2020/04/14_第10课_字符、字符串、变量类别

笔记本:	C
创建时间:	2020/4/14 星期二 15:57
作者:	ileemi
标签:	字符与字符串

字符与字符串

字符串的初始化

```
//String by Zero end
char szBuf[] = "HelloWorld";    //sz 以0位结尾的字符串
char strBuf[] = "HelloWorld";

puts(szBuf);    //自动添加回车

gets(); //支持输入空格 该函数不限定字符长度

fgets(szBuf, 8, stdin);
```

字节对齐:
16位环境，对其单位2
32位环境，4字节对齐
64位环境，对齐单位8
所有变量或者数组或者函数，它们的首地址都是4的倍数
字节对齐，提高访问速度

循环结构的首地址也做到了4的倍数

字符串操作函数

strcmp(字符串1, 字符串2)

函数作用：比较字符串1和字符串2，两个字符串从左至右逐个字符比较（按照ASCII码值的大小进行比较），直至字符不同或者遇见 '\0' 。

如果全部字符都相同，则返回值为0，如果不相同，则返回两个字符串中第一个不相同的字符的ASCII码值得差，即字符串1和字符串2时返回值为整数，反之，返回值为负数。

两个字符串的差值为 0 或者 比较没有字符比较时 或者 比较到字符串的末尾时退出循环

C标准规定反汇两个字符间得差值，而微软规定返回 1，0，-1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void mystrcmp(char szDst[], char szSrc[])
{
    int i = 0;
    //while(*szDst++ = *szSrc++);
    //while(szDst[i] = szSrc[i++]);

    while(szDst[i] == '\0')
    {
        szDst[i] = szSrc[i];
        i++;
    }
}
int main()
{
    int nNum = 1024;
    char szBuf1[] = "Hello World";
    char szBuf2[16] = { 0 };
    mystrcmp(szBuf2, szBuf1);

    nNum = strcmp("aaa", "aaa");
    if (strcmp("aaa", "aaa") == 0)
    {
        printf("str1 == str2\r\n");
    }
    if (strcmp("aaa", "zzz") < 0)
    {
        printf("str1 < str2\r\n");
    }
    if (strcmp("ddd", "aaa") > 0)
    {
        printf("str1 > str2\r\n");
```

```
    }

    printf("%s\r\n", strlwr(szBuf1));
    printf("%s\r\n", strupr(szBuf1));

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

strlwr(字符串)

函数作用：将字符串中的大写字母转换成小写字母

strupr(字符串)

函数作用：将字符串中的小写字母转换成大写字母

变量类别

全局变量存储于数据区
数据区：

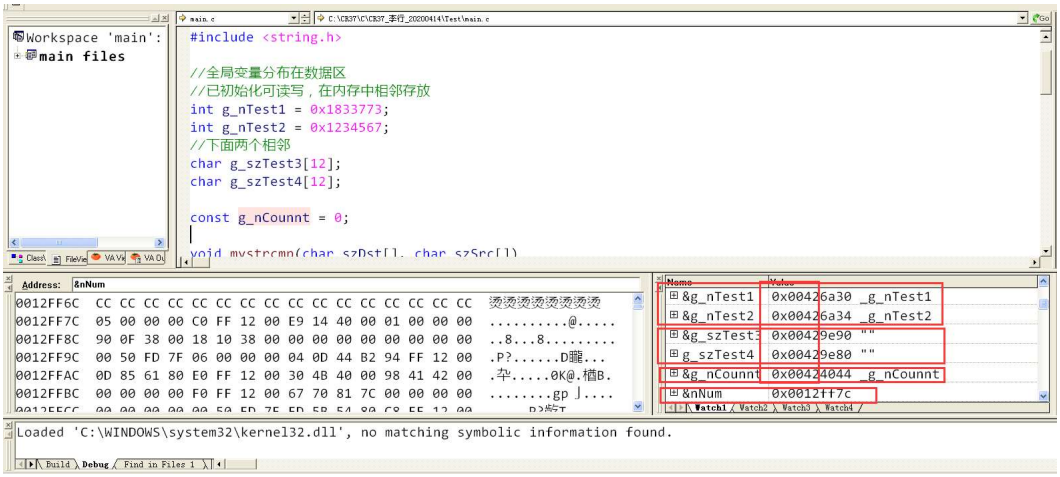
- 未初始化数据区
- 已初始化数据区

<table>
<tr><td>放代码</td><td></td><td>代码区</td><td>可读、可执行</td></tr>
<tr><td>放全局变量，静态变量</td><td rowspan="3">数据区</td><td>未初始化数据区</td><td>可读写</td></tr>
<tr><td></td><td>可读写</td><td rowspan="2">初始化数据区</td></tr>
<tr><td>放常量</td><td>只读</td></tr>
<tr><td>局部变量，参数，返回值等</td><td></td><td>栈</td><td>可读写</td></tr>
<tr><td></td><td></td><td>堆</td><td>可读写</td></tr>
</table>

	默认地址开始区	
	0x00401000 <	代码区
	0x00420000 <	数据区
根据系统环境而定	0x0018ff00（win7）<	栈
		堆

编译器分配变量的原则：

按同内存属性分配



作用域实际上是编译器按照C标准所规定的语法做出了访问限制

```
//全局变量分布在数据区
//已初始化可读写，在内存中相邻存放
int g_nNum1;     //0x00429E74
int g_nNum2;     //0x00429E88

int g_nTest1 = 0x1833773;    //0x00426A30
int g_nTest2 = 0x1234567;    //0x00426A34
//下面两个变量地址相邻
char g_szTest3[12];          //0x00429EA0
char g_szTest4[12];          //0x00429E90

const g_nCounnt = 0;         //0x0042401C


int nTest = 0x1833773;
{
```

```
    int nTest2 = 999;
    printf("%p\r\n", &nTest);
    printf("%p\r\n", &nTest2);
}
    //可通过指针下标访问块内变量的地址
    printf("%p\r\n", (&nTest)[-1]);
```

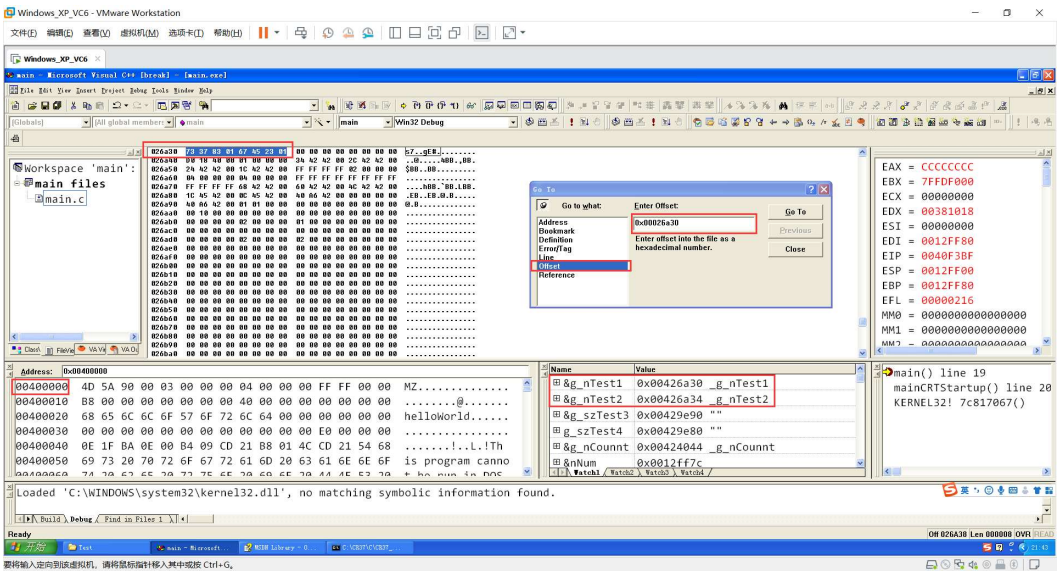
变量的作用域	变量的生命期	举例
块作用域	（特殊的函数）函数的开始到函数的结束	
函数作用域	函数的开始到函数的结束	参数和局部变量
文件作用域	从所处模块装载到所处模块卸载	静态全局变量
进程作用域	程序开始运行到程序结束	全局变量

块作用域的内层可以访问外层，块作用域的外层不可以访问块作用域的内层

全局变量声明周期：从所处模块装载到所处模块卸载

文件作用域：只能在当前定义的文件内访问

在我们创建进程的时候，操作系统会读取对应的可执行文件，然后将可执行文件装载到进程中合适的位置。



全局变量要慎用，应为在软件的设计时它会破坏软件的封装性（会破坏强内聚，低耦合的情况）

扩展知识

数据段的初始化数据区、未初始化的数据区、只读数据区的大小是由编译器根据程序的定义划分具体大小的

PE加载是通过内存文件映射的方式，把可执行文件的数据映射到内存的，没有生长一说，因为全局变量这些都是经编译器编译到可执行文件里面的，是定义好的。

全局变量在编译成可执行文件后，位置大小都已经固定好了，无法再次更改（看操作系统）。一次执行过程中不会发生变化，等下次执行程序，位置就可能发生变化。

操作系统里有一个机制：地址随机化
早期的XP系统没有这个机制，从win7开始有的，当然也可以使用一些手段，可以在XP系统下实现地址随机分配。

整个EXE的地址是操作系统随机分配的，所以其它地址也不可控，但是全局变量，相对于EXE载入到内存的基地址的偏移量是固定的。