**C语言成员变量--栈与堆**

2019-03-28 22:11:01 [Unclezs](https://me.csdn.net/qq_42006120) 阅读数 156更多

分类专栏： [C](https://blog.csdn.net/qq_42006120/article/category/8800092)

版权声明：本文为博主原创文章，遵循 [CC 4.0 BY-SA](http://creativecommons.org/licenses/by-sa/4.0/" \t "_blank) 版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/qq_42006120/article/details/88879753>

**一、写在前面**

最近学数据结构打算重新学一遍c语言，c语言函数返回值不像是java一样，java在定义一个成员变量后给其赋值，直接返回还是能拿到他的值，c却不一样了，成员变量放在栈内存中，栈内存中的成员变量会在方法执行后会释放，所以在成员变量赋值后返回是拿不到数据的

**二、栈中的成员变量**

**1.不能取得值1**

**例子**

看看这个例子

char \* testJB()

{

char a='H';

return &a;

}

void main()

{

char p=testJB();

putchar(p);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

这个打印的时候你会发现没看到想要的’H’

**分析**

当testJB执行完成后，a执行的栈内存区域将自动被释放，这时候再返回a的地址已经拿不到数据了

**2.不能取得值2**

**例子**

char \* test()

{

char arr[]="hello world";

return arr;

}

void main()

{

char \*p=test();

puts(p);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

运行结果：乱码输出；

**分析**

char arr[]="hello world";

* 1

内存过程:  
首先将“hello world” 放入常量区，然后再把其重常量区拷贝到arr的栈内存中，函数执行完成后，arr的栈内存被自动释放，返回的arr首地址也拿不到数据了；

**二、堆内存**

**1.可以取得值1**

**例子**

char \* test()

{

char \*p = malloc(sizeof(char)\*10);

\*p = "hello word";

return p;

}

void main()

{

char \*p=test();

puts(p);

free(p);

}

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

运行结果:hello word；

**分析：**

malloc 函数是在**堆内存**中开辟一块空间，并且不会自动释放,需要用完后手动释放；  
因此，当函数执行完成后，堆内存不被释放，返回首地址仍然可以取得数据，注意用完后手动释放；

**2.不可取得值1**

**例子**

void test(char \*num)

{

num = malloc(sizeof(char) \* 10);

num="hello word";

}

void main()

{

char \*p=NULL;

testCY(p);

puts(p);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

运行结果：报错；

**分析：**

test函数中的num指针和main函数中的p指针不是同一个指针,因为形参也可以看作成员变量；在test函数开辟的空间是给num指针开辟的，而不是给p指针开辟的，这时候的p指针一直为NULL；所以这时候报错；

**3.可以取得值2**

**例子**

void testCY(char \*\*num)

{

char \*tmp = malloc(100);

memset(tmp,0,100);

strcpy(tmp,"hello word");

\*num = tmp;

}

void main()

{

char \*p=NULL;

testCY(&p);

puts(p);

free(p);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

运行结果：hello word

**分析**

首先将二级指针num的值赋值为一级指针p的地址值，再创建临时指针tmp，在堆中开辟一块空间,tmp指针指向这块内存，将“hello world”从常量区拷贝到这块内存，再将一级指针p指向tmp指向的地址，这样在函数结束后，二级指针num、一级指针tmp会被销毁，但是一级指针p已经指向了开辟的堆内存；

优化：不需要tmp也可以做到，用*num代替*tmp；