C语言函数调用约定计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022_C_E8_AF_AD_E8_A8_80_E5_87_BD_c97_645857.htm 在C语言中，假设我们有这样的一个函数： int function（int a，int b） 调用时只要用result = function（1，2）这样的方式就可以使用这个函数。但是，当高级语言被编译成计算机可以识别的机器码时，有一个问题就凸现出来：在CPU中，计算机没有办法知道一个函数调用需要多少个、什么样的参数，也没有硬件可以保存这些参数。也就是说，计算机不知道怎么给这个函数传递参数，传递参数的工作必须由函数调用者和函数本身来协调。为此，计算机提供了一种被称为栈的数据结构来支持参数传递。 栈是一种先进后出的数据结构，栈有一个存储区、一个栈顶指针。栈顶指针指向堆栈中第一个可用的数据项（被称为栈顶）。用户可以在栈顶上方向栈中加入数据，这个操作被称为压栈（Push），压栈以后，栈顶自动变成新加入数据项的位置，栈顶指针也随之修改。用户也可以从堆栈中取走栈顶，称为弹出栈（pop），弹出栈后，栈顶下的一个元素变成栈顶，栈顶指针随之修改。 函数调用时，调用者依次把参数压栈，然后调用函数，函数被调用以后，在堆栈中取得数据，并进行计算。函数计算结束以后，或者调用者、或者函数本身修改堆栈，使堆栈恢复原装。 在参数传递中，有两个很重要的问题必须得到明确说明： 当参数个数多于一个时，按照什么顺序把参数压入堆栈 函数调用后，由谁来把堆栈恢复原装 在高级语言中，通过函数调用约定来说明这两个问题。常见的调用约定有： stdcall cdecl fastcall thiscall naked call

stdcall调用约定 stdcall很多时候被称为pascal调用约定，因为pascal是早期很常见的一种教学用计算机程序设计语言，其语法严谨，使用的函数调用约定就是stdcall.在Microsoft C 系列的C/C 编译器中，常常用PASCAL宏来声明这个调用约定，类似的宏还有WINAPI和CALLBACK. stdcall调用约定声明的语法为（以前文的那个函数为例）： int __stdcall function（int a，int b） stdcall的调用约定意味着：1）参数从右向左压入堆栈，2）函数自身修改堆栈 3）函数名自动加前导的下划线，后面紧跟一个@符号，其后紧跟着参数的尺寸 以上述这个函数为例，参数b首先被压栈，然后是参数a，函数调用function（1，2）调用处翻译成汇编语言将变成： push 2 第二个参数入栈 push 1 第一个参数入栈 call function 调用参数，注意此时自动把cs：eip入栈 而对于函数自身，则可以翻译为： push ebp 保存ebp寄存器，该寄存器将用来保存堆栈的栈顶指针，可以在函数退出时恢复 mov ebp，esp 保存堆栈指针 mov eax，[ebp 8H] 堆栈中ebp指向位置之前依次保存有ebp，cs：eip，a，b，ebp 8指向a add eax，[ebp 0CH] 堆栈中ebp 12处保存了b mov esp，ebp 恢复esp pop ebp ret 8 而在编译时，这个函数的名字被翻译成_function@8 注意不同编译器会插入自己的汇编代码以提供编译的通用性，但是大体代码如此。其中在函数开始处保留esp到ebp中，在函数结束恢复是编译器常用的方法。 从函数调用看，2和1依次被push进堆栈，而在函数中又通过相对于ebp（即刚进函数时的堆栈指针）的偏移量存取参数。函数结束后，ret 8表示清理8个字节的堆栈，函数自己恢复了堆栈。 cdecl调用约定 cdecl调用约定又称为C调用约定，是C语言缺省的调用约定，它的定义语法是： int function

（int a，int b） //不加修饰就是C调用约定 int __cdecl function（int a，int b）//明确指出C调用约定 在写本文时，出乎我的意料，发现cdecl调用约定的参数压栈顺序是和stdcall是一样的，参数首先由有向左压入堆栈。所不同的是，函数本身不清理堆栈，调用者负责清理堆栈。由于这种变化，C调用约定允许函数的参数的个数是不固定的，这也是C语言的一大特色。对于前面的function函数，使用 cdecl后的汇编码变成：调用处 push 1 push 2 call function add esp，8 注意：这里调用者在恢复堆栈 被调用函数_function处 push ebp 保存ebp寄存器，该寄存器将用来保存堆栈的栈顶指针，可以在函数退出时恢复 mov ebp，esp 保存堆栈指针 mov eax，[ebp 8H] 堆栈中ebp指向位置之前依次保存有ebp，cs：eip，a，b，ebp 8指向a add eax，[ebp 0CH] 堆栈中ebp 12处保存了b mov esp，ebp 恢复esp pop ebp ret 注意，这里没有修改堆栈 MSDN中说，该修饰自动在函数名前加前导的下划线，因此函数名在符号表中被记录为_function，但是我在编译时似乎没有看到这种变化。 由于参数按照从右向左顺序压栈，因此最开始的参数在最接近栈顶的位置，因此当采用不定个数参数时，第一个参数在栈中的位置肯定能知道，只要不定的参数个数能够根据第一个后者后续的明确的参数确定下来，就可以使用不定参数，例如对于CRT中的sprintf函数，定义为： int sprintf（char* buffer，const char* format，……） 由于所有的不定参数都可以通过format确定，因此使用不定个数的参数是没有问题的。 fastcall fastcall调用约定和stdcall类似，它意味着： 函数的第一个和第二个DWORD参数（或者尺寸更小的）通过ecx和edx传递，其他参数通过从右向左的顺序压栈 被调用函数清理堆栈

函数名修改规则同stdcall 其声明语法为：int fastcall function（int a，int b） thiscall thiscall是唯一一个不能明确指明的函数修饰，因为thiscall不是关键字。它是C 类成员函数缺省的调用约定。由于成员函数调用还有一个this指针，因此必须特殊处理，thiscall意味着： 参数从右向左入栈 如果参数个数确定，this指针通过ecx传递给被调用者；如果参数个数不确定，this指针在所有参数压栈后被压入堆栈。 对参数个数不定的，调用者清理堆栈，否则函数自己清理堆栈。 