第14讲：深入理解指针(4)

目录

正文开始

1. 回调函数是什么?

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，当这个指针被用来调用其所指向的函数时，被调用的函数就是回调函数。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外的一方调用的，用于对该事件或条件进行响应。

第13讲中我们写的计算机的实现的代码中，红色框中的代码是重复出现的，其中虽然执行计算的逻辑是区别的，但是输入输出操作是冗余的，有没有办法，简化一些呢?

因为红色框中的代码，只有调用函数的逻辑是有差异的，我们可以把调用的函数的地址以参数的形式传递过去，使用函数指针接收，函数指针指向什么函数就调用什么函数，这里其实使用的就是回调函数的功能。

```c
//使用回调函数改造前
#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;
```

```c
//使用回到函数改造后
#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;
```

```c
}
int div(int a, int b)
{
    return a / b;
}
int main()
{
    int x, y;
    int input = 1;
    int ret = 0;
    do
    {
        printf("*****************
        printf("  1:add
        printf("  3:mul
        printf("*****************
        printf("请选择: ");
        scanf("%d", &input);
        switch (input)
        {
        case 1:
            printf("输入操作数: ");
            scanf("%d %d", &x, &y
            ret = add(x, y);
            printf("ret = %d\n", r
            break;
        case 2:
            printf("输入操作数: ");
            scanf("%d %d", &x, &y
            ret = sub(x, y);
            printf("ret = %d\n", r
            break;
        case 3:
            printf("输入操作数: ");
            scanf("%d %d", &x, &y
            ret = mul(x, y);
            printf("ret = %d\n", r
            break;
        case 4:
            printf("输入操作数: ");
            scanf("%d %d", &x, &y
            ret = div(x, y);
            printf("ret = %d\n", r
            break;
        case 0:
            printf("退出程序\n");
            break;
```

```c
}
int div(int a, int b)
{
    return a / b;
}
void calc(int(*pf)(int, int))
{
    int ret = 0;
    int x, y;
    printf("输入操作数: ");
    scanf("%d %d", &x, &y);
    ret = pf(x, y);
    printf("ret = %d\n", ret);
}
int main()
{
    int input = 1;
    do
    {
        printf("*****************
        printf("  1:add
        printf("  3:mul
        printf("*****************
        printf("请选择: ");
        scanf("%d", &input);
        switch (input)
        {
        case 1:
            calc(add);
            break;
        case 2:
            calc(sub);
            break;
        case 3:
            calc(mul);
            break;
        case 4:
            calc(div);
            break;
        case 0:
            printf("退出程序\n");
            break;
        default:
            printf("选择错误\n");
            break;
        }
    } while (input);
```

```
61          default:
62              printf("选择错误\n");
63              break;
64          }
65      } while (input);
66
67      return 0;
68  }
```

```
61
62      return 0;
63  }
```

2. qsort使用举例

2.1 使用qsort函数排序整型数据

```
1  #include <stdio.h>
2
3  //qosrt函数的使用者得实现一个比较函数
4  int int_cmp(const void * p1, const void * p2)
5  {
6       return (*( int *)p1 - *(int *) p2);
7  }
8
9  int main()
10 {
11     int arr[] = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 0 };
12     int i = 0;
13
14     qsort(arr, sizeof(arr) / sizeof(arr[0]), sizeof (int), int_cmp);
15     for (i = 0; i< sizeof(arr) / sizeof(arr[0]); i++)
16     {
17        printf( "%d ", arr[i]);
18     }
19     printf("\n");
20     return 0;
21 }
```

2.2 使用qsort排序结构数据

```
1  struct Stu //学生
2  {
3      char name[20];//名字
4      int age;//年龄
```

```c
};

//假设按照年龄来比较
int cmp_stu_by_age(const void* e1, const void* e2)
{
    return ((struct Stu*)e1)->age - ((struct Stu*)e2)->age;
}

//strcmp - 是库函数，是专门用来比较两个字符串的大小的
//假设按照名字来比较
int cmp_stu_by_name(const void* e1, const void* e2)
{
    return strcmp(((struct Stu*)e1)->name, ((struct Stu*)e2)->name);
}

//按照年龄来排序
void test2()
{
    struct Stu s[] = { {"zhangsan", 20}, {"lisi", 30}, {"wangwu", 15} };
    int sz = sizeof(s) / sizeof(s[0]);
    qsort(s, sz, sizeof(s[0]), cmp_stu_by_age);
}

//按照名字来排序
void test3()
{
    struct Stu s[] = { {"zhangsan", 20}, {"lisi", 30}, {"wangwu", 15} };
    int sz = sizeof(s) / sizeof(s[0]);
    qsort(s, sz, sizeof(s[0]), cmp_stu_by_name);
}

int main()
{
    test2();
    test3();
    return 0;
}
```

3. qsort函数的模拟实现

使用回调函数，模拟实现qsort（采用冒泡的方式）。

注意：这里第一次使用 `void*` 的指针，讲解 `void*` 的作用。

```c
#include <stdio.h>

```

```c
int int_cmp(const void * p1, const void * p2)
{
          return (*( int *)p1 - *(int *) p2);
}

void _swap(void *p1, void * p2, int size)
{
    int i = 0;
    for (i = 0; i< size; i++)
    {
        char tmp = *((char *)p1 + i);
       *(( char *)p1 + i) = *((char *) p2 + i);
       *(( char *)p2 + i) = tmp;
    }
}

void bubble(void *base, int count , int size, int(*cmp )(void *, void *))
{
    int i = 0;
    int j = 0;
    for (i = 0; i< count - 1; i++)
    {
       for (j = 0; j<count-i-1; j++)
       {
            if (cmp ((char *) base + j*size , (char *)base + (j + 1)*size) > 0)
           {
               _swap(( char *)base + j*size, (char *)base + (j + 1)*size, size);
           }
       }
    }
}
int main()
{
    int arr[] = { 1, 3, 5, 7, 9, 2, 4, 6, 8, 0 };
    int i = 0;
    bubble(arr, sizeof(arr) / sizeof(arr[0]), sizeof (int), int_cmp);
    for (i = 0; i< sizeof(arr) / sizeof(arr[0]); i++)
    {
       printf( "%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

完