对角线。

完整的程序代码为如下：

```
#include<stdio.h>
int main()
{
   int i,j;
   int a[4][4],b[4][6];
    /*   数组从键盘读入：   */
        for(i=0;i<4;i++)
           for(j=0;j<4;j++)
              scanf("%d",&a[i][j]);
    /*  产生新数组：     */
       for(i=0;i<4;i++)
           for(j=0;j<6;j++)
              if(j==0)
                 b[i][j]=a[i][i];
              else if(j==5)
                 b[i][j]=a[i][3-i];
              else
                 b[i][j]=a[i][j-1];
    /*   输出新数组：   */
        for(i=0;i<4;i++){
           for(j=0;j<6;j++)
             printf("%4d",b[i][j]);
           printf("\n");
        }
        return 0;
}
```

9.5.2　指向二维数组的数组指针

1. 数组指针的定义

定义数组指针变量的一般形式如下：

类型说明符　（*指针变量名）[下标];

例如：

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

数组指针指向二维数组，用数组指针可以很灵话地引用二维数组。

例如：

```
int a[3][4],b[4][4];
p=a;
```

可用 *(p[i]+j)、 *(*(p+i)+j) 表示数组元素 a[i][j]，同理 *(p+i) 为数组 a 的第 i 行首地址， *(p+i)+j 表示数组 i 行 j 列元素地址。

同理：

```
p=b;
```

可用 *(p[i]+j)、 *(*(p+i)+j) 表示数组元素 b[i][j]，同理 *(p+i) 为数组 b 的第 i 行首地址， *(p+i)+j 表示数组 i 行 j 列元素地址。

【例 9.11 】　数组指针对二维数组引用的例子。

```
#include<stdio.h>
int main( )
{
        static int a[3][5]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
        int i,j,(*p)[5];
        p=a;
        for (i=0;i<3;i++){
                for(j=0;j<5;j++)
                        printf("%4d",*(*(p+i)+j));
        printf("\n");
        }
        p++;
        printf("**********************\n");
        for(j=0;j<5;j++)
        printf("%4d",*(*p+j));
        printf("\n");
        return 0;
}
```

程序的运行结果如下：

```
   1   2   3   4   5
   6   7   8   9  10
  11  12  13  14  15
**********
*****
   6   7   8   9  10
```

9.6 指 针 数 组

指针数组指数组的元素由指针作为数组元素构成的数组。

1. 指针数组的定义

定义指针数组的一般形式如下：

类型说明符 *数组名[下标];

例如： int *a[4];

它表示一个数组，此数组的元素由指针组成，指针数组具有数组的所有性质，使用指针数组使字符串的处理更为灵活方便。

2. 指针数组的初始化

指针数组的初始化格式如下：

static 数据类型 *数组名[下标]={地址1，地址2，……，地址n};

例如：

```
static char *w[7]={"Sun","Mon","Tue","Wed","Thu","Fri","sat"};
```

它表示数组元素w[0]指向字符串“sun”，w[1]指向字符串“Mon”，……，w[6]指向字符串“sat”，当要输出这些字符串时，可用右旁语句：

```
for （j=0;j<7;j++）
  printf （"%s",w[j]）;
```

3. 指针数组的赋值方法

（1）指针数组的赋值方法之一

```
int a[2][3],*p[2];
for （i=0;i<2;i++） p[i]=a[i];
```

（2）指针数组的赋值方法之二

```
static char *name[]={"Liu" , "Fang" , "Zhang"};
```

这表示指针 name[0] 指向字符串 "Liu" 的首地址；指针 name[1] 指向字符串 "Fang" 的首地址；指针 name[2] 指向字符串 "Zhang" 的首地址。

【例 9.13 】 阅读以下程序，分析输出结果。

```
#include<stdio.h>
void main( )
{
  static char *name[]={"Liu" , "Chang" , "Zhang"};
  int i;
  for(i=0;i<3;i++)
    if(name[i][0]=='C')
  printf("%s\n",name[i]);
}
```

Co

9.7 二 级 指 针

如一个指针变量所指向的地址中存放的变量仍为一个指针变量，而被指向的指针变量指向一个具体的非指针变量，则这个指针称为二级指针，或者说指向指针的指针。依此类推，还可以定义三级指针、四级指针。

1. 二级指针的定义

二级指针的定义形式如下：

<数据类型> **<指针名>;

例如：

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

它们的含义如图 9.6 所示。

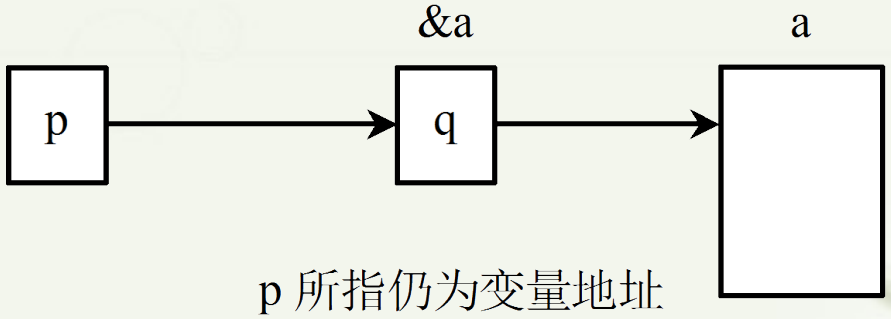

图 9.6 二级指针定义

【例 9.16】　执行下面的语句组后，*a、*p、**p、*pp、**pp 的值分别是多少。

```
int a[ ]={0,1,2,3,4};int *p[ ]={a,a+1,a+2,a+3,a+4};int **pp=p;
```

分析：如图 9.7 所示，a 为数组名，代表数组的首地址，*a 为数组 a 的第一个元素，值为 0；p 为指针数组名，代表指针数组 p 的首地址，因此 *p 为指针数组 p 的第一个元素，即 a，值为数组 a 的首地址；而 **p=*a，值为数组 a 的元素，即 0；pp 是指向指针的指针变量，其初值为指针数组 p 的首地址，*pp 即 *p，值为数组 a 的首地址；**pp 即 *（*p）=*（a），值为 0。

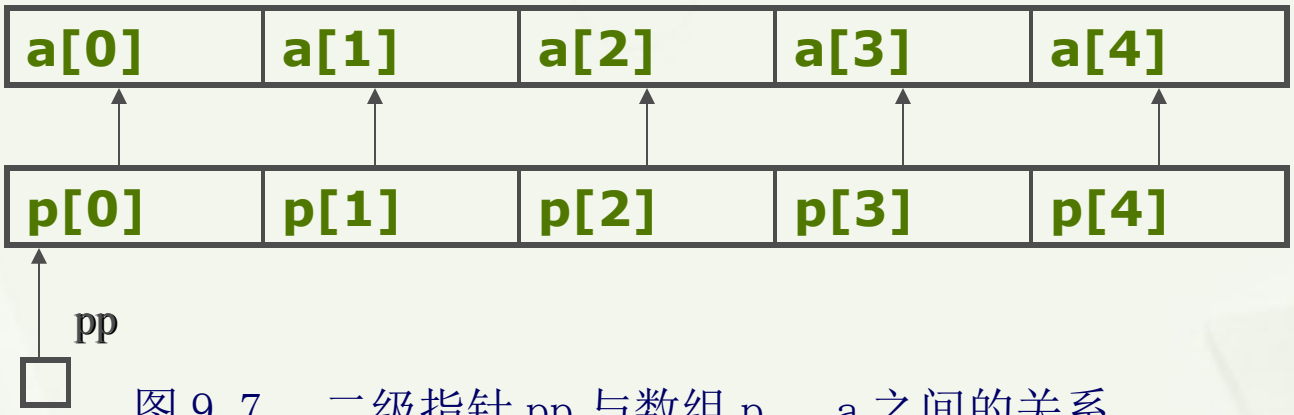

图 9.7　二级指针 pp 与数组 p、a 之间的关系

【例 9. 17 】　阅读以下程序，分析输出结果。

```
#include<stdio.h>
void main( )
{
   int **ppx;
   int *px;
   int x;
   x=10;
   px=&x;
   ppx=&px;
   printf("x=%d\n",x);
   printf("*px=%d\n",*px);
   printf("**ppx=%d\n",**ppx);
}
```

输出结果如下：

x=10

*px=10

**ppx=10

分析：程序中指针所指的情况如图 9.8 所示，输出 *px 、 **ppx 都是指变量 x 的值。

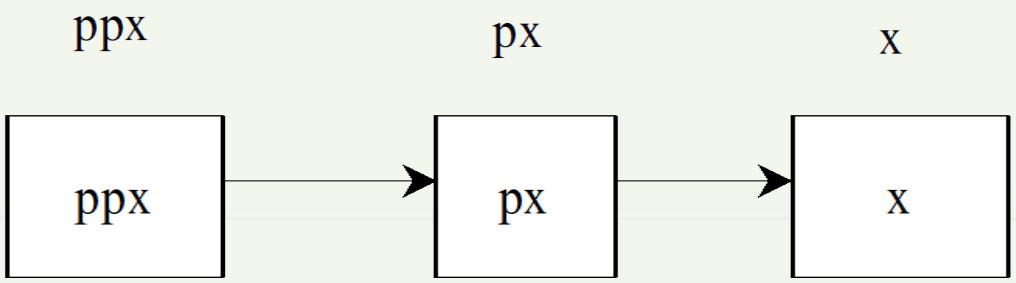

图 9.8　二级指针 ppx 与一级指针 px 、变量 x 之间的关系