

【C语言】关键字static——static修饰局部变量、全局变量和函数详解！

原创 釉色清风 已于 2023-11-13 21:27:24 修改 阅读量1.3k 收藏 29 点赞数 22 版权

分类专栏: C语言 文章标签: c语言 java 开发语言



C语言 专栏收录该内容

0 订阅 22 篇文章 订阅专栏

在C语言中，static是修饰变量和函数的。static修饰局部变量称为静态局部变量，static修饰全局变量称为静态全局变量，static修饰函数称为静态函数。

文章目录

- 静态变量在静态区分配内存
- static修饰全局变量
- static修饰局部变量
- static修饰函数

静态变量在静态区分配内存

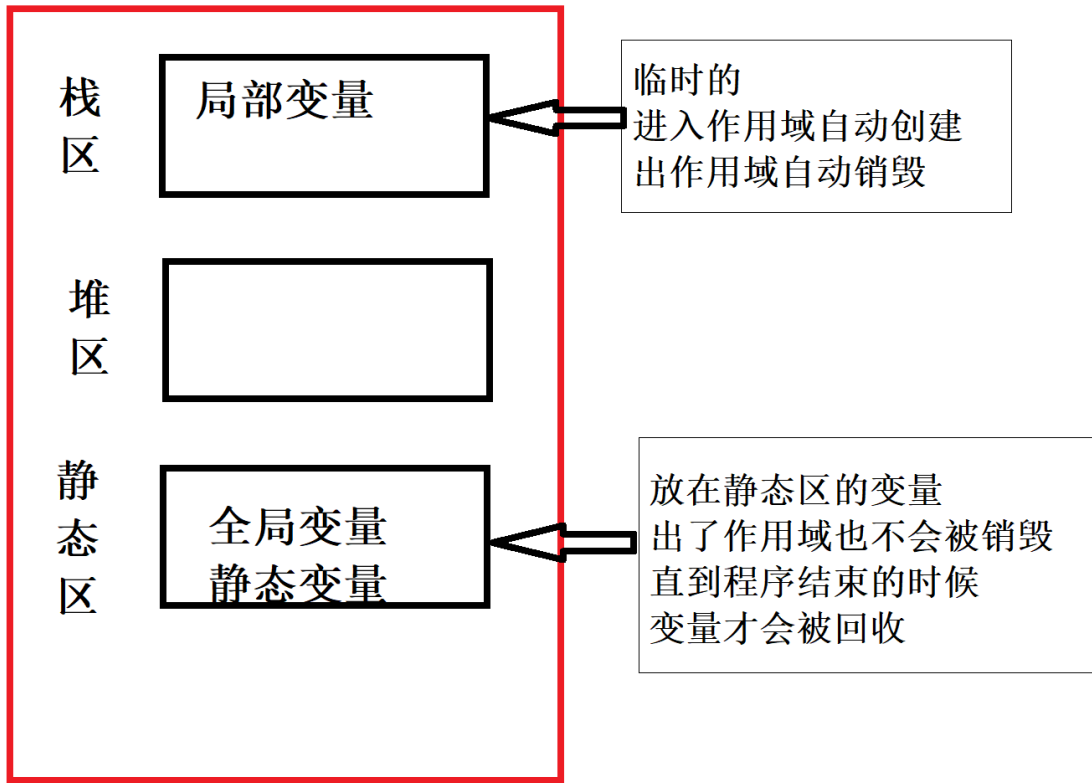
全局变量和被static修饰后的局部变量都在 **静态区** 分配内存。

对于内存，我们可以简单的理解为，内存分为三个部分，栈区、堆区和静态区。

栈区： 保存局部变量，栈上的内容只在函数的范围内存在，当函数运行结束，这些内容也会自动被销毁。栈区的特点是效率高，但是空间有限。

堆区： 由malloc系列函数或new操作符分配内存。其生命周期由free或delete决定。在没有释放之前一直存在，直到程序结束。其特点是使用灵活，空间比较大，但是容易出错。

静态区： 保存全局变量和静态变量，静态区的内容在整个程序的生命周期内都存在，由编译系统在编译的时候分配。



CSDN @釉色清风

static修饰全局变量

静态全局变量 有以下特点：

未经初始化的静态全局变量会被程序自动初始化为0

```
1 #include <stdio.h>
2 static int g_val;
3 int main()
4 {
5     printf("%d", g_val); // 0
6     return 0;
7 }
```

而在函数体内声明的自动变量的值是**随机**的，除非它被显式初始化，而在函数体外被声明的自动变量也会被初始化为0。

静态全局变量在它的整个文件都是可见的，而在文件之外是不可见的

首先，我们要知道全局变量是有外部链接属性的，只要合理声明，全局变量在其他源文件内部，可以使用。而被static修饰后，外部链接属性就变成了内部链接属性，只能在自己所在的源文件内部使用了。

全局变量的外部链接属性：

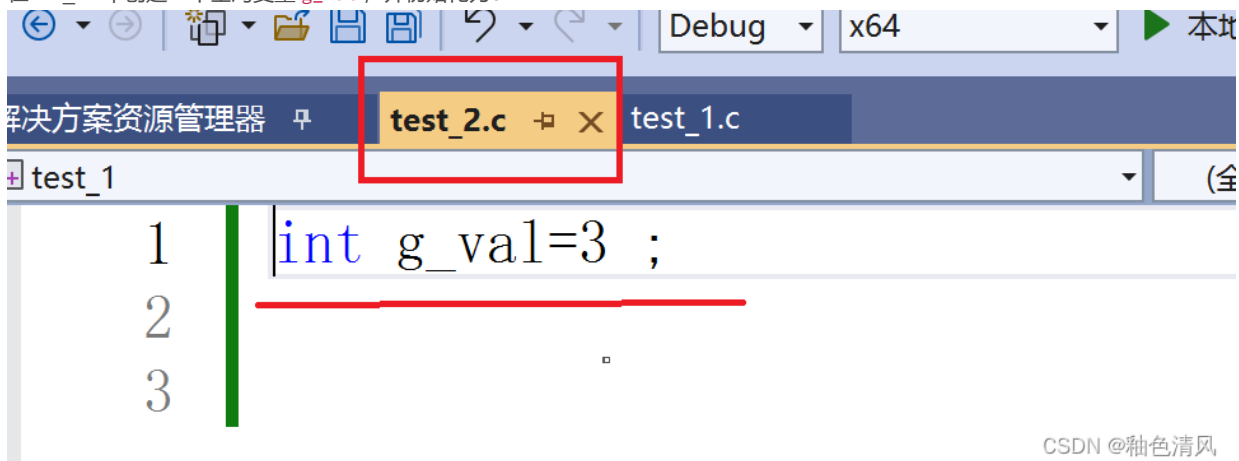
简单点说，就是在一个工程中，有多个.c文件，在一个.c文件中定义了全局变量，是可以跨文件使用的，在其他.c文件也是可以使用这个全局变量的。

但是对于全局变量的跨文件使用，不是直接使用就可以使用，也是需要声明的。

下面举个简单的例子说明：

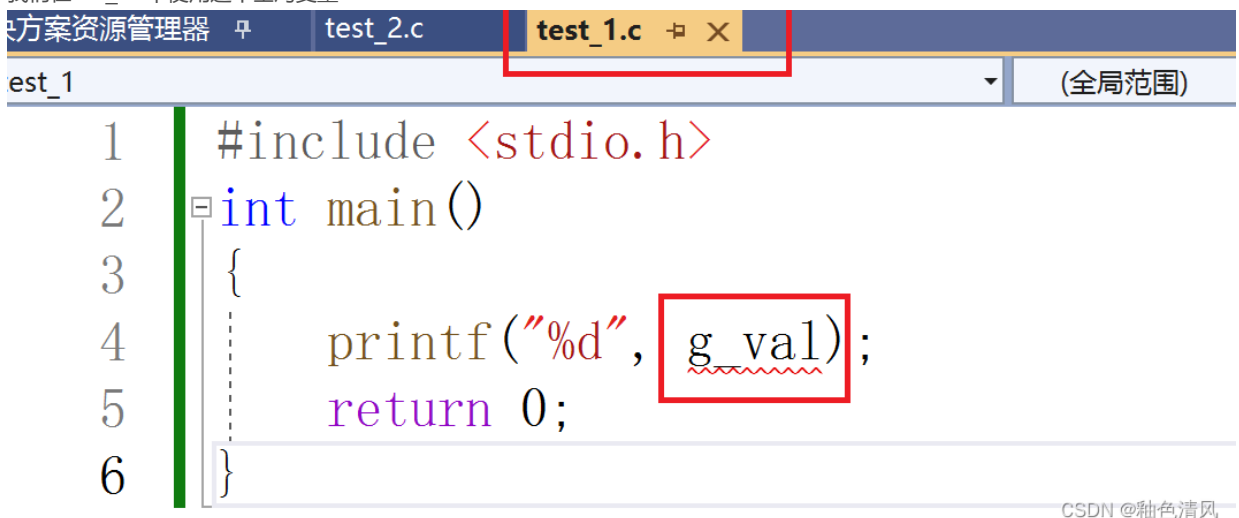
我们在一个工程中创建 test_1.c 和 test_2.c 两个源文件。

在test_2.c中创建一个全局变量 g_val，并初始化为3。



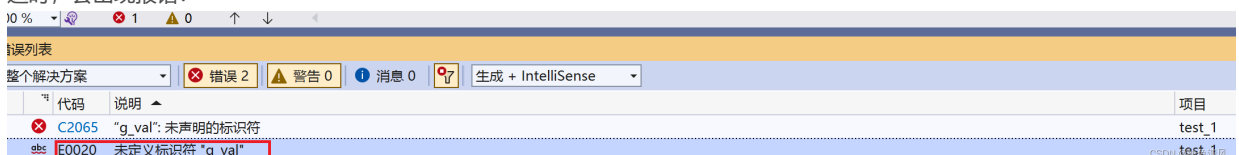
CSDN @釉色清风

我们在test_1.c中使用这个全局变量。

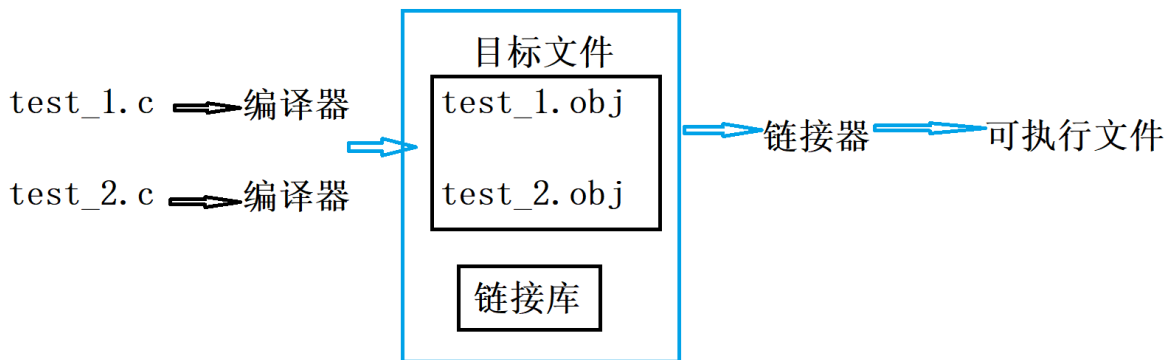


CSDN @釉色清风

这时，会出现报错：



原因：这是因为编译器在编译的时候，是对每个.c文件进行单独进行编译的。
也就是说，对test_1.c进行编译的时候，对于出现的g_val在这个源文件的见面并没有进行定义，所以就会报错。
(下面简单补充对于含多个文件的工程编译连接的过程)



CSDN @釉色清风

解决：使用关键字extern，来声明外部变量。

```
t_1 (全局范围)
1  #include <stdio.h>
2  extern int g_val; //extern 声明外部符号
3  int main()
4  {
5      printf("%d", g_val);
6      return 0;
7  }
```

CSDN @釉色清风

extern声明外部符号，以此告诉编译器，有一个变量叫 g_val，它的类型是int.

static修饰全局变量：

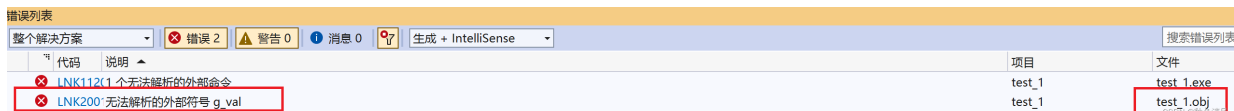
```
//全局变量
static int g_val=3 ;
```

CSDN @釉色清风

```
1 #include <stdio.h>
2 extern int g_val; //extern 声明外部符号
3 int main()
4 {
5     printf("%d", g_val);
6     return 0;
7 }
```

CSDN @袖色清风

发现这时就会报错：



链接test_1.obj时报错。

原因：

这正是因为static修饰全局变量后，使得全局变量只能在自己所在的源文件内部使用，其他源文件无法使用。

所以，全局变量是有 外部链接属性 的，只要合理声明，全局变量在其他源文件内部，可以使用。而被 static修饰 后，外部链接属性就变成了 内部链接属性，只能在自己所在的源文件内部使用了。

好处：

的确，定义全局变量可以实现变量在整个文件中的共享，但定义静态全局变量也有一下好处：

- 静态全局变量不能被其他文件所用。
- 其他文件中可以定义相同名字的变量，不会发生冲突。

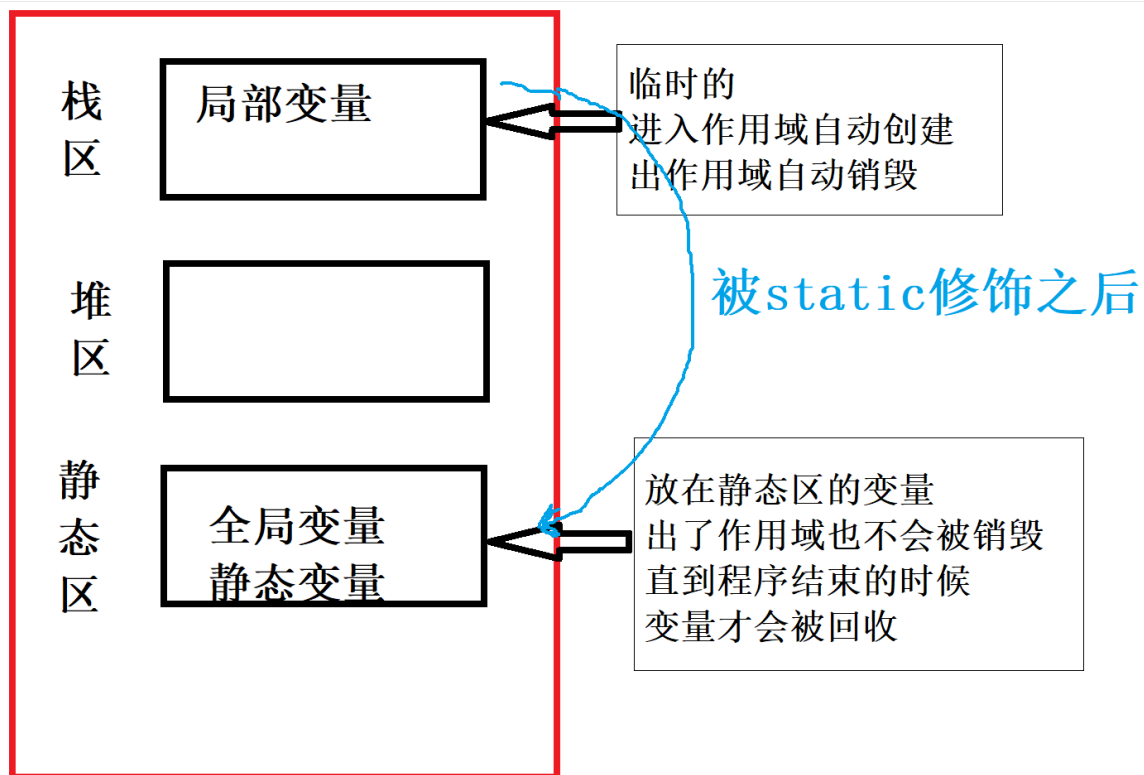
static修饰局部变量

在局部变量前，加上关键字 static，该变量就会被定义为 静态局部变量。

静态局部变量 有以下特点：

静态局部变量在静态区配内存

static修饰局部变量使得变量的存储位置发生了变化，本来局部变量是放在栈区的，被static修饰后，放在内存的静态区，生命周期变得更长了，但是作用域没有发生变化。



CSDN @轴色清风

静态局部变量在编译阶段赋初值，且只赋值一次，在程序运行时它已有初值。

首先我们先看一个普通的局部变量，在程序运行期间：

```
test_2.c  test_1.c  ×
test_1  (全局范围)  test()
1
2  #include <stdio.h>
3  void test()
4  {
5  | int a = 2; 已用时间 <= 2ms
6  | a++;
7  | printf("%d\n", a);
8  | }
9  int main()
10 {
11 | int i = 0;
12 | while (i < 5)
13 | {
14 | | test();
15 | | i++;
16 | }
```

100 % 未找到相关问题 CSDN @轴色清风

```
地址(A): test(...)
查看选项
00007FF72C3721B0 push rbp
00007FF72C3721B2 push rdi
00007FF72C3721B3 sub rsp, 108h
00007FF72C3721BA lea rbp, [rsp+20h]
00007FF72C3721BF lea rcx, [__82B0EE3F_test_1@c (07FF72C381008h)]
00007FF72C3721C6 call __CheckForDebuggerJustMyCode (07FF72C37136Bh)
int a = 2;
00007FF72C3721CB mov dword ptr [a], 2
a++;
00007FF72C3721D2 mov eax, dword ptr [a]
00007FF72C3721D5 inc eax
00007FF72C3721D7 mov dword ptr [a], eax
printf("%d\n", a);
00007FF72C3721DA mov edx, dword ptr [a]
00007FF72C3721DD lea rcx, [string "%d\n" (07FF72C379C24h)]
```

再来看被static修饰的局部变量，
对于语句

```
1 | static int a=2;
```

是没有反汇编代码的，所以这条赋值语句在程序的运行阶段是不运行的。

```
test_1 (全局范围)
1
2 #include <stdio.h>
3 void test()
4 {
5     static int a = 2;
6     a++; 已用时间 <= 2ms
7     printf("%d\n", a);
8 }
9 int main()
10 {
11     int i = 0;
12     while (i < 5)
13     {
14         test();
15         i++;
16     }
```

```
▼ 查看选项
00007FF622D121B2  push     rdi
00007FF622D121B3  sub      rsp, 0E8h
00007FF622D121BA  lea      rbp, [rsp+20h]
00007FF622D121BF  lea      rcx, [__82B0EE3F_test_1@c (07FF622D21008h)]
00007FF622D121C6  call     CheckForDebuggerJustMyCode (07FF622D1136Bh)
static int a = 2;
a++;
00007FF622D121CB  mov      eax, dword ptr [a (07FF622D1C010h)]
00007FF622D121D1  inc      eax
00007FF622D121D3  mov      dword ptr [a (07FF622D1C010h)], eax
printf("%d\n", a);
00007FF622D121D9  mov      edx, dword ptr [a (07FF622D1C010h)]
00007FF622D121DF  lea      rcx, [string "%d\n" (07FF622D19C24h)]
00007FF622D121E6  call     printf (07FF622D11195h)
}
```

CSDN @柚色清风

静态局部变量一般在声明处初始化，如果没有显式初始化，会被程序自动初始化为0；

```
1  #include <stdio.h>
2  int main()
3  {
4      static int a;
5      printf("%d", a); //0
6      return 0;
7  }
```

静态局部变量在程序执行到该对象的声明处时被首次初始化，即以后的函数调用不再进行初始化；

首先，我们来看一下下面的这个简单的程序：

```
1  #include <stdio.h>
2  void test()
3  {
4      int a = 2;
5      a++;
6      printf("%d\n", a);
7  }
8  int main()
9  {
10     int i = 0;
11     while (i < 5)
12     {
13         test();
14         i++;
15     }
16     return 0;
17 }
```

它的运行结果是

```
1  3
2  3
3  3
4  3
5  3
```

简单分析一下，程序从主函数开始，定义整型变量*i*并赋初值0，*i*=0，进入循环体，在循环体中，调用函数test()，程序运行的控制权交到了test()函数中。

在test()函数体内定义了一个局部变量*a*，并赋值为2，系统给局部变量*a* 分配栈内存。

局部变量*a*的作用域是整个函数体，也就是{}内，生命周期为进入这个函数体开始，出这个函数体结束。随着程序出了函数体，系统就会收回

栈内存，局部变量a也相应失效。

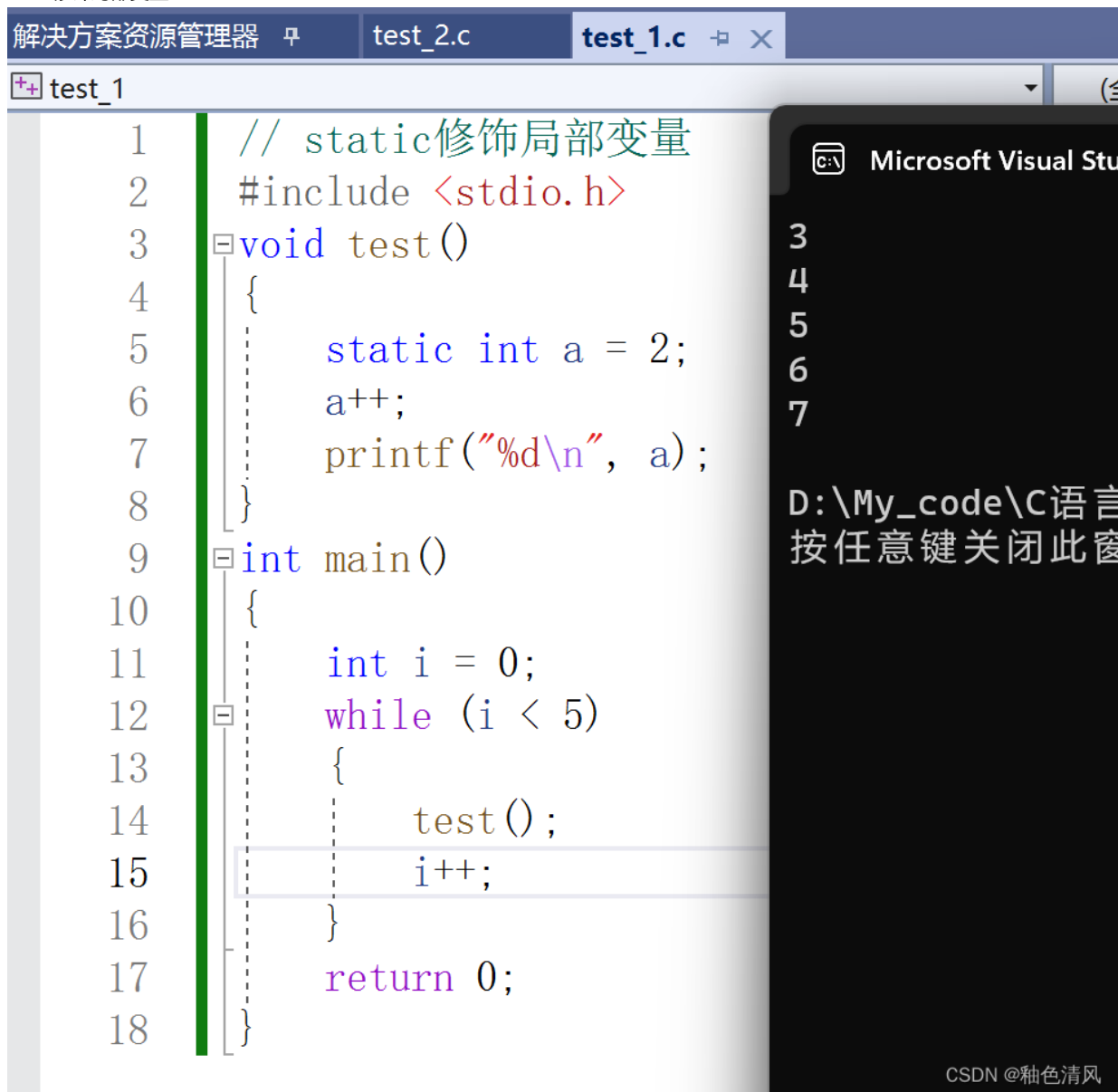
这时程序的控制权又交给了主函数，i自增，变为1，再次进入循环，将程序控制权交给test()函数，系统再次重新为局部变量a分配栈内存，并赋值为2.随着程序出了函数体，系统又会收回栈内存，局部变量a失效。

.....

一直重复上述过程，i=2，i=3，i=4

所以每一次调用函数a的值都为2，所以最终程序的运行结果为3\n3\n3\n3\n3\n。

static修饰局部变量



```
1 // static修饰局部变量
2 #include <stdio.h>
3 void test()
4 {
5     static int a = 2;
6     a++;
7     printf("%d\n", a);
8 }
9 int main()
10 {
11     int i = 0;
12     while (i < 5)
13     {
14         test();
15         i++;
16     }
17     return 0;
18 }
```

这是因为，static修饰局部变量，使得局部变量出了作用域并 不会被销毁，空间不会被回收，下一次进入函数，依然使用的是上次留下的值。

如果反复调用函数，会产生 累积 的效果。

产生这种累计效果的本质也是在于被static修饰的局部变量的位置发生了变化，存储位置由原来的栈区到了静态区，从而导致它的生命周期更长了。

static修饰函数

在函数的返回类型前加上static关键字，函数即被定义为 静态函数。

静态函数 与普通函数不同，它只能在声明它的文件当中可见，不能被其它文件使用。（同static修饰全局变量一样）

下面举个简单的例子做对比：

普通函数：

This screenshot shows the Visual Studio IDE with the 'test_2.c' file open. The function definition for 'Add' is visible, with the function signature 'int Add(int x, int y)' highlighted by a red box. The code is as follows:

```
1
2 int Add(int x, int y)
3 {
4     return x + y;
5 }
6
```

The status bar at the bottom indicates the scope is '(全局范围)' (Global Scope). A watermark 'CSDN @釉色清风' is visible in the bottom right corner.

This screenshot shows the Visual Studio IDE with the 'test_1.c' file open. The code includes the definition of 'Add' from 'test_2.c' and uses it in the 'main' function. The 'extern Add(int x, int y);' line is highlighted by a red box. The code is as follows:

```
1
2 #include <stdio.h>
3 extern Add(int x, int y);
4 int main()
5 {
6     int a=10, b=20, c;
7     c = Add(a, b);
8     printf("%d\n", c);
9     return 0;
10 }
```

The status bar at the bottom indicates the scope is '(全局范围)' (Global Scope). A watermark 'CSDN @釉色清风' is visible in the bottom right corner.

被static修饰的函数:

解决方案资源管理器 test_2.c test_1.c

test_1

```
1
2 static int Add(int x, int y)
3 {
4     return x + y;
5 }
6
```

CSDN @釉色清风

错误列表

整个解决方案 错误 2 警告 0 消息 0 生成 + IntelliSense

代码	说明
LNK1120	1 个无法解析的外部命令
LNK2019	无法解析的外部符号 Add，函数 main 中引用了该符号

CSDN @釉色清风

这和static修饰全局变量一样，本来，函数是具有 外部链接属性 的，在其他源文件内部可以被调用，被static修饰后，外部链接属性就变成了 内部链接属性 ，使得这个函数只能在自己所在的源文件内部使用，其他源文件无法使用。

定义静态函数的好处：

- 静态函数不能被其他文件所用
- 其他文件中可以定义相同名字的函数，不会发生冲突

欢迎大家指出我的问题一起进步！
如果你觉得我写得还不错，不要忘记点个赞哦！ O(≥▽≤)O

文章知识点与官方知识档案匹配，可进一步学习相关知识

C技能树 函数与程序结构 局部变量和全局变量 185238 人正在系统学习中

显示推荐内容

觉得还不错？一键收藏



釉色清风

关注

22



29