

程序设计基础

第 11 章: 指针进阶

刘新国

浙江大学计算机学院
浙江大学 CAD&CG 国家重点实验室

December 22, 2021

指针进阶

- 指针数组

- 二级指针

- 数组指针

- 指针数组和数组指针区别

- 指针返回值

指针进阶

指针数组

二级指针

数组指针

指针数组和数组指针区别

指针返回值

指针进阶

指针数组

二级指针

数组指针

指针数组和数组指针区别

指针返回值

例 11-1, 查找字符串

```
#include<stdio.h>
int main(void)
{
    int i; char ch;
    char * colors[5] = {"red", "blue",
        "yellow", "green", "black"};
    printf("输入单词首字母: ");
    ch = getchar();
    for( i = 0; i<5; i++ )
        if( *colors[i]==ch )
            break;
    if( i<5 ) puts(colors[i]);
    else
        printf("没有找到以%c开头的字符串\n",
            ch);
    return 0;
}
```

例 11-1, 查找字符串

```
#include<stdio.h>
int main(void)
{
    int i; char ch;
    char * colors[5] = {"red", "blue",
        "yellow", "green", "black"};
    printf("输入单词首字母: ");
    ch = getchar();
    for( i = 0; i<5; i++ )
        if( *colors[i]==ch )
            break;
    if( i<5 ) puts(colors[i]);
    else
        printf("没有找到以%c开头的字符串\n",
            ch);
    return 0;
}
```

定义 colors 为指针数组, 里面有 5 个指针

例 11-1, 查找字符串

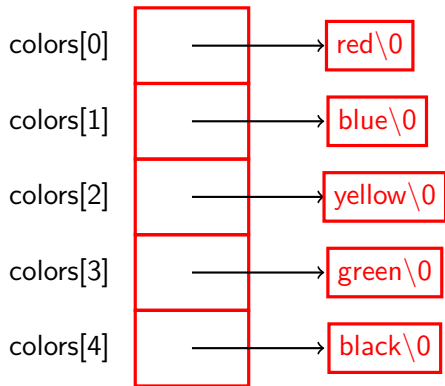
```
#include <stdio.h>
int main(void)
{
    int i; char ch;
    char * colors[5] = {"red", "blue",
                        "yellow", "green", "black"};
    printf("输入单词首字母: ");
    ch = getchar();
    for( i = 0; i<5; i++ )
        if( *colors[i] == ch )
            break;
    if( i<5 ) puts(colors[i]);
    else
        printf("没有找到以%c开头的字符串\n",
               ch);
    return 0;
}
```

定义 colors 为指针数组, 里面有 5 个指针

每个指针指向一个字符串

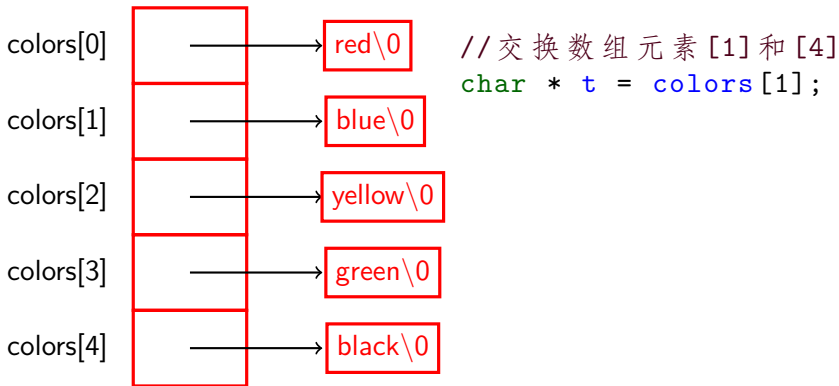
指针数组的优点：便于交换数据

```
char * colors[5] = {"red", "blue",  
                    "yellow", "green", "black"};
```



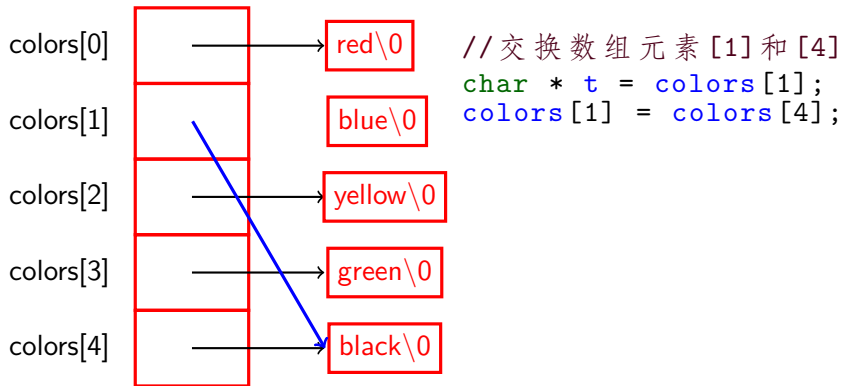
指针数组的优点：便于交换数据

```
char * colors[5] = {"red", "blue",  
                    "yellow", "green", "black"};
```



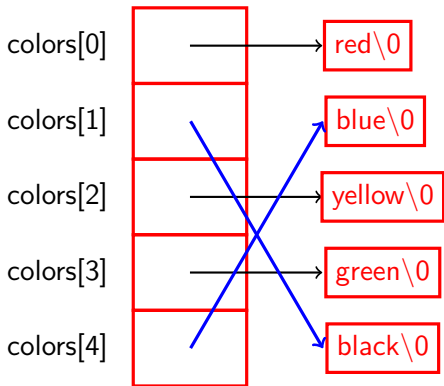
指针数组的优点：便于交换数据

```
char * colors[5] = {"red", "blue",  
                    "yellow", "green", "black"};
```



指针数组的优点：便于交换数据

```
char * colors[5] = {"red", "blue",  
                    "yellow", "green", "black"};
```

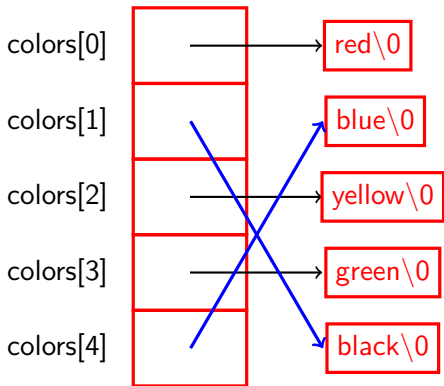


// 交换数组元素[1]和[4]

```
char * t = colors[1];  
colors[1] = colors[4];  
colors[4] = t;
```

指针数组的优点：便于交换数据

```
char * colors[5] = {"red", "blue",  
                    "yellow", "green", "black"};
```



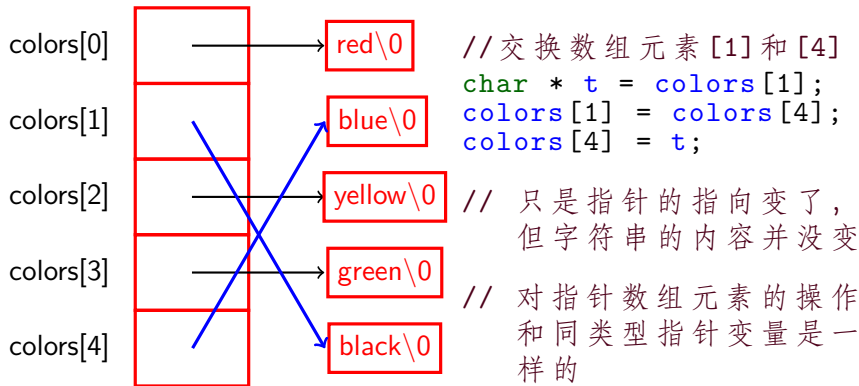
// 交换数组元素[1]和[4]

```
char * t = colors[1];  
colors[1] = colors[4];  
colors[4] = t;
```

// 只是指针的指向变了，
但字符串的内容并没变

指针数组的优点：便于交换数据

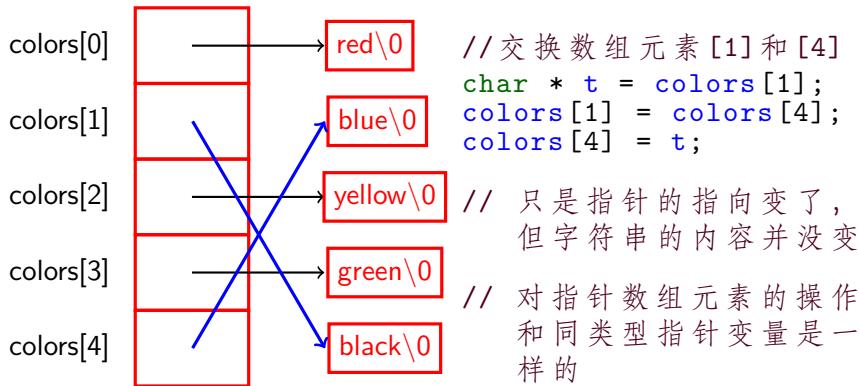
```
char * colors[5] = {"red", "blue",  
                    "yellow", "green", "black"};
```



► 指针数组的优点：便于交换数据

指针数组的优点：便于交换数据

```
char * colors[5] = {"red", "blue",  
                    "yellow", "green", "black"};
```



- ▶ 指针数组的优点：便于交换数据
- ▶ 不需要赋值整个数组的内容

指针进阶

指针数组

二级指针

数组指针

指针数组和数组指针区别

指针返回值

二级指针 – 指向指针的指针

指针回顾

二级指针 – 指向指针的指针

指针回顾

- ▶ 指针是一种数据类型，存储变量的地址

二级指针 – 指向指针的指针

指针回顾

- ▶ 指针是一种数据类型，存储变量的地址
- ▶ 指针指向某一个变量后，可以通过指针访问改变量（取值、赋值）

二级指针 – 指向指针的指针

指针回顾

- ▶ 指针是一种数据类型，存储变量的地址
- ▶ 指针指向某一个变量后，可以通过指针访问改变量（取值、赋值）

```
int a, *p, b;
```

```
p = &a;           // 让指针p指向变量a
```

```
*p = 5;           // 通过指针p，给变量a赋值
```

```
b = *p + 5;       // 通过指针p，使用变量a进行运算
```

二级指针 – 指向指针的指针

指针回顾

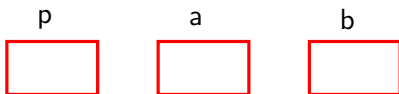
- ▶ 指针是一种数据类型，存储变量的地址
- ▶ 指针指向某一个变量后，可以通过指针访问改变量（取值、赋值）

```
int a, *p, b;
```

```
p = &a;           // 让指针p指向变量a
```

```
*p = 5;           // 通过指针p，给变量a赋值
```

```
b = *p + 5;       // 通过指针p，使用变量a进行运算
```



二级指针 – 指向指针的指针

指针回顾

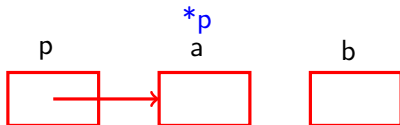
- ▶ 指针是一种数据类型，存储变量的地址
- ▶ 指针指向某一个变量后，可以通过指针访问改变量（取值、赋值）

```
int a, *p, b;
```

```
p = &a; // 让指针p指向变量a
```

```
*p = 5; // 通过指针p，给变量a赋值
```

```
b = *p + 5; // 通过指针p，使用变量a进行运算
```



二级指针 – 指向指针的指针

指针回顾

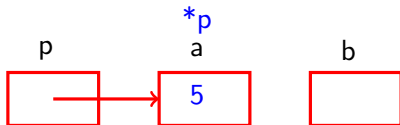
- ▶ 指针是一种数据类型，存储变量的地址
- ▶ 指针指向某一个变量后，可以通过指针访问改变量（取值、赋值）

```
int a, *p, b;
```

```
p = &a;           // 让指针p指向变量a
```

```
*p = 5;           // 通过指针p，给变量a赋值
```

```
b = *p + 5;       // 通过指针p，使用变量a进行运算
```



二级指针 – 指向指针的指针

指针回顾

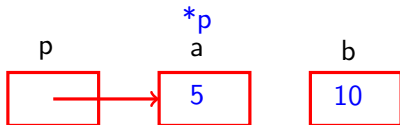
- ▶ 指针是一种数据类型，存储变量的地址
- ▶ 指针指向某一个变量后，可以通过指针访问改变量（取值、赋值）

```
int a, *p, b;
```

```
p = &a;           // 让指针p指向变量a
```

```
*p = 5;           // 通过指针p，给变量a赋值
```

```
b = *p + 5;       // 通过指针p，使用变量a进行运算
```



二级指针 – 指向指针的指针

- ▶ 如果变量本身也是指针呢？

二级指针 – 指向指针的指针

- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“指针的指针”

二级指针 – 指向指针的指针

- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“**指针的指针**”
也称为“**二级指针**”

二级指针 – 指向指针的指针

- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“**指针的指针**”
也称为“**二级指针**”

```
int a, *p, **pp;
```

```
pp = &p; // 二级指针pp指向指针p(*pp就是p了)
```

```
*pp = &a; // 通过二级指针pp，让指针p指向a。
```

等价于 `p = &a;`

二级指针 – 指向指针的指针

- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“**指针的指针**”
也称为“**二级指针**”

```
int a, *p, **pp;
```

```
pp = &p; // 二级指针pp指向指针p(*pp就是p了)
```

```
*pp = &a; // 通过二级指针pp，让指针p指向a。
```

等价于 `p = &a;`



二级指针 – 指向指针的指针

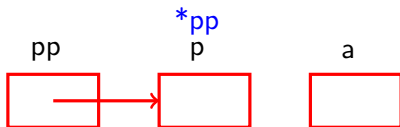
- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“**指针的指针**”
也称为“**二级指针**”

```
int a, *p, **pp;
```

```
pp = &p; // 二级指针pp指向指针p(*pp就是p了)
```

```
*pp = &a; // 通过二级指针pp，让指针p指向a。
```

等价于 `p = &a;`



二级指针 – 指向指针的指针

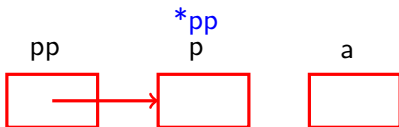
- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“**指针的指针**”
也称为“**二级指针**”

```
int a, *p, **pp;
```

```
pp = &p; // 二级指针pp指向指针p(*pp就是p了)
```

```
*pp = &a; // 通过二级指针pp，让指针p指向a。
```

等价于 `p = &a;`



二级指针 – 指向指针的指针

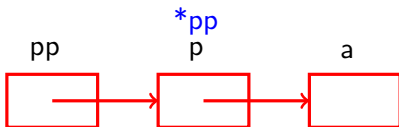
- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“**指针的指针**”
也称为“**二级指针**”

```
int a, *p, **pp;
```

```
pp = &p; // 二级指针pp指向指针p(*pp就是p了)
```

```
*pp = &a; // 通过二级指针pp，让指针p指向a。
```

等价于 p = &a;



二级指针 – 指向指针的指针

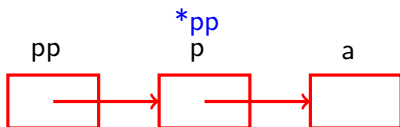
- ▶ 如果变量本身也是指针呢？
- ▶ 那么，指向这种变量的指针是“**指针的指针**”
也称为“**二级指针**”

```
int a, *p, **pp;
```

```
pp = &p; // 二级指针pp指向指针p(*pp就是p了)
```

```
*pp = &a; // 通过二级指针pp，让指针p指向a。
```

等价于 `p = &a;`



`a = 5;` // 下面的语句也都是给给变量a赋值，等价的

```
*p = 5;
```

```
**p = 5;
```


二级指针示例：例 11-3，查找字符串

```
#include<stdio.h>
int main(void)
{
    int i, flag = 0;
    char ch;
    char *colors[5] = { "red", "blue", "yellow",
                        "green", "black"};
    char **pc;
    printf("输入颜色首字母：");
    ch = getchar();
    for(pc = colors, i = 0; i<5; i++ )
        if( **(pc+i)==ch ) { flag = 1; break; }
    if( flag ) puts( *(pc+i) );
    else printf("没找到%c开头的颜色\n", ch);
    return 0;
}
```

二级指针示例：例 11-3，查找字符串

```
#include<stdio.h>
int main(void)
{
    int i, flag = 0;
    char ch;
    char *colors[5] = { "red", "blue", "yellow",
                        "green", "black"};
    char **pc;
    printf("输入颜色首字母: ");
    ch = getchar();
    for(pc = colors, i = 0; i<5; i++ )
        if( *(pc+i)==ch ) { flag = 1; break; }
    if( flag ) puts( *(pc+i) );
    else printf("没找到%c开头的颜色\n", ch);
    return 0;
}
```

定义 colors 为指针数组，里面有 5 个指针

二级指针示例：例 11-3，查找字符串

```
#include<stdio.h>
int main(void)
{
    int i, flag = 0;
    char ch;
    char *colors[5] = { "red", "blue", "yellow",
                        "green", "black"};
    char **pc;
    printf("输入颜色首字母: ");
    ch = getchar();
    for(pc = colors, i = 0; i<5; i++ )
        if( *(pc+i)==ch ) { flag = 1; break; }
    if( flag ) puts( *(pc+i) );
    else printf("没找到%c开头的颜色\n", ch);
    return 0;
}
```

二级指针变量

二级指针示例：例 11-3，查找字符串

```
#include<stdio.h>
int main(void)
{
    int i, flag = 0;
    char ch;
    char *colors[5] = { "red", "blue", "yellow",
                        "green", "black"};
    char **pc;
    printf("输入颜色首字母: ");
    ch = getchar();
    for( pc = colors, i = 0; i<5; i++ )
        if( *(pc+i)==ch ) { flag = 1; break; }
    if( flag ) puts( *(pc+i) );
    else printf("没找到%c开头的颜色\n", ch);
    return 0;
}
```

指向指针数组的首元素

二级指针示例：例 11-3，查找字符串

```
#include<stdio.h>
int main(void)
{
    int i, flag = 0;
    char ch;
    char *colors[5] = { "red", "blue", "yellow",
                        "green", "black"};
    char **pc;
    printf("输入颜色首字母: ");
    ch = getchar();
    for(pc = colors, i = 0; i<5; i++ )
        if( ** (pc+i) == ch ) { flag = 1; break; }
    if( flag ) puts( *(pc+i) );
    else printf("没找到%c开头的颜色\n", ch);
    return 0;
}
```

等价于 $*(pc+i)$ ，等价于 $*pc[i]$
即：颜色字符串 colors[i] 首字符

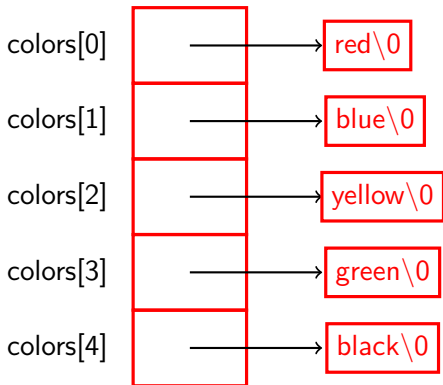
二级指针示例：例 11-3，查找字符串

```
#include<stdio.h>
int main(void)
{
    int i, flag = 0;
    char ch;
    char *colors[5] = { "red", "blue", "yellow",
        "green", "black"};
    char **pc;
    printf("输入颜色首字母: ");
    ch = getchar();
    for(pc = colors, i = 0; i<5; i++ )
        if( *(pc+i)==ch ) { flag = 1; break; }
    if( flag ) puts( *(pc+i) );
    else printf("没找到%c开头的颜色\n", ch);
    return 0;
}
```

等价于 pc[i]，也是颜色字符串 colors[i]

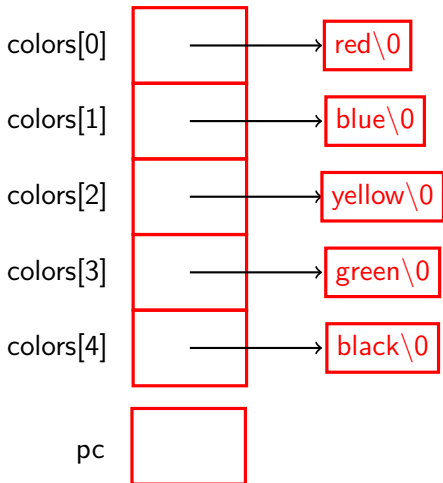
二级指针示例：例 11-3，查找字符串

```
char *colors[5] = { "red", "blue", "yellow",  
                    "green", "black"};  
char **pc;
```



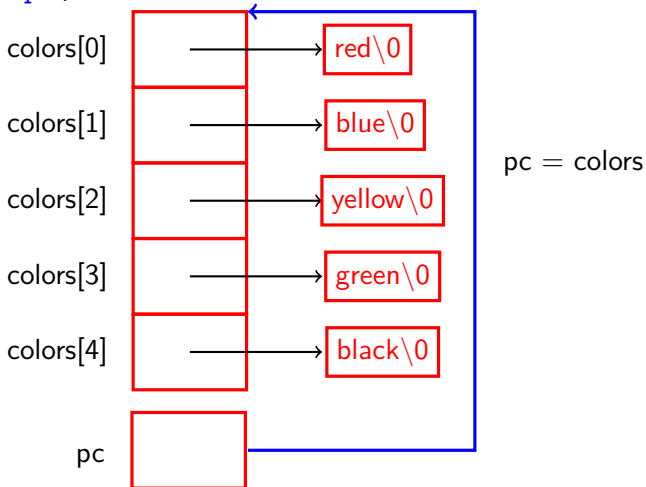
二级指针示例：例 11-3，查找字符串

```
char *colors[5] = { "red", "blue", "yellow",  
                    "green", "black"};  
char **pc;
```



二级指针示例：例 11-3，查找字符串

```
char *colors[5] = { "red", "blue", "yellow",  
                    "green", "black"};  
char **pc;
```



指针进阶

指针数组

二级指针

数组指针

指针数组和数组指针区别

指针返回值

数组指针 – 指向数组的指针

一维、二维数组回顾

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

▶ 二维数组可以认为是数组的数组

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

- ▶ 二维数组可以认为是数组的数组
- ▶ 例如：b 是一个长度为 5 的数组，它的元素是长度为 10 的数组。因此，

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

- ▶ 二维数组可以认为是数组的数组
- ▶ 例如：b 是一个长度为 5 的数组，它的元素是长度为 10 的数组。因此，
 - ▶ b[k] 是一个长度为 10 的数组

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

- ▶ 二维数组可以认为是数组的数组
- ▶ 例如：b 是一个长度为 5 的数组，它的元素是长度为 10 的数组。因此，
 - ▶ b[k] 是一个长度为 10 的数组
 - ▶ 我们知道 b+k 是一个指针，指向 b 的第 k 个元素

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];      // 长度为10的数组  
int b[5][10];  // 5行、10列
```

- ▶ **二维数组**可以认为是**数组的数组**
- ▶ 例如：b 是一个长度为 5 的数组，它的元素是**长度为 10 的数组**。因此，
 - ▶ $b[k]$ 是一个长度为 10 的数组
 - ▶ 我们知道 $b+k$ 是一个指针，指向 b 的第 k 个元素
 - ▶ $b+k$ 是一个**数组指针**（指向数组 $b[k]$ 的指针）

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

b + 0										b[0]
b + 1										b[1]
b + 2										b[2]
b + 3										b[3]
b + 4										b[4]

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

b + 0										b[0]
b + 1										b[1]
b + 2										b[2]
b + 3										b[3]
b + 4										b[4]

► 二维数组名 **b** 是指向数组 **b[0]** 的指针，数组指针

数组指针 – 指向数组的指针

一维、二维数组回顾

```
int a[10];    // 长度为10的数组  
int b[5][10]; // 5行、10列
```

b + 0										b[0]
b + 1										b[1]
b + 2										b[2]
b + 3										b[3]
b + 4										b[4]

- ▶ 二维数组名 `b` 是指向数组 `b[0]` 的指针，数组指针
- ▶ `b+k` 是数组指针，指向数组 `b[k]` 的指针

数组指针 – 指向数组的指针

数组指针的定义

数组元素类型名 (*指针名)[数组长度名];

数组指针 – 指向数组的指针

数组指针的定义

数组元素类型名 (*指针名)[数组长度名];

```
int (*p)[10];
```

数组指针 – 指向数组的指针

数组指针的定义

数组元素类型名 (*指针名)[数组长度名];

```
int (*p)[10];
```

- ▶ 定义了一个指针 (只占 4 个 byte), p

数组指针 – 指向数组的指针

数组指针的定义

数组元素类型名 (*指针名)[数组长度名];

```
int (*p)[10];
```

- ▶ 定义了一个指针 (只占 4 个 byte), p
- ▶ 该指针 p 可以用于指向长度为 10 的 int 数组

数组指针 – 指向数组的指针

数组指针的定义

数组元素类型名 (*指针名)[数组长度名];

```
int (*p)[10];
```

- ▶ 定义了一个指针 (只占 4 个 byte), p
- ▶ 该指针 p 可以用于指向长度为 10 的 int 数组

```
int a[10], b[5][10];
```

数组指针 – 指向数组的指针

数组指针的定义

数组元素类型名 (*指针名)[数组长度名];

```
int (*p)[10];
```

- ▶ 定义了一个指针 (只占 4 个 byte), p
- ▶ 该指针 p 可以用于指向长度为 10 的 int 数组

```
int a[10], b[5][10];
```

```
p = &a; // 将p指向数组a
```

```
p = &b[k]; // 假设k是一个整数变量
```

数组指针 – 指向数组的指针

数组指针的定义

数组元素类型名 (*指针名)[数组长度名];

```
int (*p)[10];
```

- ▶ 定义了一个指针 (只占 4 个 byte), p
- ▶ 该指针 p 可以用于指向长度为 10 的 int 数组

```
int a[10], b[5][10];
```

```
p = &a; // 将p指向数组a
```

```
p = &b[k]; // 假设k是一个整数变量
```

数组指针 p 不可以指向下面的数组

```
int c[20];
```

```
p = &c; // 错! 数组c的长度是20
```

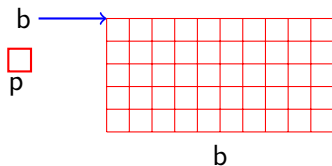
```
float d[10];
```

```
p = &d; // 错! 数组c的元素类型是float
```

数组指针 – 指向数组的指针

数组指针的使用

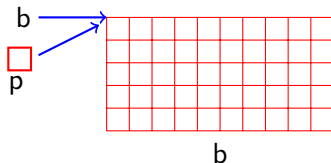
```
int b[5][10], (*p)[10];
```



数组指针 – 指向数组的指针

数组指针的使用

```
int b[5][10], (*p)[10];  
p = &b[0]; // 指针p指向b[0]  
p = b; // &b[0] 就是 b
```

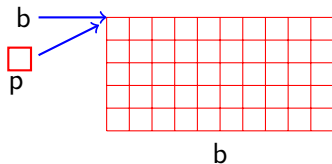


数组指针 – 指向数组的指针

数组指针的使用

```
int b[5][10], (*p)[10];  
p = &b[0]; // 指针p指向b[0]  
p = b; // &b[0] 就是 b
```

指针 $p + k$ 指向哪里？



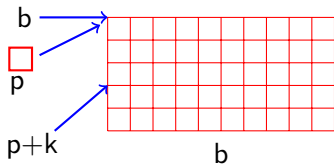
数组指针 – 指向数组的指针

数组指针的使用

```
int b[5][10], (*p)[10];  
p = &b[0]; // 指针p指向b[0]  
p = b; // &b[0] 就是 b
```

指针 $p + k$ 指向哪里？

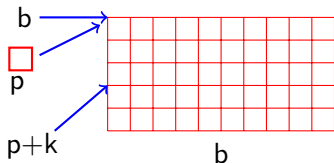
指针 $p + k$ 指向二维数组 b 的第 k 行上的一维数组



数组指针 – 指向数组的指针

数组指针的使用

```
int b[5][10], (*p)[10];  
p = &b[0]; // 指针p指向b[0]  
p = b; // &b[0] 就是 b
```



指针 `p + k` 指向哪里？

指针 `p + k` 指向二维数组 `b` 的第 `k` 行上的一维数组

假设 `j, k` 都是整型变量，想一想：

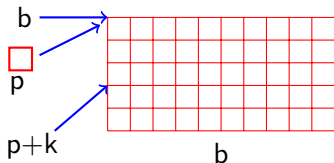
`p[k]`、`b[k]` 是什么？

`p[k][j]`、`b[k][j]` 是什么？

数组指针 – 指向数组的指针

数组指针的使用

```
int b[5][10], (*p)[10];  
p = &b[0]; // 指针p指向b[0]  
p = b; // &b[0] 就是 b
```



指针 `p + k` 指向哪里？

指针 `p + k` 指向二维数组 `b` 的第 `k` 行上的一维数组

假设 `j, k` 都是整型变量，想一想：

`p[k]`、`b[k]` 是什么？

`p[k][j]`、`b[k][j]` 是什么？

其实 `p[k]`、`b[k]` 都是二维数组 `b` 第 `k` 行上的一维数组
`p[k][j]`、`b[k][j]` 都是 `b` 第 `k` 行第 `j` 列上的元素

指针进阶

指针数组

二级指针

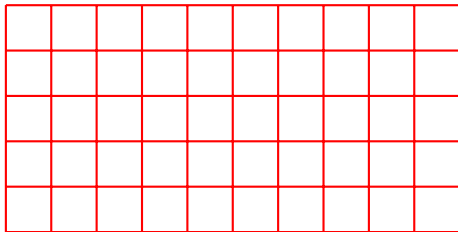
数组指针

指针数组和数组指针区别

指针返回值

数组指针与指针数组的区别

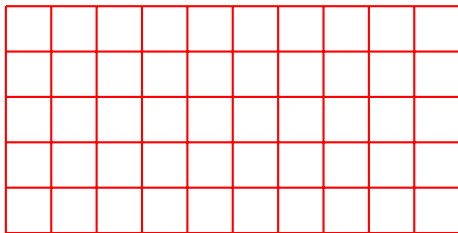
```
int b[5][10], (*p)[10], *a[10];
```



b

数组指针与指针数组的区别

```
int b[5][10], (*p)[10], *a[10];
```



b

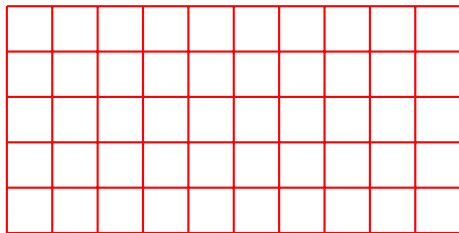


p

指针类型

数组指针与指针数组的区别

```
int b[5][10], (*p)[10], *a[10];
```



b



p

指针类型



a

数组，元素是 `int*` 指针

数组指针与指针数组的区别

```
char ccolor[][7] = { "red", "blue", "yellow",  
    "green", "black"};  
char * pcolor[] = { "red", "blue", "yellow",  
    "green", "black"};
```

r	e	d	\0			
b	l	u	e	\0		
y	e	l	l	o	w	\0
g	r	e	e	n	\0	
b	l	a	c	k	\0	

char ccolor[5][7]

数组指针与指针数组的区别

```
char ccolor[][7] = { "red", "blue", "yellow",  
                    "green", "black"};  
char * pcolor[] = { "red", "blue", "yellow",  
                   "green", "black"};
```

r	e	d	\0			
b	l	u	e	\0		
y	e	l	l	o	w	\0
g	r	e	e	n	\0	
b	l	a	c	k	\0	

char ccolor[5][7]

char* pcolor[5]

数组指针与指针数组的区别

```
char ccolor[][7] = { "red", "blue", "yellow",  
                    "green", "black"};  
char * pcolor[] = { "red", "blue", "yellow",  
                   "green", "black"};
```

r	e	d	\0			
b	l	u	e	\0		
y	e	l	l	o	w	\0
g	r	e	e	n	\0	
b	l	a	c	k	\0	

char ccolor[5][7]

	→	r	e	d	\0		
	→	b	l	u	e	\0	
	→	y	e	l	l	o	w \0
	→	g	r	e	e	n	\0
	→	b	l	a	c	k	\0

char* pcolor[5]

指针数组示例：例 11-4，字符串排序

```
#include<stdio.h>
#include<string.h>
#define SWAP(a,b,t) t=a,a=b,b=t
void fsort(char *s[], int n);
int main(void)
{
    int i;
    char *pcolor[5] = { "red", "blue", "yellow",
                        "green", "black"};
    fsort(pcolor,5);
    for( i = 0; i<5; i++ )
        printf("%s\n", pcolor[i]);
    return 0;
}
```

指针数组示例：例 11-4，字符串排序

```
#include<stdio.h>
#include<string.h>
#define SWAP(a,b,t) t=a,a=b,b=t
void fsort(char *s[], int n);
int main(void)
{
    int i;
    char *pcolor[5] = { "red", "blue", "yellow",
                        "green", "black"};
    fsort(pcolor,5);
    for( i = 0; i<5; i++ )
        printf("%s\n", pcolor[i]);
    return 0;
}
```

pcolor 为指针数组，里面有 5 个指针

指针数组示例：例 11-4，字符串排序

```
#include<stdio.h>
#include<string.h>
#define SWAP(a,b,t) t=a,a=b,b=t
void fsort(char *s[], int n);
int main(void)
{
    int i;
    char *pcolor[5] = { "red", "blue", "yellow",
                        "green", "black"};
    fsort(pcolor,5);
    for( i = 0; i<5; i++ )
        printf("%s\n", pcolor[i]);
    return 0;
}
```

调用排序函数

指针数组示例：例 11-4，字符串排序

```
#include<stdio.h>
#include<string.h>
#define SWAP(a,b,t) t=a,a=b,b=t
void fsort(char *s[], int n);
int main(void)
{
    int i;
    char *pcolor[5] = { "red", "blue", "yellow",
                        "green", "black"};
    fsort(pcolor,5);
    for( i = 0; i<5; i++ )
        printf("%s\n", pcolor[i]);
    return 0;
}
```

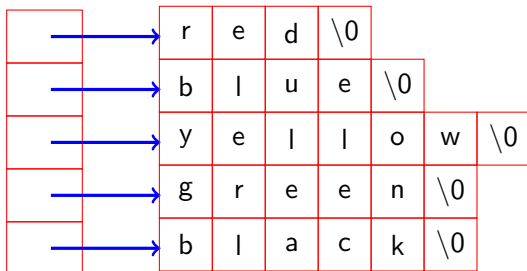
排序函数

指针数组示例：例 11-4，字符串排序

```
void fsort(char *s[], int n)
{
    int k, j;
    for( k=1; k<n; k++ )
        for( j=0; j<n-k; j++ ) {
            char * temp;
            if( strcmp(s[j], s[j+1])>0 )
                SWAP(s[j], s[j+1], temp);
        }
}
```

指针数组示例：例 11-4，字符串排序

```
void fsort(char *s[], int n)
{
    int k, j;
    for( k=1; k<n; k++ )
        for( j=0; j<n-k; j++ ) {
            char * temp;
            if( strcmp(s[j],s[j+1])>0 )
                SWAP(s[j],s[j+1],temp);
        }
}
```



	r	e	d	\0			
	b	l	u	e	\0		
	y	e	l	l	o	w	\0
	g	r	e	e	n	\0	
	b	l	a	c	k	\0	

char* s[]

指针数组示例：例 11-4，字符串排序

```
void fsort(char *s[], int n)
{
    int k, j;
    for( k=1; k<n; k++ )
        for( j=0; j<n-k; j++ ) {
            char * temp;
            if( strcmp(s[j],s[j+1])>0 )
                SWAP(s[j],s[j+1],temp);
        }
}
```

相邻比较、交换

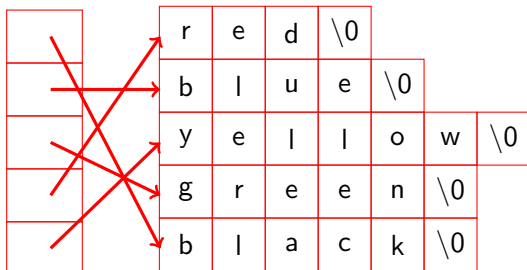
冒泡排序

	→	r	e	d	\0			
	→	b	l	u	e	\0		
	→	y	e	l	l	o	w	\0
	→	g	r	e	e	n	\0	
	→	b	l	a	c	k	\0	

char* s[]

指针数组示例：例 11-4，字符串排序

```
void fsort(char *s[], int n)
{
    int k, j;
    for( k=1; k<n; k++ )
        for( j=0; j<n-k; j++ ) {
            char * temp;
            if( strcmp(s[j],s[j+1])>0 )
                SWAP(s[j],s[j+1],temp);
        }
}
```



char* s[]

指针进阶

指针数组

二级指针

数组指针

指针数组和数组指针区别

指针返回值

指针类型的返回值

```
#include<stdio.h>
char *match(char *s, char ch);
int main(void)
{
    char ch, str[80], *p;
    printf("输入一个字符串: ");
    scanf("%s",str); getchar();
    printf("输入一个准备匹配的字符: ");
    ch = getchar();
    p = match(str,ch);
    printf("%s\n", p ? p : "字符匹配失败");
    return 0;
}
```

指针类型的返回值

匹配函数

```
#include<stdio.h>
char *match(char *s, char ch);
int main(void)
{
    char ch, str[80], *p;
    printf("输入一个字符串: ");
    scanf("%s", str); getchar();
    printf("输入一个准备匹配的字符: ");
    ch = getchar();
    p = match(str, ch);
    printf("%s\n", p ? p : "字符匹配失败");
    return 0;
}
```

指针类型的返回值

```
#include <stdio.h>
char *match(char *s, char ch);
int main(void)
{
    char ch, str[80], *p;
    printf("输入一个字符串: ");
    scanf("%s", str); getchar();
    printf("输入一个准备匹配的字符: ");
    ch = getchar();
    p = match(str, ch);
    printf("%s\n", p ? p : "字符匹配失败");
    return 0;
}
```

match 函数: 在字符串 s 中查找字符 ch。

指针类型的返回值

```
#include<stdio.h>
char *match(char *s, char ch);
int main(void)
{
    char ch, str[80], *p;
    printf("输入一个字符串: ");
    scanf("%s",str); getchar();
    printf("输入一个准备匹配的字符: ");
    ch = getchar();
    p = match(str,ch);
    printf("%s\n", p ? p : "字符匹配失败");
    return 0;
}
```

match 函数: 在字符串 `s` 中查找字符 `ch`。

► 若找到, 则返回所在的指针; 否则返回空指针NULL

指针类型的返回值

```
#include<stdio.h>
char *match(char *s, char ch);
int main(void)
{
    char ch, str[80], *p;
    printf("输入一个字符串: ");
    scanf("%s", str); getchar();
    printf("输入一个准备匹配的字符: ");
    ch = getchar();
    p = match(str, ch);
    printf("%s\n", p ? p : "字符匹配失败");
    return 0;
```

调用匹配函数

} match 函数：在字符串 s 中查找字符 ch。

- ▶ 若找到，则返回所在的指针；否则返回空指针NULL
- ▶ 匹配的返回结果保存在指针p中。

指针类型的返回值

```
char *match(char *s, char ch)
{
    while ( *s && *s!=ch )
        s++;
    return ( *s ? s : NULL );
}
```

指针类型的返回值

```
char *match(char *s, char ch)
{
    while ( *s && *s!=ch )
        s++;
    return ( *s ? s : NULL );
}
```

while 循环结束后, *s==\0 或者 *s==ch

指针类型的返回值

```
char *match(char *s, char ch)
{
    while ( *s && *s!=ch )
        s++;
    return ( *s ? s : NULL );
}
```

while 循环结束后, *s==\0 或者 *s==ch

► 若 *s==\0, 意味着未有匹配, 则返回空指针 NULL

指针类型的返回值

```
char *match(char *s, char ch)
{
    while ( *s && *s!=ch )
        s++;
    return ( *s ? s : NULL );
}
```

while 循环结束后, *s==\0 或者 *s==ch

- ▶ 若 *s == \0, 意味着未有匹配, 则返回空指针 NULL
- ▶ 否则必有 *s == ch, 匹配到了, 且是指针 s 所指的字符, 则返回指针 s

指针类型的返回值

```
char *match(char *s, char ch)
{
    while ( *s && *s!=ch )
        s++;
    return ( *s ? s : NULL );
}
```

while 循环结束后, *s==\0 或者 *s==ch

- ▶ 若 *s == \0, 意味着未有匹配, 则返回空指针 NULL
- ▶ 否则必有 *s == ch, 匹配到了, 且是指针 s 所指的字符, 则返回指针 s

注意: 返回的指针值

指针类型的返回值

```
char *match(char *s, char ch)
{
    while ( *s && *s!=ch )
        s++;
    return ( *s ? s : NULL );
}
```

while 循环结束后, *s==\0 或者 *s==ch

- ▶ 若 *s == \0, 意味着未有匹配, 则返回空指针 NULL
- ▶ 否则必有 *s == ch, 匹配到了, 且是指针 s 所指的字符, 则返回指针 s

注意: 返回的指针值

- ▶ 空指针 NULL, 一个特殊的指针值, 传递失败信息

指针类型的返回值

```
char *match(char *s, char ch)
{
    while ( *s && *s!=ch )
        s++;
    return ( *s ? s : NULL );
}
```

while 循环结束后, *s==\0 或者 *s==ch

- ▶ 若 *s == \0, 意味着未有匹配, 则返回空指针 NULL
- ▶ 否则必有 *s == ch, 匹配到了, 且是指针 s 所指的字符, 则返回指针 s

注意: 返回的指针值

- ▶ 空指针 NULL, 一个特殊的指针值, 传递失败信息
- ▶ 返回的指针值所代表的位置在字符串 s 中, 该指针是合法/可用的 (对调用 match 函数的调用者来说)

指针类型的返回值

指针值类型的函数返回值**必须是一个合法的指针值**

指针类型的返回值

指针值类型的函数返回值**必须是一个合法的指针值**

- ▶ 对调用者来说，返回的指针是必须合法/可用的

指针类型的返回值

指针值类型的函数返回值**必须是一个合法的指针值**

- ▶ 对调用者来说，返回的指针是必须合法/可用的
- ▶ 也可以是空指针 NULL，用这**特殊的指针**传递失败信息

指针类型的返回值

指针值类型的函数返回值**必须是一个合法的指针值**

- ▶ 对调用者来说，返回的指针是必须合法/可用的
- ▶ 也可以是空指针 NULL，用这**特殊的指针**传递失败信息

一些指针值类型的函数返回值

```
char *strcpy(char *s, char *t);  
char *strcat(char *s, char *t);  
char *gets(char *s);  
void *malloc(unsigned int size);
```

指针类型的返回值

指针值类型的函数返回值**必须是一个合法的指针值**

指针类型的返回值

指针值类型的函数返回值必须是一个合法的指针值

```
#include <stdio.h>
char *monthname(int m);
int main(void)
{
    char * p;
    int m;
    printf("输入月份值 (1-12) : ");
    scanf("%d",&m);
    p = monthname(m);
    printf("你输入的月份是: %s\n", p);
    return 0;
}
```

指针类型的返回值

指针值类型的函数返回值必须是一个合法的指针值

```
char *monthname(int m)
{
    char n[][4] = {"Jan", "Feb", "Mar", "Apr",
                  "May", "JUN", "JUL", "AUG", "SEP", "OCT",
                  "NOV", "DEC" };
    return n[m-1];
}
```

指针类型的返回值

指针值类型的函数返回值必须是一个合法的指针值

```
char *monthname(int m)
{
    char n[][4] = {"Jan", "Feb", "Mar", "Apr",
                  "May", "JUN", "JUL", "AUG", "SEP", "OCT",
                  "NOV", "DEC" };
    return n[m-1];
}
```

- ▶ 不可以是局部变量的指针

指针类型的返回值

指针值类型的函数返回值**必须是一个合法的指针值**

```
char *monthname(int m)
{
    char n[][4] = {"Jan", "Feb", "Mar", "Apr",
                  "May", "JUN", "JUL", "AUG", "SEP", "OCT",
                  "NOV", "DEC" };
    return n[m-1];
}
```

当函数返回后，局部变量
会被回收，指向它们
的指针也变得不合法

► 不可以是局部变量的指针

指针类型的返回值

指针值类型的函数返回值必须是一个合法的指针值

```
char *monthname(int m)
{
    char n[][4] = {"Jan", "Feb", "Mar", "Apr",
                  "May", "JUN", "JUL", "AUG", "SEP", "OCT",
                  "NOV", "DEC" };
    return n[m-1];
}
```

当函数返回后，局部变量
会被回收，指向它们
的指针也变得不合法

- ▶ 不可以是局部变量的指针
- ▶ 纠正方法：静态化，让变量 `n` 一直存在，不被收回

```
static char n[][4] = {"Jan", "Feb", "Mar",
                    "Apr", "May", "JUN", "JUL", "AUG", "
                    SEP", "OCT", "NOV", "DEC" };
```

总结

指针进阶

- 指针数组

- 二级指针

- 数组指针

- 指针数组和数组指针区别

- 指针返回值

今天到此为止