2020/04/15_第11课_变量在内存中的存储方式

笔记本:	C
创建时间:	2020/4/15 星期三 15:20
作者:	ileemi
标签:	C语言函数名以及块作用域间的名称粉碎, 静态局部变量, 静态全局变量

自动变量

属于自动存储类别的变量具有自动存储期、块作用域且无链接。默认情况下，声明在块或函数头中的任何变量都属于自动存储类别，关键字：`auto`

外部链接的静态变量

外部链接的静态变量具有文件作用域、外部链接和静态存储器。关键字：`extern`

静态全局变量

特点

该变量在全局数据区分配内存，如果未初始化的静态全局变量会被程序自动初始化位0，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。静态全局变量在声明它的整个文件都是可见的，而在文件之外是不可见的。

静态变量(全局、局部)都在全局数据区分配内存

作用域：文件作用域，只能在当前定义的文件内访问，出了这个文件，就不能访问
生命期：从所处模块的装载到所处模块的卸载

静态变量本质上是受编译器按语法约束的全局变量

extern 外部声明
定义：为标识符分配内存单元
声明：通知编译器该标识符存在，并非书写错误

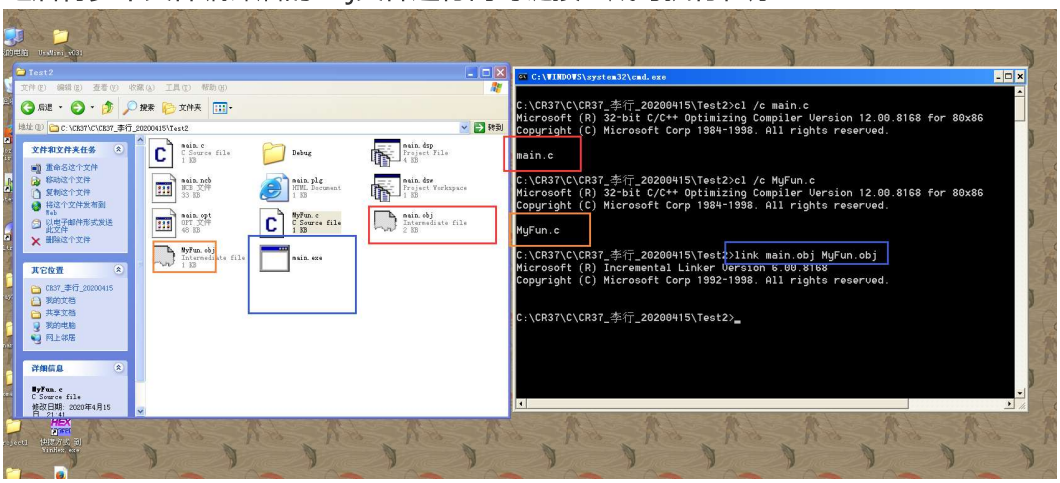
分配内存是 定义的时候做的事

```
int nTest = 0x1833773;  //声明并定义
```

如果定义的全局变量只想在当前文件内使用，就可以在变量名前添加 `static` 关键字进行修饰。

简单模仿IDE编译链接的机制

在一个工程多文件的情况下：
依次编译.c文件
之后将多个文件编译后的obj文件进行同时链接生成可执行程序

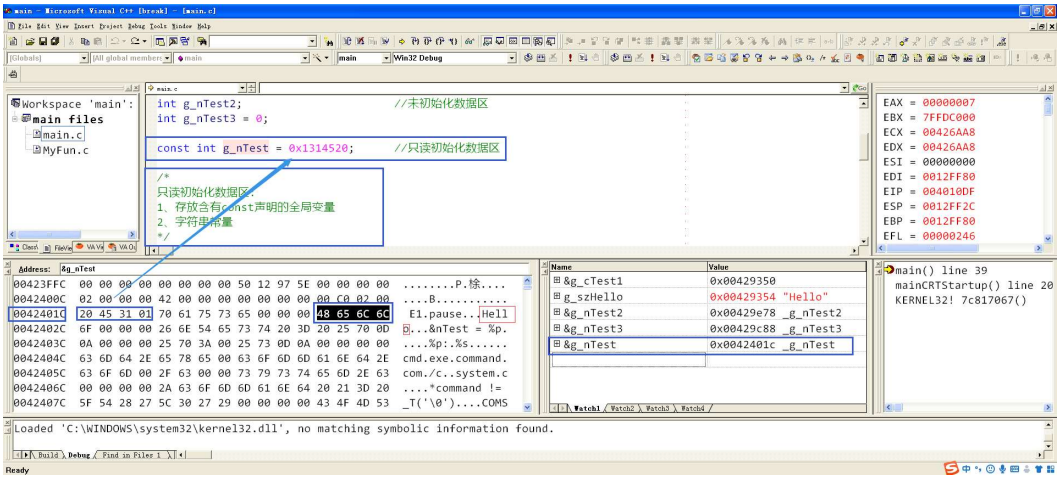


dbug下编译器会将工程下的所有.c或.cpp文件依次执行编译不链接操作

文件作用域：静态全局变量 只能在当前定义的文件内使用

只读数据区只存储：

- 存放有const 声明的全局变量
- 字符串常量
  两者在内存中是相邻的



跨文件访问静态全局变量

由于编译器的限制，在编译阶段，不能在另一个文件内访问另一个文件中的静态全局变量，
检查机制：实际是在链接阶段检查的

编译可以编译，链接阶段就会进行检测

声明的静态全局变量
静态变量编译器的实现：
导出：提供某个符号给其它的模块使用
导入：使用其它的模块中的符号

静态变量的本质是全局变量，作用是

限制导出的机制：C语言，汇编（ASM）

函数也分静态函数和全局函数
静态函数别人不能够调用，全局函数别人可以调用

早期编译器的私有概念
后来才完善这个概念，并逐步发展为其他的面向对象语言，比如C++

强内聚， 低耦合，分责任

公有读，私有写

面向对象核心的两句话：

- 关你什么事
- 关我什么事

静态全局变量具备的面向对象的启蒙

数据结构中的堆和内存管理中的堆概念不一样

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//存储在数据区
//全局作用域
static int g_cTest1 = 0x1833773;     //已初始化数据区，地址：0x00426A30

char g_szHello[8] = "heihei";        //已初始化数据区，地址：0x00426A34

int g_nTest2;                        //未初始化数据区，地址：0x00429E78
int g_nTest3 = 0;                    //未初始化数据区，地址：0x00429E88

const int g_nTest = 0x1314520;       //只读初始化数据区，地址：0x0042401C
/*
只读初始化数据区：
1、存放含有const声明的全局变量
```

```
2、字符串常量
*/

void foo();
int main()
{
    int nTest2 = 0x1314520;
    {
        int nTest = 999;
        printf("%p:",&nTest2);
        printf("%p\r\n", nTest);
    }
    //在块的外部访问块内部变量的地址
    //printf("%p\r\n", nTest);
    printf("&nTest = %p\r\n", (&nTest2)[-1]);    //通过下标运算访问块内静
态局部变量的地址

    printf("%p:",&g_cTest1);
    printf("%p\r\n",g_cTest1);

    printf("%p:",&g_nTest);
    printf("%p\r\n",g_nTest);

    strcpy(g_szHello, "Hello");

    foo();
    system("pause");
    return 0;
}
```

静态局部变量

特点:
全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域中。

作用域:
静态局部变量具有局部作用域，它只被初始化一次，自从第一次被初始化直到程序运行结束都一直存在，它和全局变量的区别在于全局变量对所有的函数都是可见的，而静态局部变量只对定义自己的函数体始终可见。
生命期：和全局变量一样，从模块的装载到模块的卸载

局部静态变量的地址和静态全局变量的地址在内存中是挨在一起

编译器的机制使用了名称粉碎:

```c
/*
首先将静态局部变量当成全局变量，之后变量名和作用域名（函数名）混合出一个新的名字
当使用这个变量的时候，然后就按照这个规则去匹配，当前在哪个作用域，就去匹配这个作用域的名称

通过这样就可以把一个全局变量限制到一个函数内才可以访问了
*/
static int snTeat_Fun = 1250;    //规则匹配snTeat_Fun，变量名_作用域名
void Fun()
{
    //函数作用域
    //声明期和静态全局等同
    //static int snTeat = 1250;
    printf("%p:", &snTeat_Fun);
    printf("%p:", snTeat_Fun);
    snTeat_fooA++;
}
int main()
{
    snTeat_main++;  //同样的规则匹配
    system("pause");
    return 0;
}
/*
不同的编译器对于局部静态变量，不同厂商编译器匹配的思路都一样，规则可以不一样，例如：可以先作用域后名字，也可以先名字后作用域
*/
```

C名称粉碎

名称粉碎：观察.obj文件
名称粉碎和编译器厂商的习惯相关，不属于标准，所以，不同厂商甚至不同的版本规则都不一样。

```c
#include <stdio.h>
#include <stdlib.h>

//全局作用域存储在数据区
static int g_cTest1 = 0x1833773;     //已初始化数据区，地址：0x00426A30

void Fun(int n)
{
    //函数作用域
    //生命期和静态全局等同
```

```
    //_?snTest@?1??Fun@@9@9     没有集成参数序列
    //_?snTest@?2??Fun@@9@9
    //_?snTest@?3??Fun@@9@9

    //关键集成：取的变量名、作用域(当前所处函数)名、层级编号
    //1--作用域层级的编号
    {   //2
        {   //3
            static int snTest = 999;
            printf("%p:", &snTest);
            printf("%p\r\n", snTest);
            snTest++;
        }
    }
}
char g_szHello[8] = "Hello";        //已初始化数据区，地址：0x00426A34
int main()
{
    Fun(1);
    Fun(2);
    Fun(3);
    Fun(2);
    Fun(1);

    system("pause");
    return 0;
}
```

函数名、变量名、层级编号

局部静态变量赋值的时候不产生代码

```
void Fun(int n)
{

    /*
     不产生二进制代码，不进行赋值
   编译期间，这句话是写给编译器看的，编译器只需要通过这段代码知道它的初值是什么，
   然后将这个值算作全局变量，初值就会为已初始化的全局变量，
   然后将其放到数据区的已初始化全局变量区，如果未初始化，就将其放置到未初始化全局变量区
    */
    static int snTest = 999;

    static int snTest = n;  //error C2099: initializer is not a constant，初始化不是常量
```

```
    //.cpp 语法允许局部静态变量初始化为变量
    //.c 语法不允许局部静态变量初始化为变量
    printf("%p:\r\n", &snTest);//产生代码
    printf("%p:\r\n", snTest);  //产生代码


    //C语言 全局变量，静态变量必须赋值为常量，或者不进行赋值，不能赋值为变量
}
```

未初始化区，已初始化区

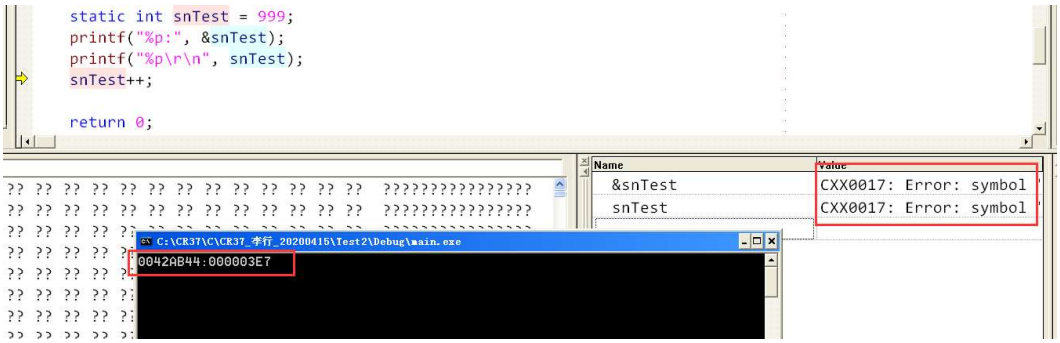
C++名称粉碎

C++名称粉碎的规则：集成了调用约定，函数的返回值，参数的类型

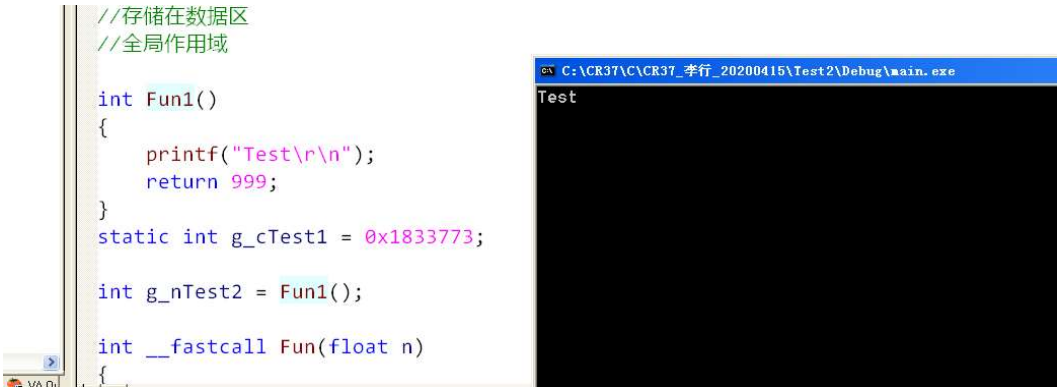
```
int __fastcall Fun(float n)
{
    //C++名称粉碎的规则：集成了调用约定，函数的返回值，参数的类型
    //_?snTest@?1??Fun@@YAXH@Z@4HA       未添加调用约定
    //_?snTest@?1??Fun@@YGXH@Z@4HA --> 添加__stdcall(标准调用约定),可以发现其对应的编码为G
    //_?snTest@?1??Fun@@YAXH@Z@4HA --> 添加__cdecl(C调用约定),可以发现其对应的编码为A,和未添加调用约定前一样
    //_?snTest@?1??Fun@@YIXH@Z@4HA --> 添加__fastcall(快速调用约定），可以发现其对应的编码为I，void类型编码表示：X
    //_?snTest@?1??Fun@@YIHH@Z@4HA --> 在__fastcall(快速调用约定）的同时，将返回值类型改为int，可以发现对应的类型编码为：H(调用约定后面)
    //_?snTest@?1??Fun@@YIHM@Z@4HA --> 接着讲参数int类型转换成float类型，可以发现float对应的编码为：M(在返回值类型后面)
    //_?snTest@?1??Fun@@YINM@Z@4HA --> 同理，将返回值类型转换为double,其对应的编码为：N
    static int snTest = n; //局部静态变量只初始化一次
    printf("%p:", &snTest);
    printf("%p\r\n", snTest);
    snTest++;

    return 0;

}
```

Debug中watch窗口解析名称粉碎的Bug：
在VC++6.0 中使用C++语法，通过编译器对源文件进行编译链接时的watch窗口用的名称粉碎还是C规则，局部静态变量的地址在watch窗口中查看不到，可通过直接打

印输出其地址。



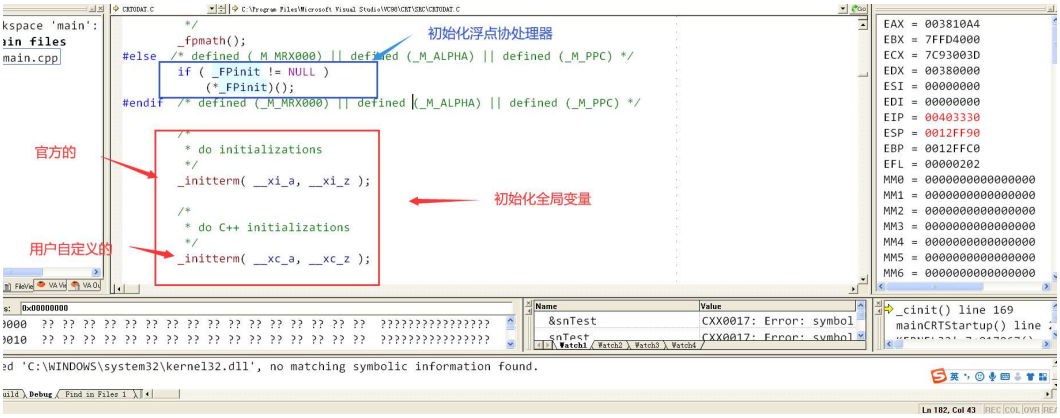
C++的全局变量可以赋值为变量或者赋值为某个函数的返回值，在main函数之前执行

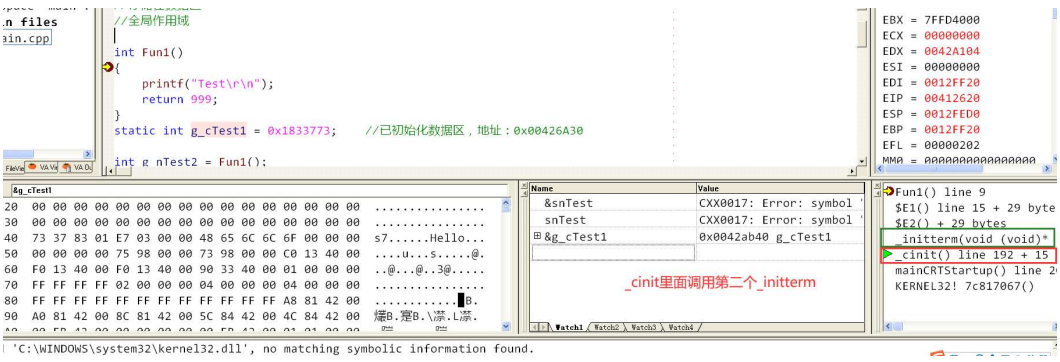


在main函数之前执行代码的方案之一：建立一个C++文件，定义一个全局变量，让其等于一个函数的返回值

_cinit();

- 初始化浮点协处理器
- 初始化全局变量



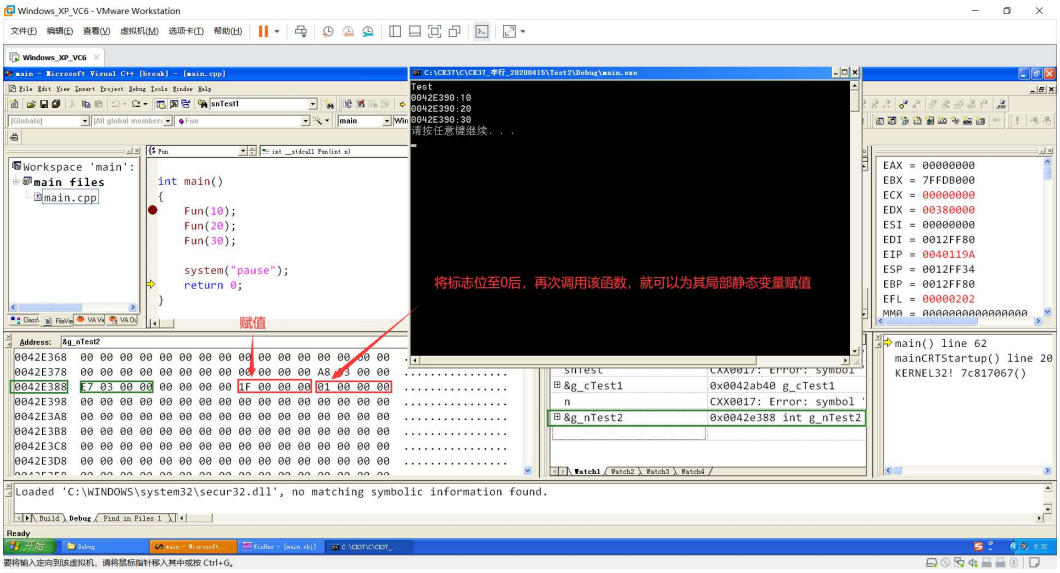


简单说全局变量就是在_cinit()里启动的

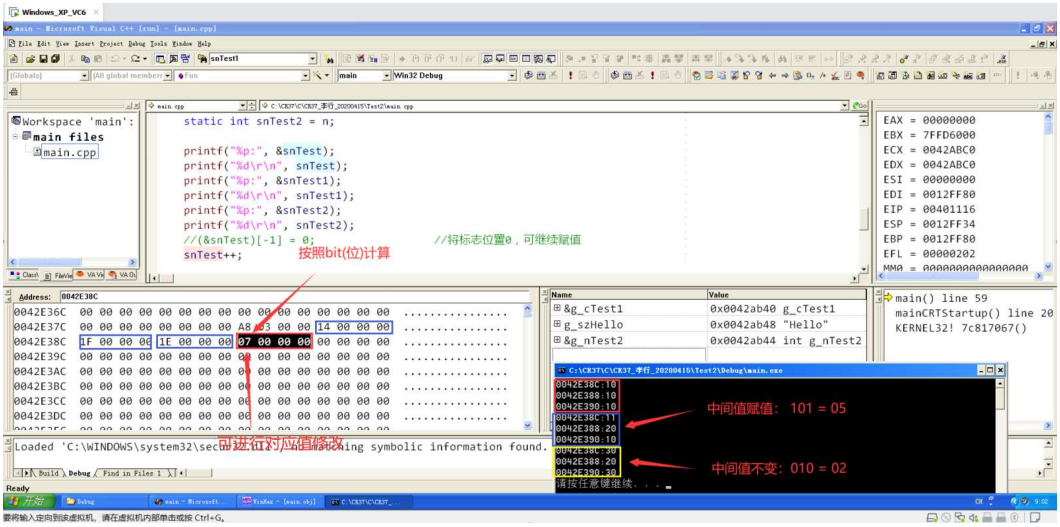
凡是初始化为变量的静态局部变量，变量的全局变量都属于未初始化区的数据，不运行不初始化

静态局部变量当赋初值为常量时不产生代码，当赋值为变量时产生代码
局部静态变量只初始化一次

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//存储在数据区
//全局作用域

int Fun1()
{
    printf("Test\r\n");
    return 999;
}
static int g_cTest1 = 0x1833773; //已初始化数据区，地址：0x00426A30

int g_nTest2 = Fun1();


static int snTest;
char g_isInined = 0; //手动模拟标志位至初值为0
int  Fun(int n)
{
    if (g_isInined == 0) //判断标志位是否为0，为0就赋值
    {
        snTest = n;
        g_isInined = 1; //赋值完成将标志位 至1，表示赋值成功
    }
    static int snTest = n; //局部静态变量只初始化一次
    printf("%p:", &snTest);
    printf("%d\r\n", snTest);
    (&snTest)[+1] = 0; //将标志位 至0，可继续赋值
    //snTest++;

    return 0;

}
int main()
{
    Fun(10); //输出10
    Fun(20); //输出20
    Fun(30); //输出30
    system("pause");
    return 0;
}
```



记录赋值的初始化状态内存中以bit(位)记录



静态全局变量 特点 机制 在什么情况下用

静态局部变量 初始化为常量 什么机制 初始化为变量 什么机制 编译器如何控制越界访问的

当跨界访问编译器是如何检查出来的 .c .cpp

各种机制