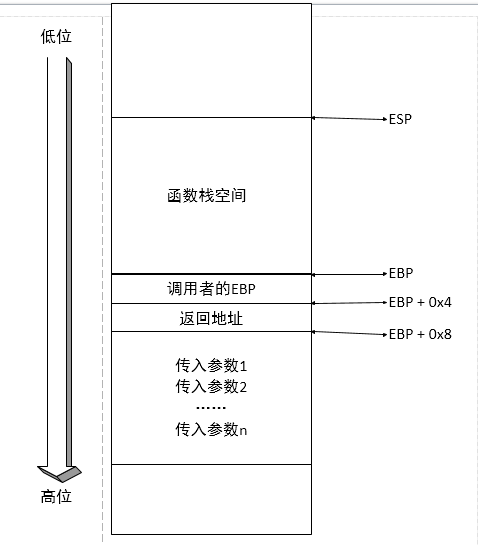
# 深入C++探索--- 二 函数中的变量

## 零、声明

1. 这个系列《深入C++探索》，以汇编和内存角度对C++的一些基本的运行机制进行讨论；
2. 这个系列中出现的汇编代码，都是来自于VS2010的Debug环境下生成的；
3. 如果觉得文章对你有帮助而需要转载，也请阁下能够注明出处；
4. 如果觉得博文对问题的讨论有误，也可以给博主留言。

## 函数的栈空间



这个篇章我们会将第一篇中的栈空间图丰富

## 简单函数

### 1.1.C++代码

函数：

void fun2()

{

int i = 0;

}

### 1.2．汇编解释

void fun2()

{

int i = 0;

012A180E mov dword ptr [i],0 //将0赋值给i所指向的地方

}

在上篇中已经介绍了这个语句，但是ptr[i]到底是在哪？没有具体解释。由于在VS2010中的汇编代码无法了解i的位置。

### 1.3．IDA中的汇编解释

所以我们用IDA在反编译这个程序，再来看一下fun2的汇编代码，有一些常量与前面篇章会有不一致，不过不影响这里介绍的内容，所以也不再解释了。

.text:004117F0 ; void \_\_cdecl fun2() //这里\_\_cdecl是函数的调用方式

//这里不做解释了

.text:004117F0 ?fun2@@YAXXZ proc near ; CODE XREF: fun2(void)j

.text:004117F0

.text:004117F0 var\_CC = byte ptr -0CCh //这里是栈临时区的大小，byte ptr是类型

.text:004117F0 i = dword ptr -8 //i = -8 ,dowrd ptr是类型

.text:004117F0

.text:004117F0 push ebp //与VS2010中类似，略

.text:004117F1 mov ebp, esp //与VS2010中类似，略

.text:004117F3 sub esp, 0CCh //与VS2010中类似，略

.text:004117F9 push ebx //与VS2010中类似，略

.text:004117FA push esi //与VS2010中类似，略

.text:004117FB push edi //与VS2010中类似，略

.text:004117FC lea edi, [ebp+var\_CC] //与VS2010中类似，略

.text:00411802 mov ecx, 33h //与VS2010中类似，略

.text:00411807 mov eax, 0CCCCCCCCh //与VS2010中类似，略

.text:0041180C rep stosd //与VS2010中类似，略

**.text:0041180E mov [ebp+i], 0 //ebp + i = ebp – 8**

**//其实这个变量的存放位置ebp和ebp- 0CCh之间**

.text:00411815 pop edi //与VS2010中类似，略

.text:00411816 pop esi //与VS2010中类似，略

.text:00411817 pop ebx //与VS2010中类似，略

.text:00411818 mov esp, ebp //与VS2010中类似，略

.text:0041181A pop ebp //与VS2010中类似，略

.text:0041181B retn //与VS2010中类似，略

.text:0041181B ?fun2@@YAXXZ endp//略

.text:0041181B //略

.text:0041181B ; ---- //略

### 2.4.C++代码

函数：

void fun2()

{

int x;

int \* p = &x;

int temp = 1;

}

### 2.5.调试解释

我们来查看p所指向的内存块是什么数据。

0x003AFA44：

cc cc cc cc cc cc cc cc 38 fb 3a 00 08 2d bd 00 00 00 00 00 00 00 00 00 00 e0 fd 7e cc cc cc

cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc

前八位都是cc，这个就是函数一开始初始化的时候赋的值。

我们再看一下地址为 0x003AFA44 - C = 0x003AFA38的内存：

44 fa 3a 00 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc 38 fb 3a 00 08 2d bd 00 00 00 00

00 00 00 00 00 00 e0 fd 7e cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc

44 fa 3a 00 这个就是地址 0x003AFA44，其实这个地址就是指针p的地址

我们直接去掉函数的公共汇编代码，首先来看调用处的汇编

012A2D08 push 0Ah //压入参数0xA，就是fun3(10)中的10

012A2D0A call fun3 (12A11CCh) //调用fun3

012A2D0F add esp,4 //平衡栈，这句代码是否出现，还要看函数的调用方式

//函数的调用方式，后面篇章详细介绍，这里就不赘述了

看一下fun3内部的汇编。之后没有特殊说明，函数的公共汇编代码都会去掉用…替代

void fun3(int x)

{

...

x += 1;

012A205E mov eax,dword ptr [x] //将参数x的值赋给eax

//x的位置在ebp-4

-4

012A2061 add eax,1 //等价于eax = eax + 1

012A2064 mov dword ptr [x],eax //等价于x = eax

}

....

## 函数参数的获取

### 3.1.C++代码

函数：

void fun3(int x)

{

x += 1;

}

### 3.2.IDA汇编代码

我们直接去掉函数的公共汇编代码，首先来看调用处的汇编，去除公共汇编代码

.text:00412040 var\_C0 = byte ptr -0C0h

.text:00412040 x = dword ptr 8

…

.text:0041205E mov eax, [ebp+x] //这三句就是实现了x += 1;这里不再赘述了

.text:00412061 add eax, 1

.text:00412064 mov [ebp+x], eax

…

我们来看第一句中的ebp+x和函数的局部变量定义类似，只是这里的x是一个正数，是位于ebp的上部。现在我们再看一下调用的代码。调用的地方，将0Ah压栈，正好在函数fun3的ebp上方。

fun3(10);

00DB2D08 push 0Ah

00DB2D0A call fun3 (0DB11CCh)

00DB2D0F add esp,4

## 三、总结

去除函数的公共汇编代码，整个参数的传入方式和返回值如何返回，整个过程都比较清晰。这里没有去讨论临时变量的存放位置和传入参数如何取得，留在一下篇中详细介绍。