指针的进阶

笔记本:	My Notebook		
创建时间:	2023/10/12 19:47	**更新时间:**	2023/10/15 21:20
作者:	dtdkc1mu		
URL:	mk:@MSITStore:D:\bo\c语言学习\MSDN\vccore.chm::/html/_crt_qsort.htm		

重点
1. 字符指针
2. 数组指针
3. 指针数组
4. 数组传参和指针传参
5. 函数指针
6. 函数指针数组
7. 指向函数指针数组的指针
8. 回调函数
9. 指针和数组面试题的解析

指针就是一个变量，用来存放地址，地址唯一标识一块内存空间
指针的大小是固定的4/8个字节（32位平台/64位平台）
指针是类型的，指针的类型决定了指针的+-整数的步长，指针解引用操作的时候的权限。
指针的运算

1.字符指针

```
int main()
{
	const char* p = "abcdef"; //"abcdef"是一个常量字符串，这里是将其的首地址赋给了p
	// *p = 'W'; // segmentation fault - 段错误 - 访问非法内存
	// printf("%c\n", *p); // a
	printf("%s\n", p); // abcdef
	return 0;
}
```

```
// 以下代码输出hehe
int main()
{
	char arr1[] = "abcdef";
	char arr2[] = "abcdef";
	const char* p1 = "abcdef"; // 常量字符串
	const char* p2 = "abcdef";
	if (p1 == p2)
	{
		printf("hehe\n");
	}
	else
	{
		printf("haha\n");
	}
	return 0;
}
```

这里p1和p2指向的是一个同一个常量字符串，C/C++会把常量字符串存储到单独的一个内存区域，当几个指针，指向同一个常量字符串的时候，他们实际上会指向同一块内存。但是用相同的

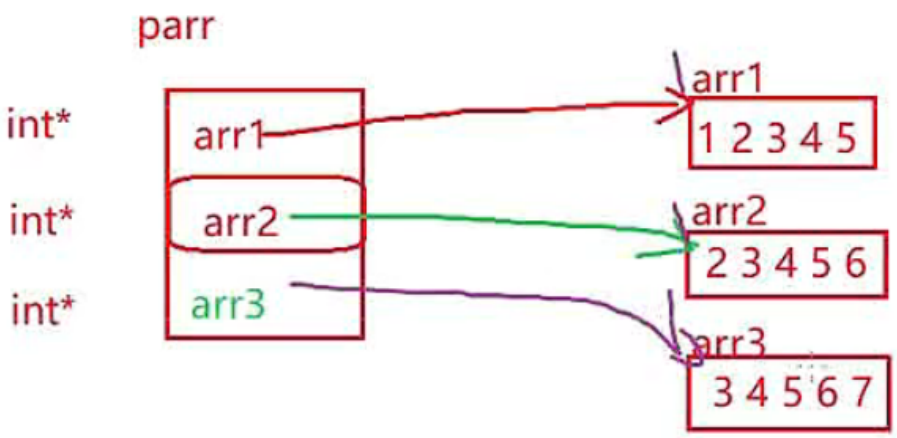
常量字符串去初始化不同的数组的时候就会开辟不同的内存块，所以arr1和arr2不同，p1和p2相同。

指针数组

指针数组是数组，用来存放指针

```
int main()
{
	int arr1[] = { 1,2,3,4,5 };
	int arr2[] = { 2,3,4,5,6 };
	int arr3[] = { 3,4,5,6,7 };
	int* parr[] = { arr1, arr2, arr3 };
	int i = 0;
	for (i = 0; i < 3; i++)
	{
		int j = 0;
		for (j = 0; j < 5; j++)
		{
			printf("%d ", *(parr[i] + j));
		}
		printf("\n");
	}
	return 0
}
```

内存结构



数组指针

数组指针 - 指针 - 存放数组的地址

```
int (*p)[10];
//解释：p先和*结合，说明p是一个指针变量，然后指着指向的是一个大小为10个整型的数组。所以p是一个
指针，指向一个数组，叫数组指针。
//这里要注意：[]的优先级要高于*号的，所以必须加上（）来保证p先和*结合。
```

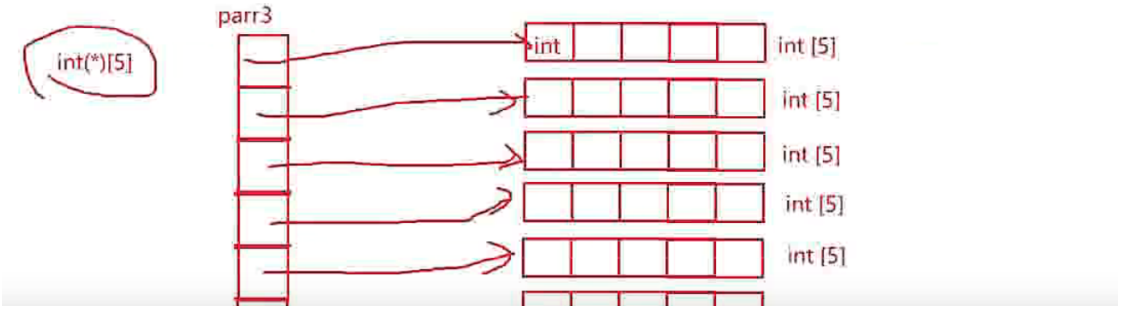
数组指针的使用

```
// print1和print2的打印结果相同
void print1(int arr[3][5], int x, int y)
{
	int i = 0;
	int j = 0;
	for (i = 0; i < x; i++)
	{
		for (j = 0; j < y; j++)
		{
			printf("%d ", arr[i][j]);
		}
```

```
                printf("\n");
        }
}
//参数是指针的形式
void print2(int (*p)[5], int x, int y)
{
        int i = 0;
        for (i = 0; i < x; i++)
        {
                int j = 0;
                for (j = 0; j < y; j++)
                {
                        printf("%d ", *(*(p + i) + j));
                }
                printf("\n");
        }
}
int main()
{
        int arr[3][5] = { {1,2,3,4,5}, {2,3,4,5,6}, {3,4,5,6,7} };
        print1(arr, 3, 5);
        // arr - 数组名 - 首元素地址 - 但是二维数组的首元素是二维数组的第一行
        // 所以这里传递的arr，其实相当于第一行的地址，是一维数组的地址
        // 所以可以用数组指针来接受
        print2(arr, 3, 5);
        return 0;
}
```

```
        int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };
        int i = 0;
        int* p = arr;
        for (i = 0; i < 10; i++)
        {
                printf("%d ", p[i]);
                printf("%d ", *(p + i));
                printf("%d ", *(arr + i));
                printf("%d ", arr[i]); // arr[i] = *(arr+i) = p[i] = *(p+i)
        }
```

```
int arr[5]; // arr是一个5个元素的整型数组
int *parr1[10]; // parr1是一个数组，数组有10个元素，每个元素的类型为int*，paar1是指针数组
int (*parr2)[10]; // parr2是一个指针，该指针指向了一个数组，数组有十个元素，每个元素的类型是int， parr2是数组指针
int (*parr3[10])[5]; // paar3是一个数组，该数字有10个元素，每个元素是一个数组指针，该数组指针指向的数组有5个元素，每个元素是int。
```



数组参数、指针参数

```
//一维数组传参
void test(int arr[])//ok
{}
```

```
void test(int arr[10])//ok
{}
void test(int *arr)//ok
{}
void test2(int *arr[20])//ok
{}
void test2(int **arr)//ok， 一级指针的地址传到二级指针中，没有问题
{}
int main()
{
int arr[10] = {0};
int *arr2[20] = {0};
test(arr);
test2(arr2);
}
```

二维数组传参
// 二维数组传参，函数形参的设计只能省略第一个[]的数字，
//因为对一个二维数组，可以不知道有多少行，但是必须知道一行多少元素

```
void test(int arr[3][5])//ok？ yes
{}
void test(int arr[][5])//ok？ yes
{}
void test(int (*arr)[5]) //ok? yes
{}
int main()
{
int arr[3][5] = {0};
test(arr);
}
```

一级指针传参
当一个函数的参数为一级指针时，函数能接收什么参数?
可以接受变量的地址或者一级指针变量

二级指针传参
可以接受一级指针变量的地址或者二级指针变量，也可以传递指针数组的首元素地址

函数指针
存放函数地址的一个指针
&函数名 和 函数名都是函数的地址

```
int Add(int a, int b)
{
        return a + b;
}
int main()
{
        int a = 10;
        int b = 20;
        // &函数名 和 函数名 都是函数的地址
        printf("%p\n", &Add);
        printf("%p\n", Add);
        // 函数指针
        int (*pa)(int, int) = Add; // pa先和*结合,说明pa是指针，指针指向的是一个函
数，指向的函数参数为(int,int)，返回值为int。
        printf("%d\n", (*pa)(a, b)); // *pa解引用得到函数，然后调用
        return 0;
}
```

两段有趣的代码

```
//代码1
(*(void (*)())0)(); // 把0强制类型转换成：void (*)()类型， *解引用然后调用0地址处的该
函数
```

```
//代码2
void (*signal(int , void(*)(int)))(int); // 函数声明，这里声明一个参数为(int,void(*)(int)),且返回值为参数为(int),返回值为void函数指针的函数signal。

//signal是一个函数声明
// signal函数的参数有两个，第一个是int，第二个是函数指针，该函数指针指向的函数的参数是int，返回值是void
// signal函数的返回类型也是一个函数指针：该函数指针指向的函数参数是int，返回类型是void
// void (*signal(int, void(*)(int))) (int);
//简化
pfun_t signal(int, pfun_t);
typedef void(*pfun_t)(int)
```

函数指针数组

```
int (*parr[10])() // parr先和[]结合，说明parr是数组
```

函数指针数组的用途：转移表

```
#include <stdio.h>
int add(int a, int b)
{
          return a + b;
}
int sub(int a, int b)
{
          return a - b;
}
int mul(int a, int b)
{
          return a*b;
}
int div(int a, int b)
{
          return a / b;
}
int main()
{
    int x, y;
    int input = 1;
    int ret = 0;
    int(*p[5])(int x, int y) = { 0, add, sub, mul, div }; //转移表
    while (input)
    {
         printf( "*************************\n" );
         printf( " 1:add           2:sub \n" );
         printf( " 3:mul           4:div \n" );
         printf( "*************************\n" );
         printf( "请选择：" );
     scanf( "%d", &input);
         if ((input <= 4 && input >= 1))
        {
         printf( "输入操作数：" );
             scanf( "%d %d", &x, &y);
             ret = (*p[input])(x, y);
        }
         else
              printf( "输入有误\n" );
         printf( "ret = %d\n", ret);
    }
     return 0;
```

指向函数指针数组的指针

```
int main()
{
        int arr[10] = { 0 };
        int(*p)[10] = arr;
        int (*pfArr[4])(int, int); // pfArr是一个数组 - 函数指针的数组
```

```
        int (*(*ppfArr)[4])(int, int) = &pfArr;
        // ppfArr 是一个数组指针，指针指向的数组有四个元素
        // 指向的数组的每个元素的类型是一个函数指针 int(*) (int, int)
}
```

回调函数
回调函数就是一个通过函数指针调用的函数，如果你把函数的指针（地址）作为参数传递给另一个函数，但这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另一方调用的，用于对该事件或条件进行响应。

void qsort(void *base, size_t num, size_t width, int (*compare)(const void *elem1, const void *elem2))
// 第一个参数：待排序数组的首元素地址
// 第二个参数：待排序的元素个数
// 第三个参数：待排序数组的每个元素的大小 - 单位是字节
// 第四个参数：是函数指针，比较两个元素的所用函数的地址-这个函数使用者自己实现
// 函数指针的两个参数是：带比较的两个元素的地址

void*
void* 类型的指针可以接受任意类型的地址
void* 类型的指针不能进行解引用操作
void* 类型的指针也不能进行+-整数的操作

```
// 使用函数指针和void*实现通用的冒泡排序
void Swap(char* buf1, char* buf2, int width)
{
        int i = 0;
        for (i = 0; i < width; i++)
        {
                char tmp = *buf1;
                *buf1 = *buf2;
                *buf2 = tmp;
                buf1++;
                buf2++;
        }
}
void bubble_sort(void* base, int sz, int width, int (*cmp)(const void* e1, const
void*  e2))
{
        int i = 0;
        for (i = 0; i < sz - 1; i++)
        {
                int flag = 0;
                int j = 0;
                for (j = 0; j < sz - i - 1; j++)
                {
                        if (cmp((char*)base + j * width, (char*)base + (j + 1) *
width) >  0)
                        {
                                // 交换
                                Swap((char*)base + j * width, (char*)base + (j + 1)
* width,  width);
                        }
                }
        }
}
```

数组名的意义
1.sizeof(数组名)，这里的数组名表示整个数组，计算的是整个数组的大小
2.&数组名，这里的数组名表示整个数组，取出的是整个数组的地址
3.除此之外所有的数组名都表示首元素的地址