第五章　指　针

1 指针的概念与运算

2 指针与数组

3 字符串指针

5.1 指针的概念和定义

- 地下工作者阿金接到上级指令，要去寻找打开密电码的密钥，这是一个整数。几经周折，才探知如下线索：密钥藏在一栋被贴上封条的小楼中。
- 夜晚，阿金潜入小楼，房间很多，不知该进哪一间；忽然走廊上的电话铃声响起。阿金抓起听筒，听筒对面陌生人说：“去打开3010房间，那里有线索”。阿金疾步上楼，打开3010房间，用电筒一照，见桌上写有：“地址2000”。阿金眼睛一亮，迅速找到2000房间，取出重要数据98，完成了任务。

指针的概念

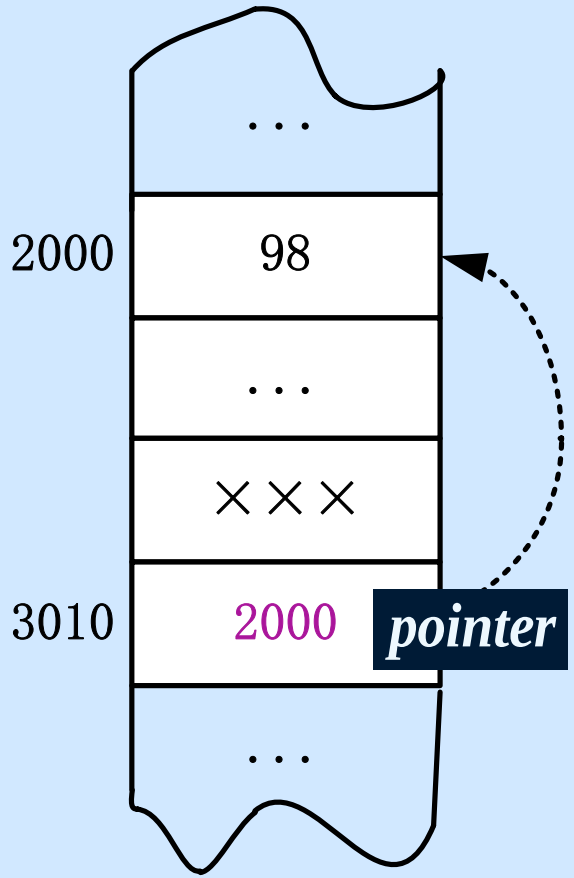


图5-1-1

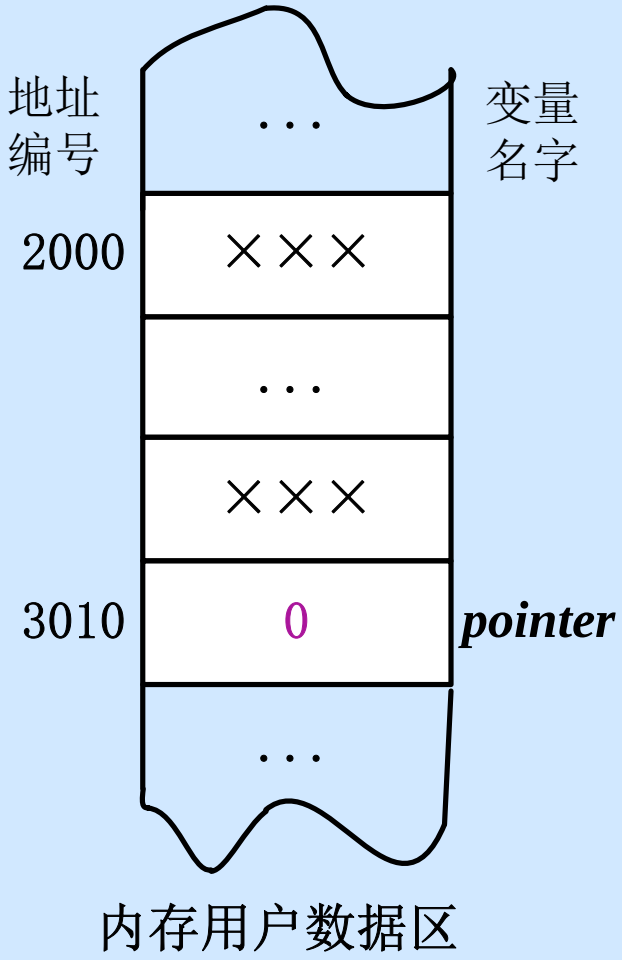
如果我们将房间看作计算机的内存单元，那么，房间的编号就是内存单元的地址，存放地址的内存单元就对应程序中的变量，这类特殊的变量在C中称***指针变量***。这里，pointer是指针变量，2000是pointer指针变量的值

1、数据存放在一个内存单元中，地址是2000。

2、地址2000又由pointer单元所指认，pointer单元的地址为3010。

3、98的直接地址是2000；98的间接地址是3010；3010中存的是直接地址2000。

4、称pointer为指针变量，2000是指针变量的值，实际上是数据在存储器中的地址。



指针：就是地址，一个变量的指针就是该变量的地址。

指针变量：专门存放目标变量的**地址**的一类特殊的变量。

指针变量也简称为指针。

指针指向的目标变量用指针变量名前加间接运算符*表示。如图地址2000所指向的目标变量可用*pointer表示。

指针是一种特殊的变量，特殊性表现在类型和值上。

从变量的角度讲，指针也具有变量的**三个要素**：

- （1）变量名，这与一般变量取名相同。
- （2）指针变量的类型，是指针所指向的变量的类型，而不是自身的类型。
- （3）指针的值是某个变量的内存地址。在32位微机系统中指针变量在内存中占有4个字节。

- 从上面的概念可知，指针本身类型是int型，因为任何内存地址都是整型的。但是**指针变量的类型却定义成它所指向的变量的类型**。

指针的定义及初始化：

指针变量定义的一般形式：

指向类型 *指针变量名；

指向类型是指针变量所指向的目标变量的数据类型，“*”是说明符，说明它所定义的是指针变量。

```
int *pointer = 0 ;
```

上面的语句定义了一个名为pointer的指针，且被初始化为0，该指针“指向”的目标类型为整型。

指针pointer一旦被定义，系统为pointer分配4字节的内存单元。这里给pointer赋值为0/NULL/’\0’，是将指针pointer初始化为0。0是唯一允许赋值给指针变量的整数值，值为0的指针不指向任何对象，但可以防止其指向未知的内存区域，是常用的指针变量的初始化方法，值得提倡。

指针变量的常见类型的定义

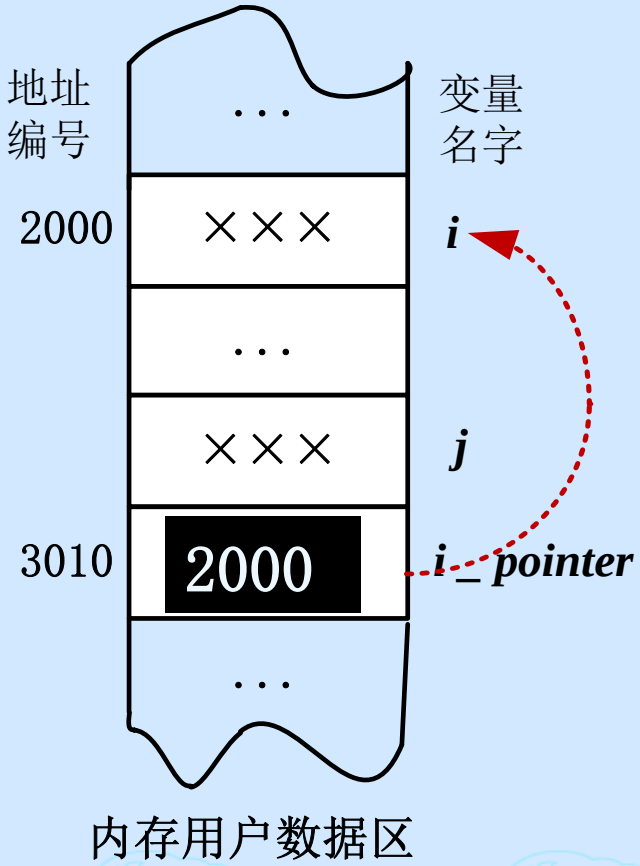
简单的指针变量的定义如下：

```
int *p, *q;    // 定义p,q为指向整型变量的指针
float *point; // point为指向float型变量的指针
char *p1=0; //p1位指向字符型变量的指针
double *pd;    // 定义pd为指向double型变量的指针
int (*pa)[10];// 定义pa为指向int型数组的指针
int **qq;          // 定义qq为指向int型指针的指针
StudentInfo *pStu; // 定义指向结构的指针pStu
```

指针的赋值：

给指针赋值就是将一个内存地址装入指针变量，有两种方法：

1. 使用取地址运算符(&)将变量的地址取出赋给指针变量，一旦赋值就意味着指针指向了该内存单元。




```
int i, j;
int *i_pointer = 0;
i_pointer = &i;
```

执行以上变量赋值语句后， i_pointer 就指向了变量 i 代表的内存单元。

2、将一个已有具体指向的指针变量赋值给另一个指针变量。

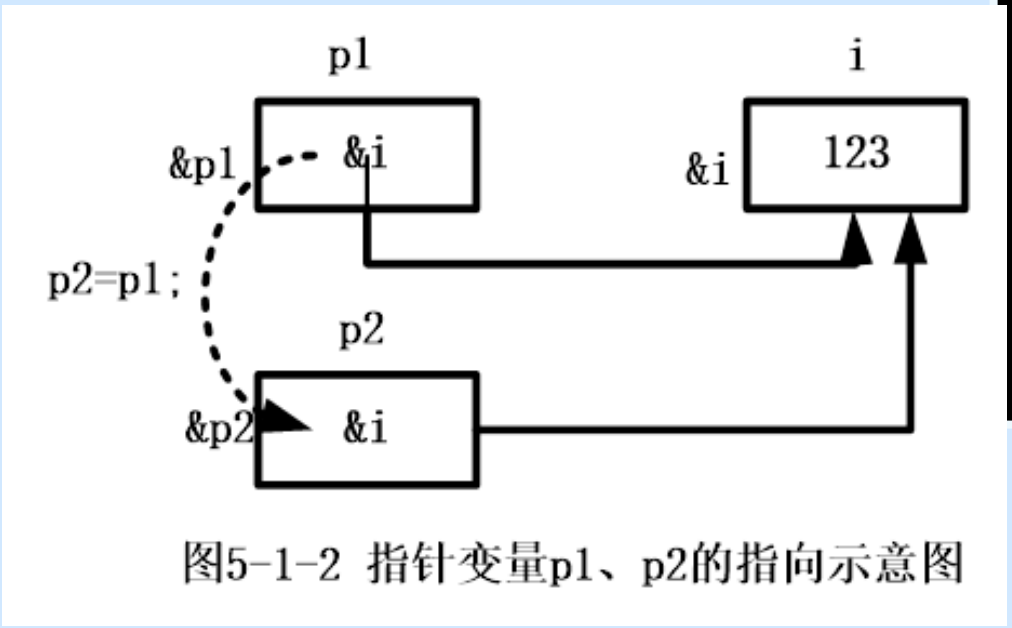


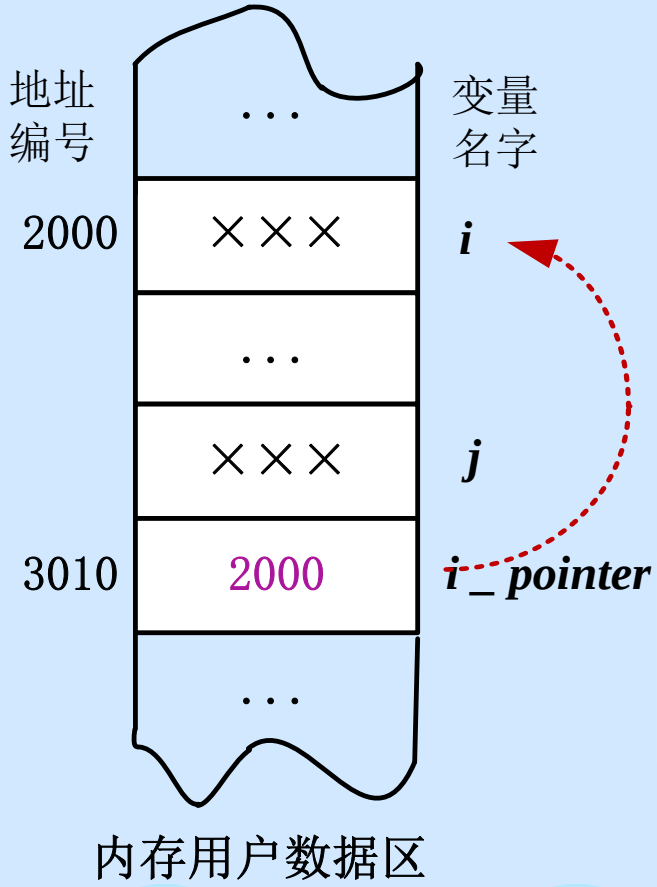
图5-1-2 指针变量p1、p2的指向示意图

```
int  i = 123;
int  *p1 = 0 , *p2 = 0;
p1 =  &i ;
p2 = p1;
```

执行以上变量赋值语句后，p1、p2 就指向了相同的内存单元。

5.2 指针的运算

变量（内存）的访问方式：



①直接访问：通过*变量名*访问。

```
i = 98;
cout<<"i="<<i<<endl;
```

②间接访问：使用间接访问运算符（*），通过*指针变量*中存放*的值*，找到最终要*访问*的变量。

```
*i_pointer = 98;
cout<<"i="<< *i_pointer <<endl;
```

通过指针变量访问变量：例

```
void main( )
{
   int  a,  b;
   int  *pa,  *pb;
   pa = &a;  pb = &b;
   a = 100;  b = 200;
   printf("a=%d,  b=%d\n",  a,  b);
   printf("a=%d,  b=%d\n", *pa,  *pb);
}
```

先让pa和pb 分别指向变量a和b

再给变量a和b赋值。

同样可以使用指针变量间接访问普通变量

通过指针变量访问变量：例

```
void main( )
{
   int  a,  b;
   int  *pa,  *pb;
   pa = &a;  pb = &b;
   *pa = 100;  *pb = 200;
   printf("a=%d,  b=%d\n",  a,  b);
   printf("a=%d,  b=%d\n", *pa,  *pb);
}
```

先让pa和pb 分别指向变量a和b

再给pa和pb指向的内存单元赋值。也就是给变量a和b赋值

两个有关指针的运算符（单目）

① 取地址运算符：&

注意：只能取变量或数组元素的地址，不能用于非左值表达式（指可以放在赋值运算符左边）和常数。

例如：

```
int  i=0,a[3]={0};
```

&i, &a[0]合法；&a, &(i+5),&i=123 均为非法（a为常数不能用于取地址运算符，&i也是一个常数不能被赋值，i+5为非左值表达式）

② 指针运算符（间接运算符）：*

优先级：与取地址运算符&同级

结合性：右结合性

用法：　　*指针变量名

例如：

```
int  i=0, *p1=0, *p2=0; p1=&i; p2=&i;

*p1=*p1+1; //相当于 i=i+1

*p2=321; //相当于 i=321
```

注意：第一行中*被用来申明指针，意味着指向目标变量的指针；后面两行*被用作间接运算符，表示访问所指向的目标变量；另外还应区分*用作乘法运算符的情况！

有关运算符*和&的说明

① 假设有 int a, *p; p = &a; 则：

- &*p 相当于 &a，因为 *p 就相当于 a
- *&a 相当于 *p，因为 &a 就相当于 p
- (*p) ++ 相当于 (a ++)

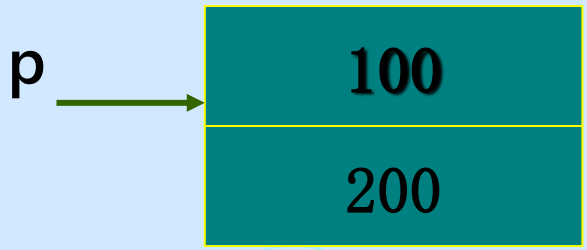
② 注意 (*p) ++ 和 *p ++ 的区别

由于 ++ 和 * 根据优先级（后缀较高），则根据结合性（右结合性），

p ++ 相当于(p ++)，与(*p) ++是完全不同的。

*(p ++)：先得到p当前指向的单元的值，再使p自增，指向下一个单元。

(*p) ++ ：是p所指向的单元的值自增



*(p ++)：得到100，p自增

(*p) ++ ：得到100，p所指向的单元变成101，而p不变

有关指针变量的说明

① 在某一个时刻，一个指针变量只能指向某一个同类型的变量

```
int   *pa,  *pb,   a,   b;
pa = &a;   pb = &b;   pa = &b;  pb = &a;
```

② 必须对指针变量进行了正确合法的初始化后，才能使用该指针变量访问它所指向的内存单元。没有具体指向的指针变量叫悬空指针，对它所指向的内存单元的使用是非法的。

```
int   *pa,  *pb,   a,   b;
*pa = 100;
或 cout<<"*pa="<<*pa<<endl;
```

有关指针变量的说明

③ 不能使用该指针变量去随意访问其它不确定的内存单元，否则，结果是不可预料的。

```
int  *pa, *pb,  a,  b;
pa = &a;  *pa = 100;
pb = &b;  *pb = 200;
cout<<"value after a = "<<*(pa + 1)<<endl;
cout<<"value after b = "<<*(pb + 1)<<endl;
*(pa + 1) = 1000;
*(pb + 1) = 2000;
```

正确安全使用指针变量

错误使用指针变量

错误使用指针变量而且可能很危险

5.3 指针和数组

5.3.1 指针与一维数组

数组名实际上是一个指针！

1. 数组的首地址也就是数组中第1个（下标0）元素的地址；
2. 在内存中数组的元素的地址是连续递增的，数组名可以代表数组的首地址，如score和&score[0]等价；
3. 通过数组的首地址，加上偏移量就可以依次得到其它元素的地址；

   &score[0]+偏移量 -> &score[1]

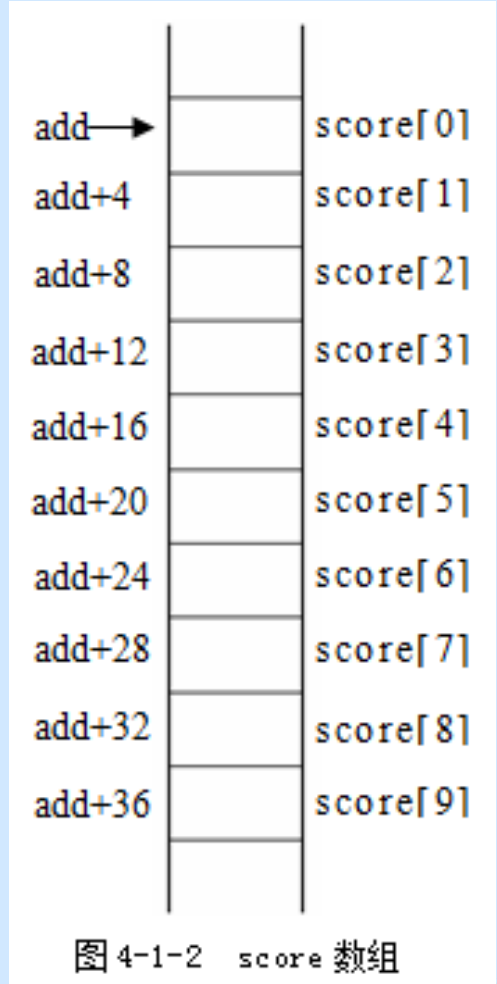


图 4-1-2 score 数组

这里的偏移量就是一个数组元素所占的字节数。编译程序会根据数组元素的类型，自动确定出不同的偏移量。

```
#include <iostream>                                   // 宏包含预编译命令
using namespace std;                                  // 引用标准命名空间
int main()                                            // 主函数
{                                                     // 主函数开始
    short arr_short[3] = {0};                         // 定义short类型的数组
    float arr_float[3] = {0};                         // 定义float类型的数组
    double arr_double[3] = {0};                       // 定义double类型的数组
    int i =0 ;                                        // 数组下标变量
    cout << "\t\tshort类型数组\tfloat类型数组\tdouble类型数组" << endl;
    cout << "==========================================" << endl;
    for(i = 0; i < 3; i++)
    {                                                 // 输出以上三种类型的数组的每个元素的地址
        cout << "元素" << i << "的地址" << "\t" << arr_short+i <<
            "\t" << arr_float+i << "\t" << arr_double+i << endl;
    }
    return 0;
}                                                     //主函数结束
```

arr_short+i等价于&arr_short[i]

```
                short类型数组    float类型数组    double类型数组
==========================================================
元素0的地址     003EFD1C        003EFD08        003EFCE8
元素1的地址     003EFD1E        003EFD0C        003EFCF0
元素2的地址     003EFD20        003EFD10        003EFCF8
```

在C程序中，程序员不必关心数组元素之间的地址偏移量是多少，只要把前一个元素的地址加1就可得到后一个元素的地址。

指针与数组的结合：使用指针访问数组元素

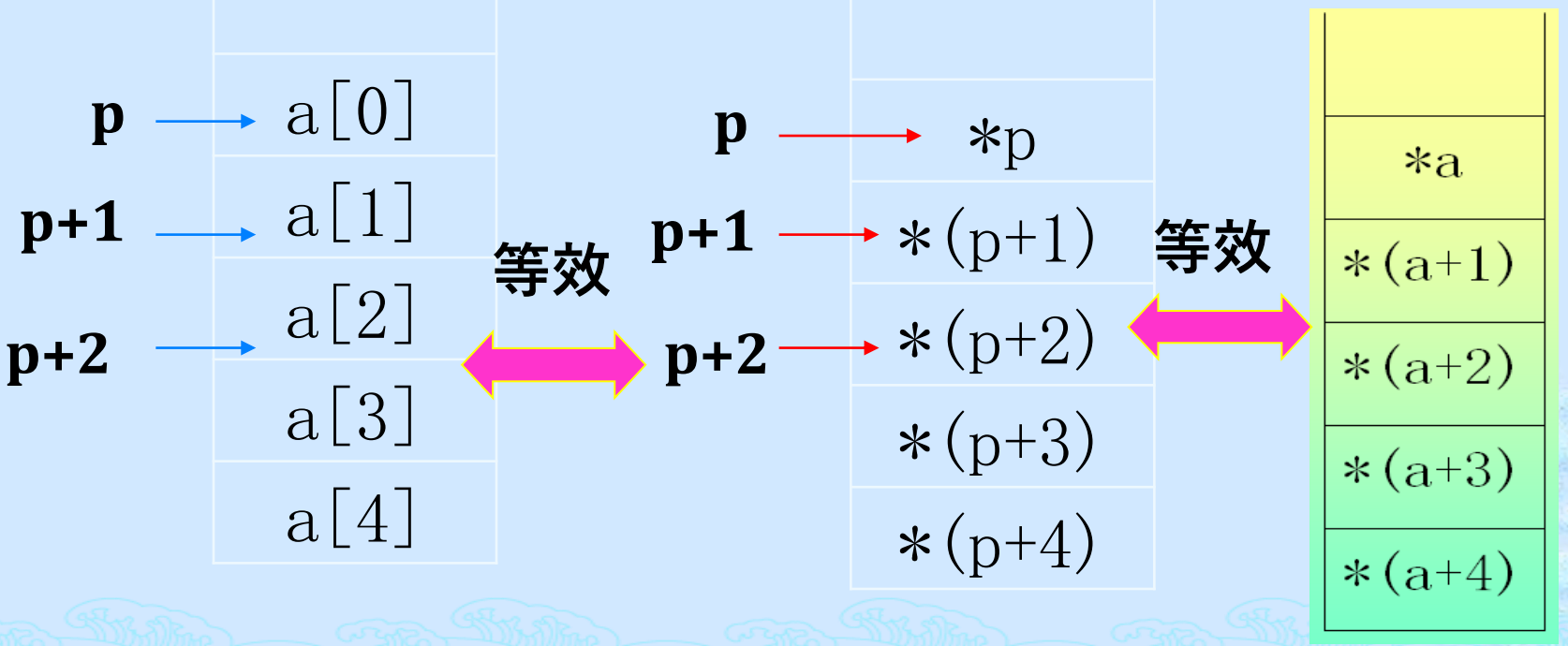
数组名是一个地址常量，不能被修改。可以申明一个指针变量p指向该数组，由于p是指针变量，可以修改它使之指向其他地方。

```
#include <iostream>      //预编译命令
using namespace std;
int main()               //主函数
{                                //函数体开始
    int a[5]={1,3,5,7,9};//定义数组，赋初值
    int *p=NULL;                 //定义指针变量
    int i;                       //定义整型变量
    p=a;                         //赋值给指针变量，让p指向a数组
    for(i=0;i<5;i=i+1)
    {                            //循环体开始
        cout<<"a["<<i<<"]="<<*p<<endl;
                                 //输出a数组元素的值
        p=p+1;                   //指针变量加1
    }                            //循环体结束
    return 0;
}                                //函数体结束
```

说明

- (1) p=a; **这**里数**组**名作**为**数**组**的起始地址， 即a[0]的地址。因此 p=a 等效于 p=&a[0];
- (2) p=p+1; 如p指向a[0], **则**p=p+1之后， p指向a[1]
- (3) 如果p=a 等效于 p=&a[0];

  则 p=a+4 等效于 p=&a[4];
- (4) **获**取数**组**起始地址的方法可以通**过**数**组**名和通**过**数**组**首元素地址两种方式。



做下面的实验

```
#include <iostream> // 预编译命令
using namespace std;
int main()              // 主函数
{                       // 函数体开始
    int a[5]={1,3,5,7,9}; // 定义数组，赋初值
    int *p=NULL;                // 定义指针变量
    int i=0;                    // 定义整型变量,赋初值
    for(p=a;p<a+5;p=p+1)  // 让p指向a数组
    {                           // 循环体开始
        cout<<"a["<<i<<"]="<<*p<<endl;
                                // 输出a数组元素的值
        i=i+1;              // 让i加1
    }                           // 循环体结束
    return 0;
}                               // 函数体结束
```

数组score首地址访问数组元素的方式有以下几种：

1. 数组元素直接访问，即下标变量方式，如score[i];
2. 指针加偏移量的间接地址访问，如*(p+i);
3. 数组名作地址值的直接地址访问，如与score[i]等价的*(score+i);例5.4.
4. 将指针变量看做数组名的下标变量方式，如p[i]。见例5.5 。

数组名是一个常量指针，其值固定不变。但可以通过指针变量的自增自减遍历数组元素。但需要注意越界问题！

如果p指向a[k],那么

P++、*(p++)、*p++、*(++p)、(*p)++分别代表什么？

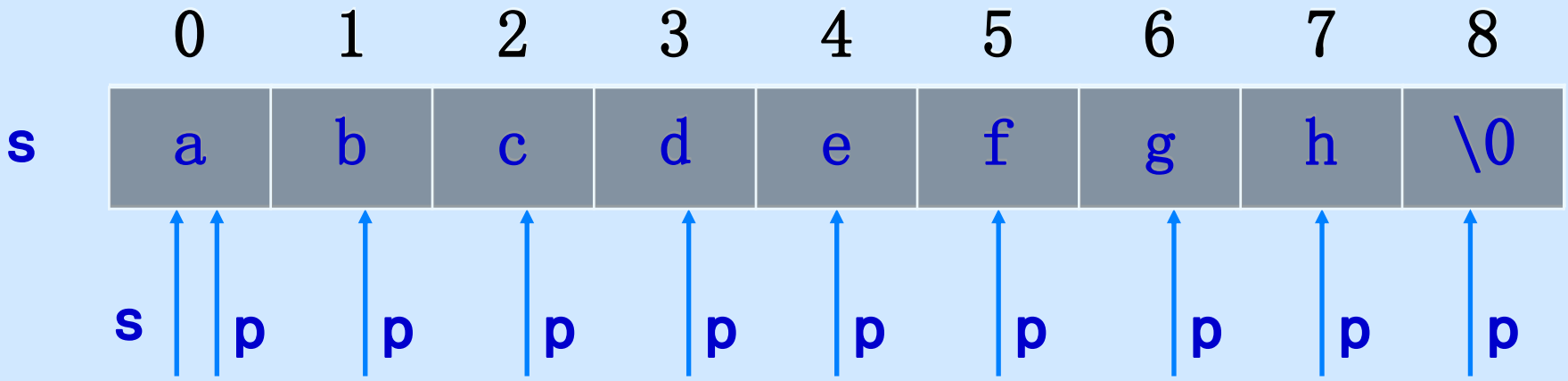
先引用p当前指向的元素a[k]，再使p指向下一个元素a[k+1]

等价于p++，*p。先使p指向a[k+1]，再引用a[k+1]。

阅读p152页最上面一段总结的指向数组指针的运算，总结那些是常见的有关指针的运算，那些运算是错误的应该避免。

数组名是一个常量指针，其值固定不变。但可以通过指针变量的自增自减遍历数组元素。但需要注意越界问题！

```
#include <iostream>
using namespace std;
int main()
{
        char *p;                        // 定义指向字符类型的指针变量p
        char s[] = "abcdefgh";          // 定义字符数组，并赋值
        p=s;                            // 数组名是一个常量指针，
                                        // 它指向该数组首地址
        while(*p != '\0' )              // 当p所指向的元素不为'\0'时
        {
                p++;                    // 让指针加1
        }
        cout<<"字符串长度为"<<p-s<<endl;
        return 0;
}
```



图中数组的首地址是s[0]的地址，即&s[0]。s可看作是指向s[0]的指针。s是不会动的，是常量指针。

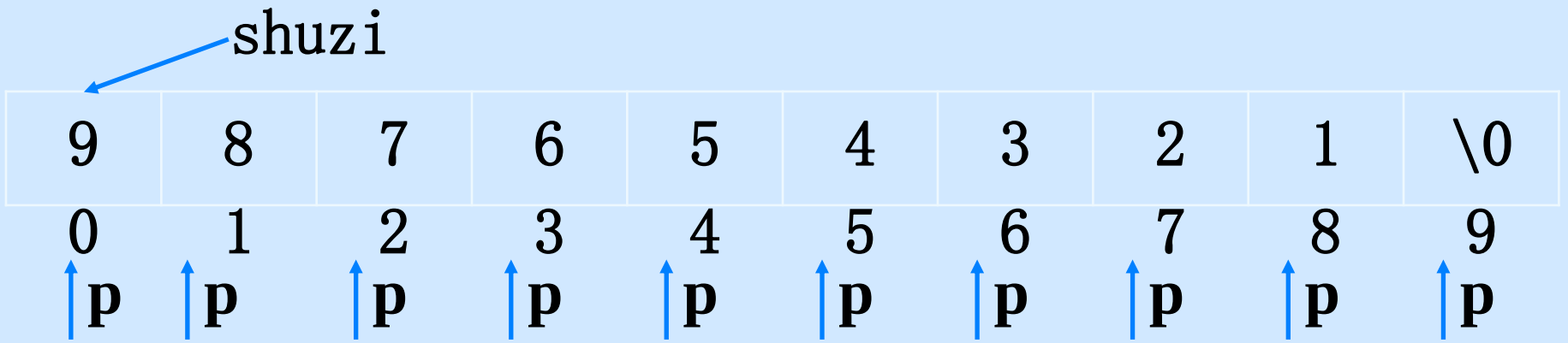
p是指针变量，其值可以改变

执行p=p+1后，p后移一格，指向s[1]

至p指向'\0'为止

数组名是一个常量指针，其值固定不变。但可以通过指针变量的自增自减遍历数组元素。但需要注意越界问题！

```
#include <iostream>
using namespace std;
int main()                              {
    char shuzi[]="987654321";           //定义数组,
                                        // 赋初值为数字字符串
    char *p=&shuzi[8];                  // 让p指向shuzi[8]元素,
                                        // 该处是字符'1'
    for(;p>=shuzi;p--)                  // 当p>=shuzi时循环
        cout<<*p;                       // 输出一个由p指向的字符
    cout<<endl;                         // 换行
    return 0;
}
```



说明：

1、字符串： 数字字符串。

2、p指向shuzi[8]，即指向串中的字符’1’ 。

3、直到型循环，用putchar函数将shuzi[8]输出到屏幕；之后让p=p-1。

4、在while中，当p>=shuzi则继续执行循环体。一旦 p<shuzi 则退出循环。这种做法使输出结果为

123456789

5、在本例中数组名shuzi是一个常量指针，永远指向shuzi[0]的地址上。

思考：如何通过p和shuzi求该数字字符串的长度

5.3.2 指针与结构

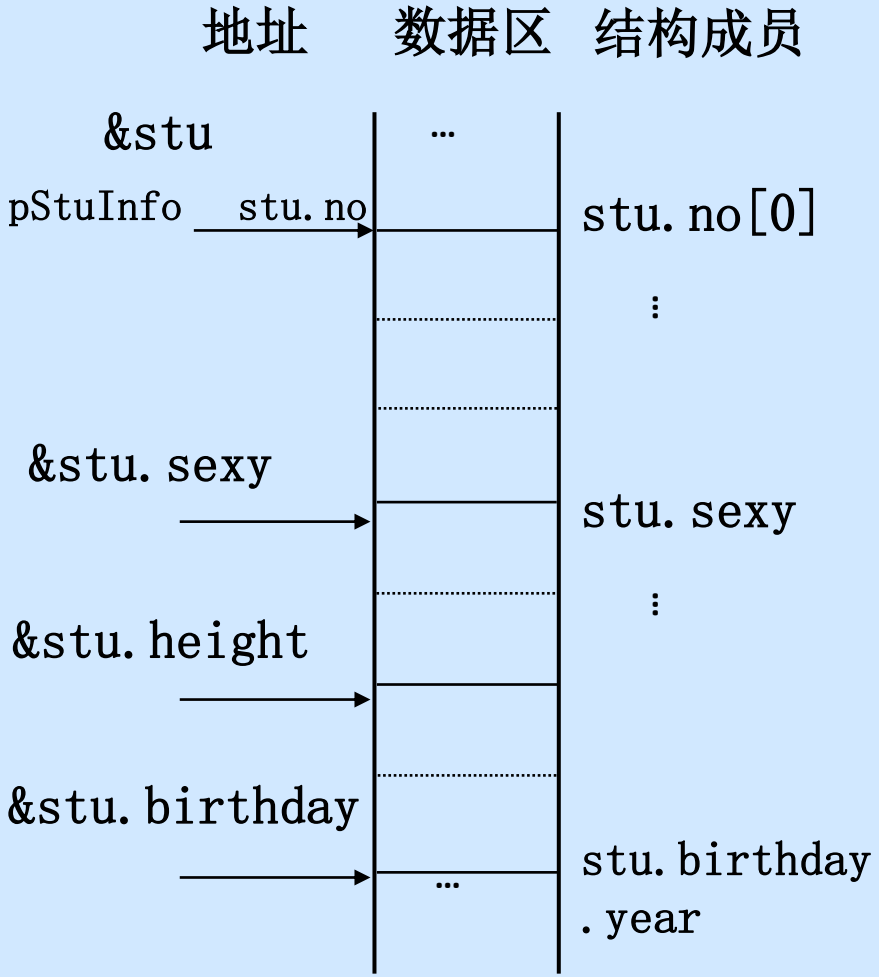


图5-3-6 指向结构的指针

```
struct StudentInfo
{
    char no[20];
    int sexy;
    double height;
    Date birthday;
};
```

```
StudentInfo stu;
```

```
StudentInfo *pStuInfo = 0;
```

```
pStuInfo = &stu;
```

引用结构变量stu的成员的三种方式：

```
stu.no = "1443011101";
```

```
(*pStuInfo).no = "1443011101";
```

```
pStuInfo -> no = "1443011101";
```

指针与结构数组的结合：通过指针存取结构数组元素及其各成员

输入学生信息

```
const int STU_NUM = 20; // 学生人数
StudentInfo stu[STU_NUM+1];
  // 定义结构数组stu，用于存储同学信息，
  // 下标i处存储第i位同学的信息，stu[0]不使用
StudentInfo *p_stuInfo = 0;
  // 指向学生信息的指针，初始化为0
int i = 0, j = 0;   // 循环变量
int changed = 0;    // 交换标志
p_stuInfo = stu+1;  //p_stuInfo指向stu[1]
for(; p_stuInfo <= stu+STU_NUM; p_stuInfo++)
{
    *p_stuInfo = readStudent();
        // 输入当前学生信息
}
```

按身高排序

```
p_stuInfo = stu; //p_stuInfo指向结构数组的开始
for(i =1; i < STU_NUM; i++)        // 扫描n-1遍
{
     changed = 0;                  // 置交换标志为，表示未交换
     for(j=1; j <= STU_NUM-i; j++)
     {                             // 每遍比较n-i次
         if((p_stuInfo[j].height) >
    (p_stuInfo[j+1].height))
         {                         // 交换stu[j]和stu[j+1]
             stu[0] = p_stuInfo[j];
             p_stuInfo[j] = p_stuInfo[j+1];
             p_stuInfo[j+1] = stu[0];
             changed = 1;          // 置交换标志为1
         }
     }
     if(changed == 0)
    {
          break;                   // 如果本遍排序中未交换，则退出循环
}
```

输出排序后学生信息

```
cout << endl <<"输入的学生信息为（按身高升序排列）："<< endl;
for(p_stuInfo++; p_stuInfo <= stu+STU_NUM; p_stuInfo++)
{
        writeStudent(*p_stuInfo);
              // 依次输出个人信息
}
```

5.4 字符串指针

可以使用指针指向字符数组或字符串常量，通过指针来使用字符数组或字符串常量。

- 指向字符串的指针变量实际上就是一个指向字符型变量的指针变量，正如同指向整型数组的指针变量就是指向一个整型变量的指针变量一样，因为一个字符串就是一个字符数组。
- 指向字符串的指针变量，它保存的是字符串中第一个字符所在内存单元的地址。

通过指针变量访问字符串常量

- C语言允许通过定义一个指向字符串常量的指针的方式来获得一片连续的内存空间，用来存放字符串和空字符。并可以使用指针访问该字符串常量。
  - 访问方法： **指针法或下标法**

如果pstr的初值为字符串的第一个字符的地址，如下定义：

char *pstr = "Hello, World"; 则有以下事实：

① **pstr + i** 就是第 **i** 个字符的地址，即它指向字符串中的第 **i** 个字符

② ***(pstr + i)** 或者 **pstr[i]** 就是它所指向的字符

③ 指针变量也可以使用下标法，即 **pstr[i]** 和***(pstr + i)** 是等价的。

注意：*pstr=*pstr+2是不允许的，因为pstr指向的是字符串常量，其值为只读，不能被改变！见例5.11（p157）

1、对字符指针变量赋值的写法

（1）char *p;
p = ″computer″;

（2） char *p=″computer″;

以上两种都行。可以整体赋值。

2、对字符数组赋初值的写法

（1）char as[12]=″department″;
// 可以。在定义时可以整体赋值
char as[] =″department″;
// 可以。在定义时可以整体赋值

（2）char as[12];
as = ″department″;
// 不可以！不可以整体赋值
as[12]=″department″;
//不可以！不可以整体赋值，数组越界

通过指针变量引用数组元素（引用数组的方法）

◆ 编程把字符串s2复制到字符串s1中

```
/* 方法1：用数组名和下标 */
#include <iostream>
using namespace std;
int main( )
{   char  s1[80] = "",  s2[ ] = "hello, world";
    int  n;
    for(n = 0;  s2[n] != '\0';  n ++)
        s1[n] = s2[n];
    s1[n] = '\0';
    cout<<"s1="<<s1<<"\ns2="<<s2<<endl;
   return 0;
}
```

通过指针变量引用数组元素：

◆ 编程把字符串s2复制到字符串s1中

```
/* 方法2：用数组名+偏移量得到元素地址，访问元素*/
#include <iostream>
using namespace std;
int main( )
{   char  s1[80] = "",  s2[ ] = "hello, world";
    int  n;
    for(n = 0;  s2[n] != '\0';  n ++)
        *(s1 + n) = *(s2 + n);
    *(s1 + n) = '\0';
    cout<<"s1="<<s1<<"\ns2="<<s2<<endl;
    return 0;
}
```

特点：

数组名本身不变
也不可能被改变

通过指针变量引用数组元素：

◆ 编程把字符串s2复制到字符串s1中

```
/* 方法3：用指针变量+偏移量得到元素地址，访问元素*/
#include <iostream>
using namespace std;
int main( )
{   char  s1[80] = "",  s2[ ] = "hello, world";
    int  n;    char  *p1 = s1,  *p2 = s2;
    for(n = 0;  *(p2 + n) != '\0';  n ++)
        *(p1 + n) = *(p2 + n);
    *(p1 + n) = '\0';
    cout<<"s1="<<s1<<"\ns2="<<s2<<endl;
    return 0;
}
```

特点：

指针变量本身的值没有变化

通过指针变量引用数组元素：

◆ 编程把字符串s2复制到字符串s1中

```
/* 方法4：用指针变量自身变化得到元素地址，访问元素*/
#include <iostream>
using namespace std;
int main( )
{   char  s1[80] = "",  s2[ ] = "hello, world";
    int  n;    char  *p1 = s1,  *p2 = s2;
    for(n = 0;  *p2 != '\0';  n ++)
        *(p1 ++) = *(p2 ++);
    *p1 = '\0';
    cout<<"s1="<<s1<<"\ns2="<<s2<<endl;
    return 0;
}
```

特点：

普通变量做循环控制变量

通过指针变量引用数组元素：

◆ 编程把字符串s2复制到字符串s1中

```
/* 方法5：用指针变量自身变化得到元素地址，访问元素 */
#include <iostream>
using namespace std;
int main( )
{   char  s1[80] = "",  s2[ ] = "hello, world";
    char  *p1 = s1,  *p2 = s2;
    for( ; *p2 != '\0'; p1 ++, p2 ++)
        *p1 = *p2;
    *p1 = '\0';
    cout<<"s1="<<s1<<"\ns2="<<s2<<endl;
    return 0;
}
```

特点：

指针变量做循环控制变量

通过指针变量访问字符串时的注意事项

注意：在利用指针变量本身的值的改变来访问字符串时，要时刻注意指针变量的当前值。

```
#include <iostream>
using namespace std;
int main( )
{   char  s1[80] = "",  s2[ ] = "hello, world";
    char  *p1 = s1,  *p2 = s2;
    for( ; *p2 != '\0'; p1 ++, p2 ++)
        *p1 = *p2;
    *p1 = '\0';
    cout<<"s1="<<s1<<"\ns2="<<s2<<endl;
    return 0;
}
```

此处用p1和p2来输出字符串，是得不到正确结果的。

指向字符串的指针变量的有关运算

如有： char str[80], *ps;

ps = str; 或 ps = &str[0];

① ps ++ 或 ps += 1，是下一个字符的地址（即 ps 指向下一个字符）

ps -- 或 ps -= 1，是上一个字符的地址（即 ps 指向上一个字符）

② * ps ++ 等价于 *(ps ++)，即先得到 ps 当前指向的字符，然后再使 ps 自增，从而指向下一个字符

③ * ++ ps 等价于 *(++ ps)，即先使 ps 自增，指向下一个字符，然后得到 ps 所指向的字符

④ (* ps) ++，则是表示 ps 当前指向的字符加1

课堂练习

以下程序错误原因是：

```
void main()
{
    int  *p, i;
    char *q, ch;
    p=&i;
    q=&ch;
    *p=40;
    *p=*q;
    …  …
}
```

A. p 和 q 的类型不一致，不能执行 *p=*q; 语句

B. *p 中存放的是地址，因此不能执行 *p=40; 语句

C. q 没有指向具体的存储单元，所以*q 没有实际意义

D. q 虽然指向了具体的存储单元，但该单元中没有确定的值，所以不能执行 *p = *q; 语句

课堂练习

若有语句 char str[80], *pstr; 则下列语句中错误的是(　　)。

(A) pstr = "Hello";

(B) *pstr = "World";

(C) pstr = str;　　(D) *str = 'Y';

(E) str = "";　　(F) pstr = 0;

答案：(B) (E)

课堂练习

若有语句 char *str = “goodjob”; 则（　　）不是对字符串中字符的正确引用。(其中6≥n≥0)

(A) * str　　(B) *(str + n)

(C) str[n]　　(D) str

答案：(D)

int *p, *q; // 定义p,q为指向整型变量的指针

float *point; // point为指向float型变量的指针

char *p1=0; //p1位指向字符型变量的指针

double *pd; // 定义pd为指向double型变量的指针

StudentInfo *pStu; // 定义指向结构的指针pStu

以上方式定义的指针可以指向变量或一维数组。下面讨论以下两种指针：

int (*pa)[10];// 定义pa为指向int型数组的指针

int **qq; // 定义qq为指向int型指针的指针

5.5 指针数组

1. 指针数组的概念

指针数组也是数组，但它的各数组元素都是指针，且必须是指向同一种数据类型的指针。

2. 指针数组的定义和初始化

假设存在3个数组，定义如下

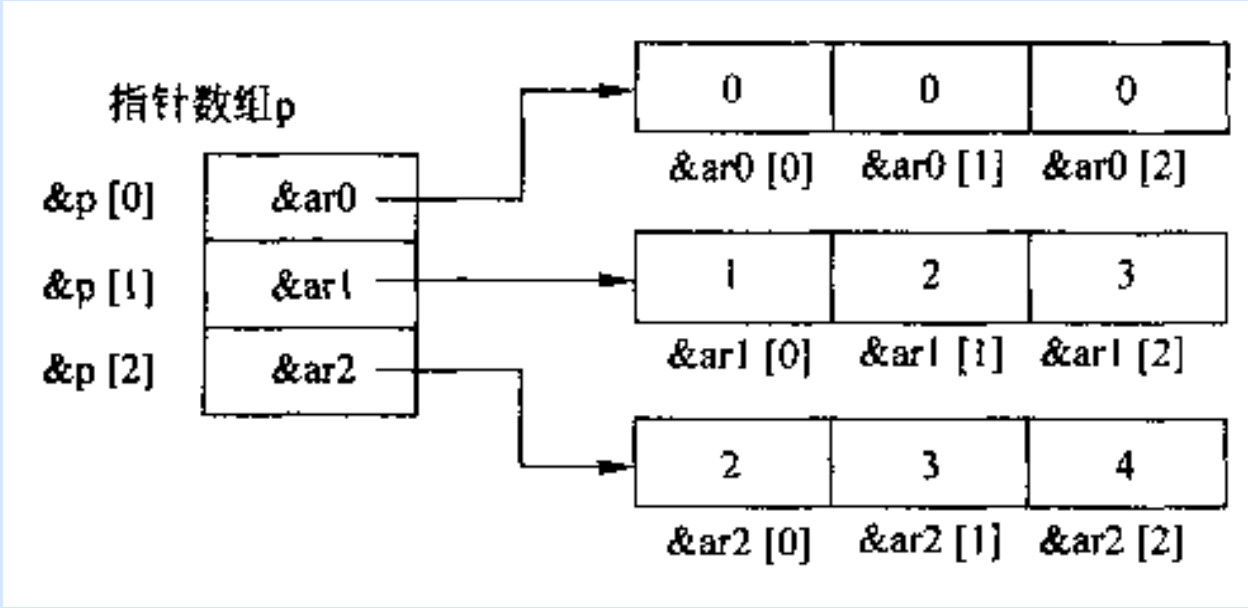
```
int ar0[]={0,1,2};
int ar1[]={1,2,3};
int ar2[]={2,3,4};
```

下面的语句定义一个指针数组p，并初始化

```
int *p[]={ar0,ar1,ar2};
```

((p+2)+2)指向元素，即p[2][2].

- 这条语句定义p是int *类型的数组，其长度由初始化的项决定，此处为3。这个数组的每个元素都是一个指针变量，分别指向数组ar0,ar1和ar2的首地址。



- 下面给出一个程序。希望在机器上运行这个程序，重点理解：

(1) 如何输出一个变量的地址

(2) 上图中p与数组ar0,ar1,ar2间的指向关系

(3) p[0][0]与p[0][1]的含义

```
数组名就是该数组的符号地址，是数组元素的首地址:
ar0=    0038F900
&ar0=   0038F900
&ar0[0]=0038F900
```

```
#include <iostream>
using namespace std;
int main()
{
    int ar0[]={0,1,2}; // 定义整型数组ar0，并初始化
    int ar1[]={3,4,5}; // 定义整型数组ar1，并初始化
    int ar2[]={6,7,8}; // 定义整型数组ar2，并初始化
    int *p[]={ar0,ar1,ar2}; // 定义指针数组并初始化
    cout<<"数组名就是该数组的符号地址，是数组元素的首地址："<<endl;
    cout<<"ar0=\t"<<ar0<<endl;     // 输出ar0
    cout<<"&ar0=\t"<<&ar0<<endl;
                                   // 输出数组ar0的首地址
    cout<<"&ar0[0]="<<&ar0[0]<<endl;
                                   // 输出数组ar0的元素首地址
```

```
cout<<"p=\t"<<p<<endl;          // 输出p
cout<<"&p=\t"<<&p<<endl;
                    // 输出数组p的地址
cout<<"&p[0]=\t"<<&p[0]<<endl;
                    // 输出数组p的元素首地址
cout<<"指针数组本身是一个数组，它的元素是连续存放的："<<endl;
cout<<"&p[0]=\t"<<&p[0]<<endl;
                    // 输出p[0]的地址
cout<<"&p[1]=\t"<<&p[1]<<endl;
                    // 输出p[1]的地址
cout<<"&p[2]=\t"<<&p[2]<<endl;
                    // 输出p[2]的地址
```

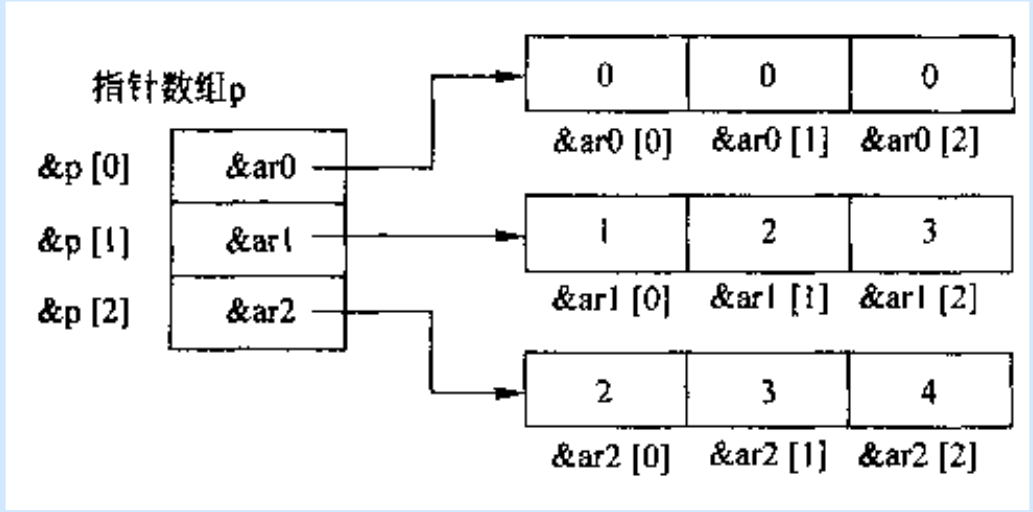
```
p=          0038F8C4
&p=         0038F8C4
&p[0]=      0038F8C4
```

```
指针数组本身是一个数组，它的元素是连续存放的：
&p[0]=      0038F8C4
&p[1]=      0038F8C8
&p[2]=      0038F8CC
```

```
cout<<"指针数组的每一个元素是一个指针："<<endl;
cout<<"p[0]=\t"<<p[0]<<endl;
                                    // 输出p[0]的内容，是一个指针
cout<<"ar0=\t"<<ar0<<endl;
                                          // 输出ar0的地址
cout<<"p[1]=\t"<<p[1]<<endl;
cout<<"ar1=\t"<<ar1<<endl;
cout<<"p[2]=\t"<<p[2]<<endl;
cout<<"ar2=\t"<<ar2<<endl;
```

```
指针数组的每一个元素是一个指针：
p[0]=    0038F900
ar0=     0038F900
p[1]=    0038F8EC
ar1=     0038F8EC
p[2]=    0038F8D8
ar2=     0038F8D8
```

```
    cout<<"可利用指针数组的元素来访问所指向的数组的元素，就像用二维数组：\n";
    cout<<"*p[0]=\t"<<*p[0]<<endl;
                    // 输出p[0]所指的内容
    cout<<"*ar0=\t"<<*ar0<<endl;
                    // 输出ar0地址中的内容
    cout<<"&p[0][0]= "<<&p[0][0]<<endl;
    cout<<"&ar0[0] = "<<&ar0[0]<<endl;
    cout<<"p[0][1] = "<<p[0][1]<<endl;
    cout<<"ar0[1]  = "<<ar0[1]<<endl;
    return 0;
}
```




```
可利用指针数组的元素来访问所指向的数组的元素，就像用二维数组：
*p[0]=   0
*ar0=    0
&p[0][0]= 0038F900
&ar0[0] = 0038F900
p[0][1] = 1
ar0[1]  = 1
```

5.6 指向指针的指针

- 指向指针的指针：双重指针变量，其值为一个指针变量的内存存放地址

```
int a = 123;
int *p_a = &a;
        //p_a 指向变量a
int **p = &p_a;
        //p 指向指针 p_a
```

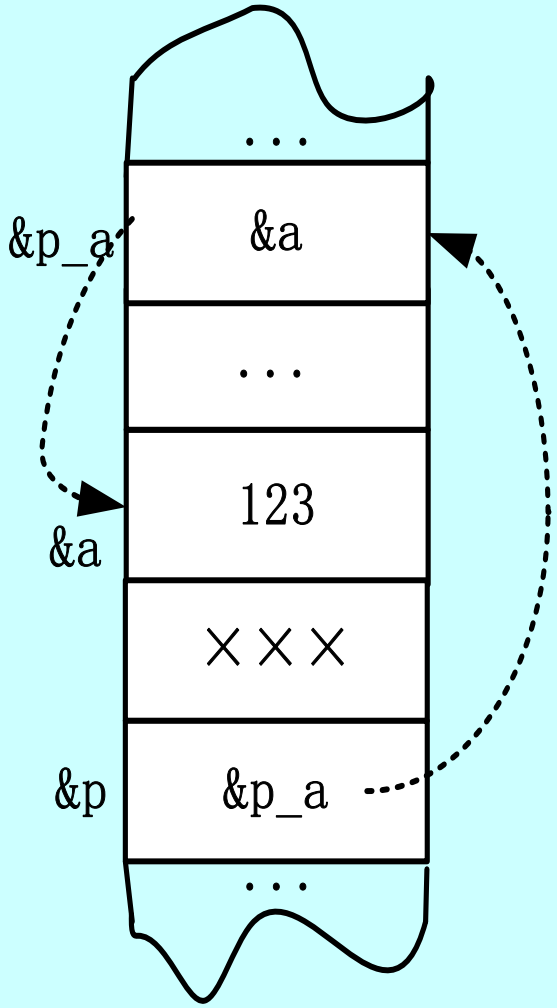


图5-6-1 双重指针p

二维数组的数组名的含义

- 二维数组的名称也是一个首地址，指向二维数组的第一个元素。
- 二维数组是一个特殊的一维数组。该一维数组的每一个元素是一个指针（指向另外的一维数组），
- 实际上，二维数组就是一个指针数组
- 二维数组的数组名可以看成是指向指针的指针。

- char name[3][20] = {"China", "America", "German"};
- 数组名：name(双重指针)
- 元素：name[0], name[1], name[2] (指针)
- 元素：name[0][0], name[1][1] (字符)

name→	name[0]→	C	h	i	n	a	\0	\0	\0
	name[1]→	A	m	e	r	i	c	a	\0
	name[2]→	G	e	r	m	a	n	\0	\0

本章总结

指向变量和一维数组的指针

指向结构数组的指针

指向指针的指针以及指针数组

平时作业：视频学习内容

视频自学内容

9.2（字符串指针） 10.2（习题） 11.2（递推）
12.1（递归）14.1（程序举例）