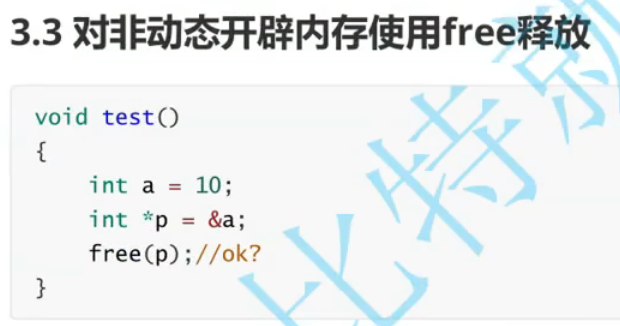
## 常见的动态内存错误



再用malloc系列函数的时候，千万记得要判断是否为空指针，不要马上就解引用！

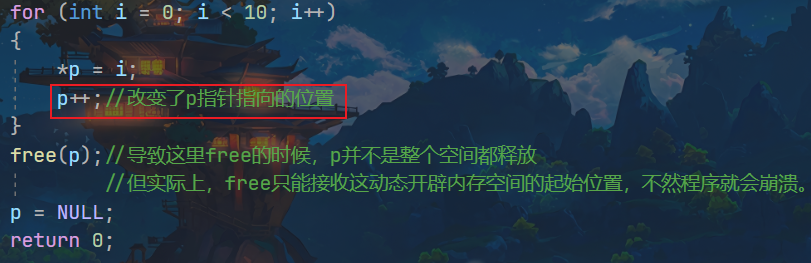


也是不能越界访问的哦~

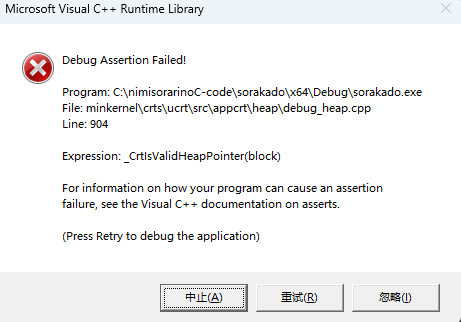


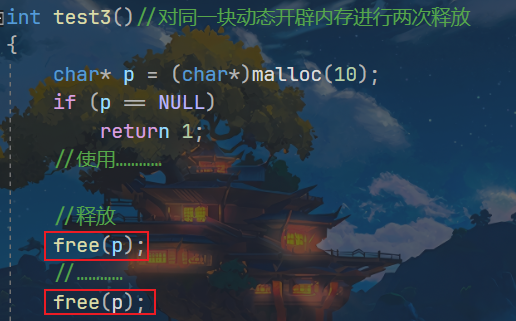
动态开辟的内存在堆区上，free怎么能释放在栈区上的内存空间呢？





P已经没有指向动态开辟内存的起始位置了，实际上，free只能接收这动态开辟内存空间的起始位置，不然程序就会崩溃。

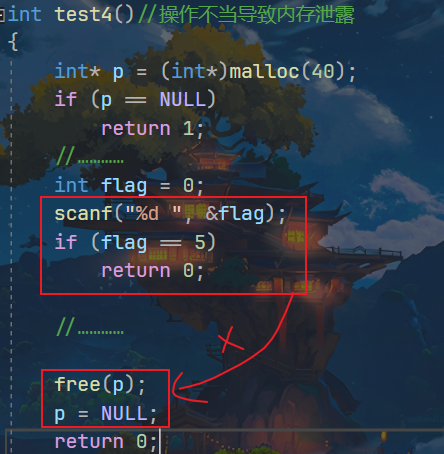




所以记得free完之后马上把指针变为空指针。

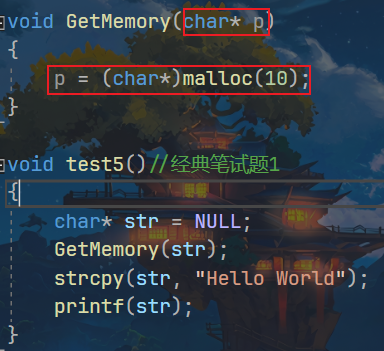
这样下一次free的时候传入空指针，free就do nothing。





有时候我们写代码的时候，可能程序在中间就已经结束了，而p指向的那块内存没能完成free。程序退出后，我们可能再也不能再内存中找到那块被遗弃的内存，系统也认为那块内存被占用着，至此形成内存泄漏。(重启吧)。

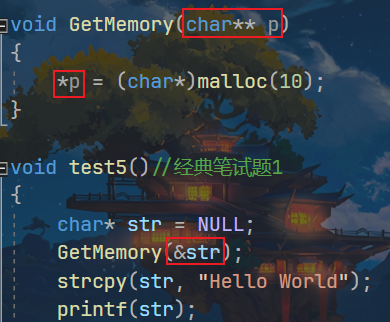
笔试题1



当我们传入Getmemory一个str时，Getmemory创建了一个形参p，并用此指针接收了用malloc动态开辟的10字节空间。但Getmemory函数结束后，p被销毁，并没有改变str的值，因此此时str仍然为空指针；而这也造成了内存泄露。

注意：指针在函数中没进行解引用的话，也是起不到改变外接参数的作用的！！

要对str的值进行改变，就应该传str的地址，那str的地址类型为二级指针，因此代码应该这么改。



注意，用malloc动态开辟内存位于堆区，当函数调用完后，函数栈帧销毁，但是堆区申请出来的空间并不会被释放，需要我们程序员手动释放。

这个printf的写法

可以不用指定格式，直接传入字符串首字符的地址。

我们之前printf(“Hello World\n”);的时候，传的其实就是H的地址

那和 char\* str=”Hello World\n”;

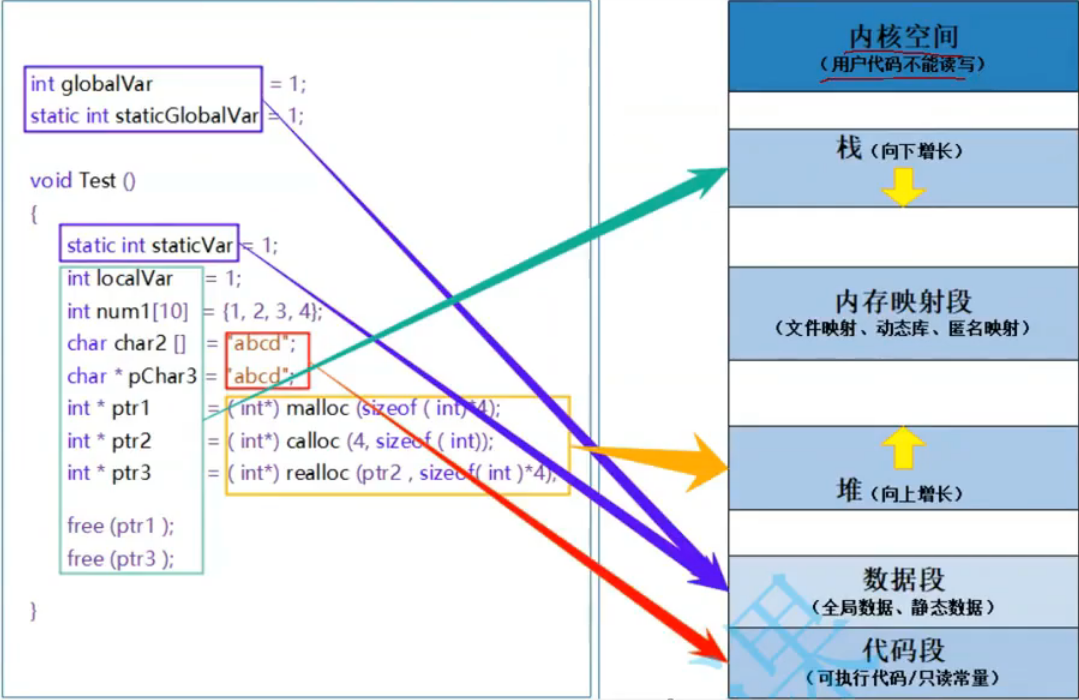
char str[20]=”Hello World\n”;

printf(str);

是一样的。



再次函数栈帧



数据段就是我们说的静态区，里面放全局变量和静态变量。

代码段：代码编译后会产生可执行程序的二进制指令，它们就存放在代码段中。

scanf的返回值: On success, the function returns the number of items of the argument list successfully filled

成功后，该函数返回成功填充的参数列表的项数

三目运算符：变量 =（条件判断）？表达式A:表达式B

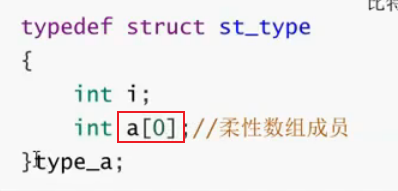
若条件为真 将表达式A赋值变量

若条件为假 将表达式B赋值变量

可以免去if条件判断赋值。

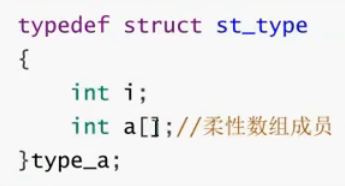
## 柔性数组(flexible array)



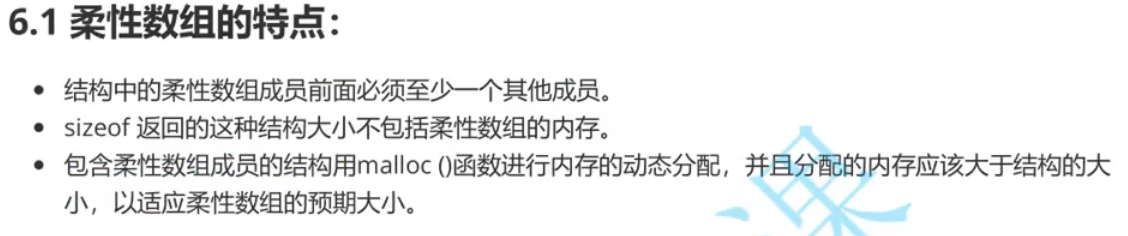


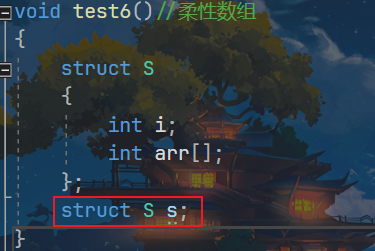
数组大小写0表示数组大小未知。

有些编译器会报错无法编译可以改成:



柔性数组在结构体中只能存在一个。

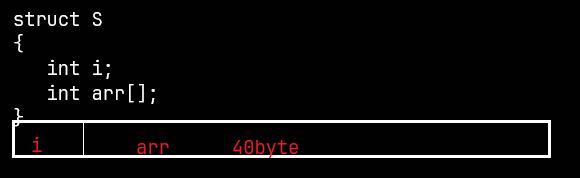


当我们创建的结构体内有柔性数组成员时，

不能这样直接创建，不然柔性数组就真没有空间了。

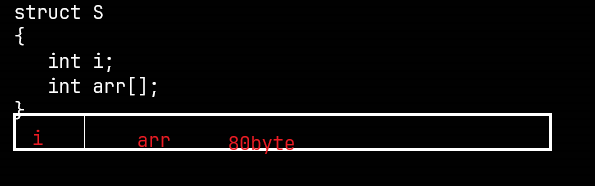
柔性数组需要用malloc函数进行动态开辟内存。





重新扩容



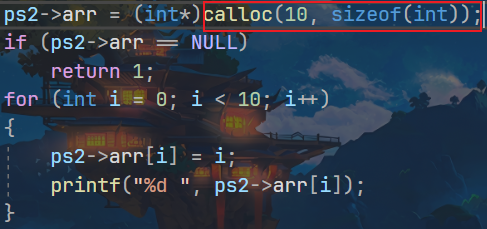


我们如果在自定义结构体中放一个指针，用这个指针去malloc的话，也能达到相似的效果。

为了保证我们申请的空间都位于堆区，我们用malloc对结构体进行初始化。



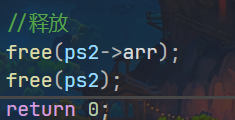
接着我们用calloc用结构体内的指针进行动态开辟



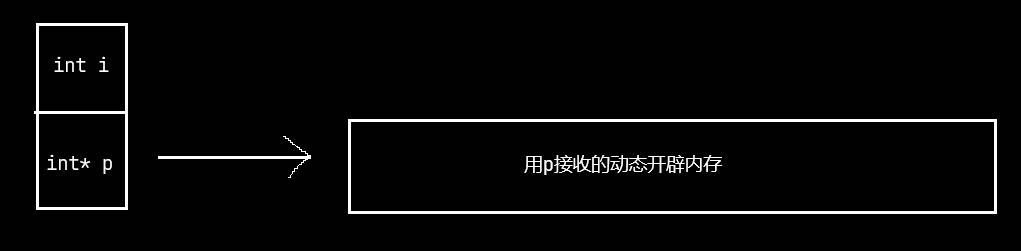
扩容



释放：



注意，释放的时候先要释放结构体指针内指向的动态开辟内存，不然先释放ps2的话就找不到那块空间了，造成内存泄漏。



相比较下，柔性数组在申请内存的时候能少用一次malloc，

并且内存空间是连续的，因此释放的时候也只用一次。

（当然如过让结构体位于栈区的话，确实是不用malloc两次）。