西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

大批量数据的简洁表示

1、数组指针

2、指针数组

7.3.3 数组指针

- 前面的指针变量指向的是数组元素，故定义指针变量时其指向类型与数组元素的数据类型相同，C语言可以定义一个指针变量，其指向类型是一个数组（一维或多维），称为数组指针，定义形式为：

①指向一维数组的指针变量定义

```
指向类型 (*指针变量名)[常量表达式],.....
```

②指向多维数组的指针变量定义

```
指向类型 (*指针变量名)[常量表达式1][常量表达式2],.....
```

7.3.3 数组指针

- 注意指针变量名必须括起来，使得（*）比（[ ]）先处理，说明定义是一个指针。否则因为（[ ]）会比（*）优先级高，变成定义数组了。

7.3.3 数组指针

- 上述语法的含义是：

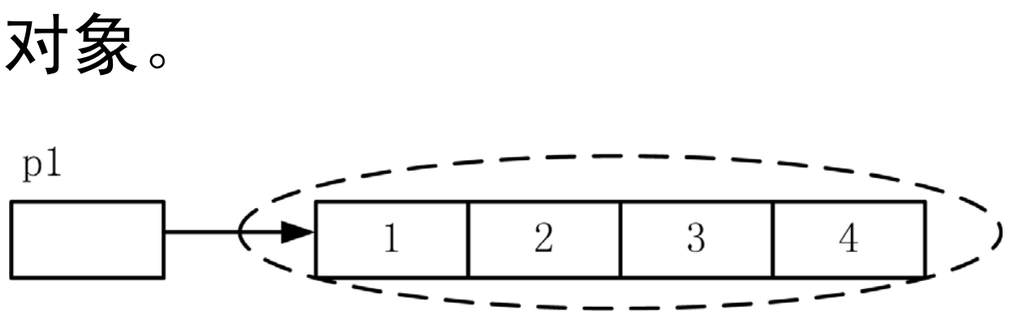
①定义一个指针变量，它指向如下形式的一维数组
　　元素类型　数组名[常量表达式]
②定义一个指针变量，它指向如下形式的多维数组
　　元素类型　数组名[常量表达式1][常量表达式2]...

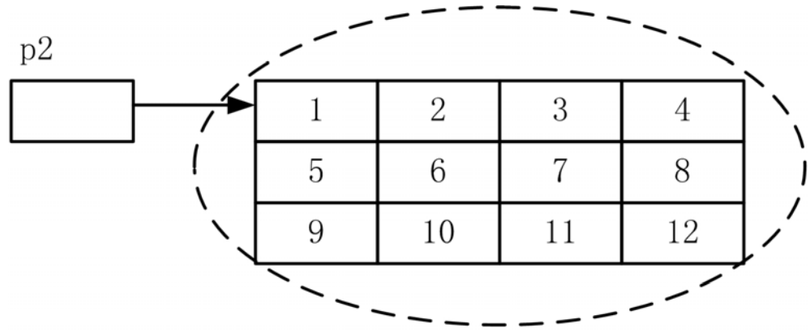
7.3.3 数组指针

- 例如：

```
int (*p1)[4]; //定义指向一维数组的指针变量p1
int (*p2)[3][4]; //定义指向二维数组的指针变量p2
```

- 指针变量p1的指向类型是一个有4个整型元素的一维数组，指针变量p2的指向类型是一个有12个整型元素的二维数组（3行4列），即虚线对应的内存单元整体是数组指针的指向对象。





7.3.3 数组指针

- 数组指针本质上是一个指针，编译器像处理其它指针变量一样为数组指针变量分配4个字节的存储空间，而不是按数组长度来分配。
- 数组指针的实际意义是若p指向一个数组，则*p就是该数组。假设

```
int a[3][4], (*p)[4], j=1;
p=a; //*p是a[0]，及 int[4]的数组
```

- 则*p就是数组a[0]，而不是数组a[0]的元素。

7.3.3 数组指针

- 通过p访问数组元素a[0][j]的写法是

```
(*p)[j]=10; //等价于a[0][j]=10
*(*p+j)=10; //等价于*(a[0]+j)=10;
```

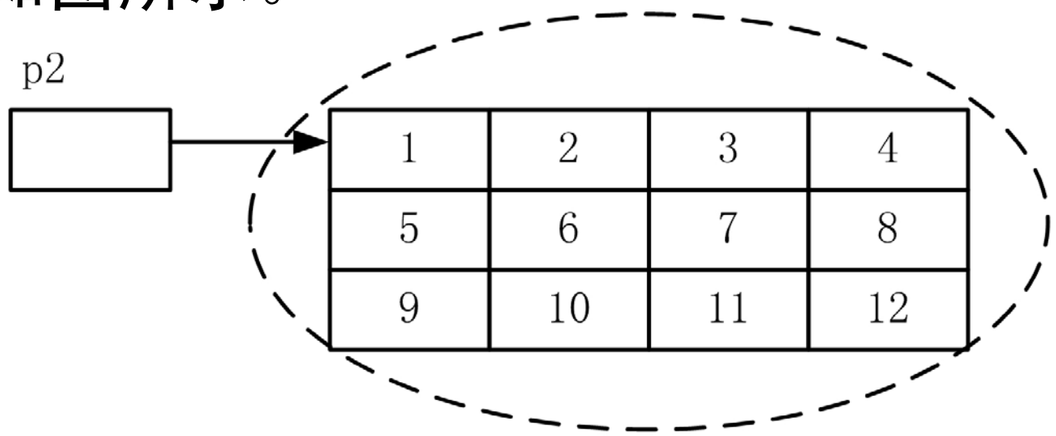
- 注意*p必须包含括号，因为（[ ]）的优先级比（*）高。

7.3.3 数组指针

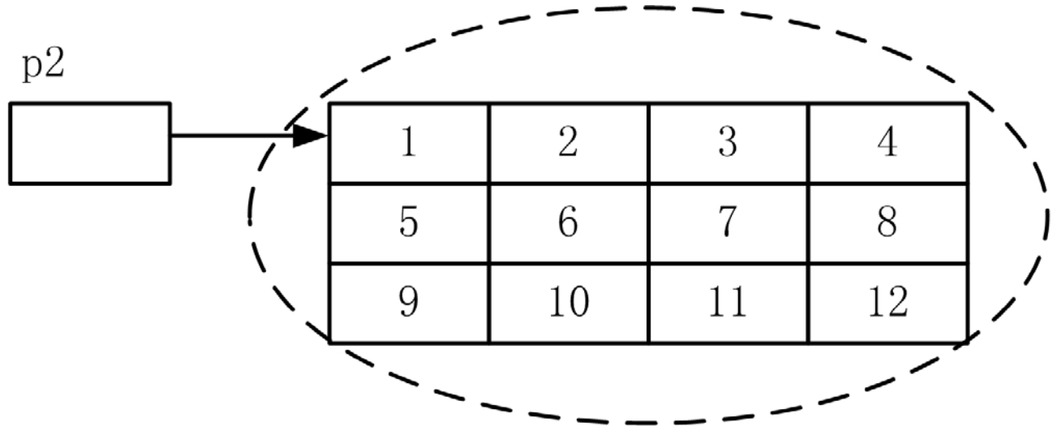
- 假设

```
int a[3][4], (*p2)[4];
p2=a;
```

- 因为p2指向一个有4个整型元素的一维数组。当p2=a时，p2指向二维数组a的第0行，如图所示。



7.3.3 数组指针



p2+1指向下一行，即指向二维数组a的第1行，p2+i指向二维数组a的第i行，那么p2+i与a+i是等价的。显而易见，*(p2+i)与*(a+i)、p2[i]与a[i]是等价的，它们均表示二维数组的第i行，而要表示元素a[i][j]可以用p[i][j]、*(a+i)[j]、*(p+i)[j]、*(*(a+i)+j)、*(*(p+i)+j)形式之一。

7.3.3 数组指针

- 实际编程中，引入数组指针可以将一个数组当作“元素”来处理，能够简化多维数组的处理。

7.3.3 数组指针

【例7.11】

通过指向一维数组的指针变量遍历二维数组元素。

7.3.3 数组指针

例7.11

```
1 #include <stdio.h>
2 int main()
3 {
4   int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12},i,j;
5   int (*p)[4]=a;
6   for (i=0; i<3; i++) { //行
7     for (j=0; j<4; j++) printf("%2d ",p[i][j]);
8     printf("\n"); //每行末尾输出换行
9   }
10  return 0;
11 }
```

7.3.3 数组指针

例7.11

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

程序运行屏幕

```
1  2  3  4
5  6  7  8
9 10 11 12
```

7.3.4 指针数组

- 一个数组，若其元素为指针类型，称为指针数组，其定义的一般形式为：

①一维指针数组的定义
　指向类型 *数组名[常量表达式],.....
②多维指针数组的定义
　指向类型 *数组名[常量表达式1][常量表达式2]...[常量表达式n],.....

7.3.4 指针数组

- 例如：

```
int *p[4]; //一维指针数组
int *s[3][4]; //二维指针数组
```

- 其中p是一个一维数组，有4个元素，每个元素都是一个指向整型的指针类型；s是一个二维数组，有12个元素，每个元素都是一个指向整型的指针类型。

7.3.4 指针数组

- 注意指针数组“int *p[4];”与“int (*p)[4];”写法的区别，后者是数组指针。

7.3.4 指针数组

- 实际编程中，使用指针数组可以方便地处理大批量指针数据，例如若干字符串、多个存储块的处理等。

7.3.4 指针数组

- 指针数组的初始化实质就是数组的初始化，例如：

```
int a[4][4]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
//二维数组
int *s[4]={a[0],a[1],a[2],a[3]}; //一维指针数组初始化
```

- 初始化后指针数组s的元素指向了二维数组各行的首元素。

7.3.4 指针数组

- 若指针数组未初始化，则它的每个元素都是一个“野指针”。
  特别的，下面的代码将指针数组元素均初始化为空指针：

```
int *s[4]={NULL,NULL,NULL,NULL}; //一维指针数组初始化
```

7.3.4 指针数组

- 指针数组的每个元素既可以按数组方式来访问，又可以用指针的方式来访问。下面是通过指针数组s的元素访问二维数组a的典型形式：
- （1）s[0]：指向a[0][0]；
- （2）*s[0]：等价于a[0][0]；
- （3）s[i]+j：指向a[i][j]；
- （4）*(s[i]+j)、*(*(s+i)+j)、s[i][j]：等价于a[i][j]。

7.3.4 指针数组

- 由于s是数组名，因此s不是左值，不能做自增自减运算等。例如：

```
s=a[0]; //错误，s不能作为左值
s[0]=a[0]; //正确，s[0]元素是变量可以作为左值
s[0]=a+1;//错误，指向类型不相同s[0]是int *，a+1是行指针（一维数组指针）
s[0]=a; //错误，指向类型不相同 s[0]是int *，a是行指针（一维数组指针）
s=s+1; //错误，s不能作为左值
s++; //错误，s不能做自增运算
```

7.3.4 指针数组

【例7.12】

通过一维指针数组遍历二维数组元素。

7.3.4 指针数组

例7.12

```
1 #include <stdio.h>
2 int main()
3 {
4   int a[4][4]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
5   int i, j, *s[4]={a[0],a[1],a[2],a[3]}; //一维指针数组初始化
6   for (i=0; i<4 ; i++) {
7     for (j=0; j<4; j++)
8       printf("%2d ",s[i][j]); //s[i][j]等价于a[i][j]
9     printf("\n");
10  }
11  return 0;
12 }
```

7.3.4 指针数组

例7.12

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

程序运行屏幕

```
 1   2   3   4
 5   6   7   8
 9  10  11  12
13  14  15  16
```


程序设计