

## **Calling Conventions Part I (Passing Integral Arguments)**

**Georg M. Penn**

Copyright (c) 2007 Georg M Penn.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

## Table of Contents

Calling Conventions Part I (Passing Integral Arguments) .....	1
The C Convention ( <code>_cdecl</code> ) .....	4
Microsoft Visual C++ 7: .....	4
Borland C++ 5.5.....	5
GCC 4.1.2.....	6
The Pascal Calling Convention (PASCAL) .....	7
Microsoft Visual C++ 7 .....	7
Borland C++ 5.5.....	8
GCC 4.1.2.....	9
The Standard Convention ( <code>_stdcall</code> ).....	9
Microsoft Visual C++ 7 .....	9
Borland C++ 5.5.....	10
GCC 4.1.2.....	10
The Fastcall convention ( <code>_fastcall</code> ) .....	11
Microsoft Visual C++ 7 .....	11
Borland C++ 5.5.....	12
GCC 4.1.2.....	13
The thiscall Calling Convention.....	14
Microsoft Visual C++ 7 .....	14
Borland C++ 5.5.....	14
GCC 4.1.2.....	14
The Default Convention.....	14
Microsoft Visual C++ 7 .....	15
Borland C++ 5.5.....	16
GCC 4.1.2.....	17
Conclusion .....	18
References .....	18
GNU Free Documentation License .....	19
ADDENDUM: How to use this License for your documents.....	24

## Abstract

Identifying the calling convention used by a compiler is a key part of the analysis of disassembled programs. Arguments can be passed to a function via the stack, via registers, and via the stack and registers simultaneously. Also arguments can be passed either by value or by reference. In the first case a copy of the corresponding variable is passed to the function; in the second case a pointer is passed. Another key issue of argument passing is who is responsible for clearing the stack. This can either be done by the caller (the calling function) or by the callee (the function which is called).

This document is a glimpse of the most popular calling convention used in a 32-Bit Intel based environment with respect to different C/C++ compilers. The following compilers and operating systems were used for analysis:

Compiler <sup>i</sup>	Operating System
Microsoft Visual C++ 7 <sup>ii</sup>	Windows XP Professional SP2
Borland C++ 5.5	Windows XP Professional SP2
GCC 4.1.2	Fedora 7

<sup>i</sup> Optimization was turned off for each compiler. However, this does not affect how arguments are passed for a specific calling convention.

<sup>ii</sup> All source code was compiled in debug mode and without /RTC1 (Runtime Checks) for simplicity.

For disassembling the examples I used IDA 5.2 for Windows and Linux. I slightly reformatted IDA's output for better readability.

It has to be noted that this document does not take into account the passing of floating point arguments, as it differs completely from passing integral data types.

## The C Convention (`__cdecl`)

The C convention directs you to push arguments onto the stack from right to left in order in which they are declared. It is the responsibility of the caller (calling function) to clear the stack. The `this` pointer (in C++ programs) is transferred via the stack last. The `__cdecl` calling convention creates larger executables than the `__stdcall` because it requires each function call to include stack cleanup code.

The names of the functions that obey to the C convention are preceded with the “\_” character, automatically inserted by the compiler.

This is the default calling convention of the Microsoft C and C++ compiler as well as of the GCC. Borland also states in the help file provided with the Borland C++ compiler that it is using the `__cdecl` calling convention by default.

## Microsoft Visual C++ 7:

Demonstration of the <code>__cdecl</code> calling convention using Microsoft Visual C++ 7 and IDA 5.2	
<pre>#include &lt;stdio.h&gt; #include &lt;string.h&gt;  int __cdecl func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main()</pre>	<pre>main proc near     push    ebp     mov     ebp, esp     sub     esp, 40h     push    ebx     push    esi     push    edi     push    offset aHello00z13     push    7</pre>

<pre> { printf("%x\n", func(6, 7, "Hello w00z13")); return 0; } </pre>	<pre> push 6 call j_func add esp, 0Ch push eax push offset asc_42401C ; "%x\n" call j_printf add esp, 8 xor eax, eax pop edi pop esi pop ebx mov esp, ebp pop ebp retn  main endp  func proc near  arg_0= dword ptr 8 arg_4= dword ptr 0Ch arg_8= dword ptr 10h  push ebp mov ebp, esp sub esp, 40h push ebx push esi push edi mov esi, [ebp+arg_0] add esi, [ebp+arg_4] mov eax, [ebp+arg_8] push eax call j_strlen add esp, 4 add eax, esi pop edi pop esi pop ebx mov esp, ebp pop ebp retn  func endp </pre>
--	--

## Borland C++ 5.5

<b>Demonstration of the <code>__cdecl</code> calling convention using Borland C++ 5.5 and IDA 5.2</b>	
<pre> #include &lt;stdio.h&gt; #include &lt;string.h&gt;  int __cdecl func(int a, int b, char* c) { return (a + b + strlen(c)); }  int main() { printf("%x\n", func(6, 7, "Hello w00z13")); return 0; } </pre>	<pre> _main proc near  argc= dword ptr 8 argv= dword ptr 0Ch envp= dword ptr 10h  push ebp mov ebp, esp push offset aHelloW00z13 push 7 ; int push 6 ; int call sub_401150 add esp, 0Ch push eax push offset format ; "%x\n" call _printf add esp, 8 xor eax, eax pop ebp retn  _main endp  sub_401150 proc near  arg_0= dword ptr 8 arg_4= dword ptr 0Ch s= dword ptr 10h  push ebp mov ebp, esp push [ebp+s] ; s call _strlen pop ecx </pre>

	<pre> mov     edx, [ebp+arg_0] add     edx, [ebp+arg_4] add     eax, edx pop     ebp <b>retn</b> sub_401150 endp         </pre>
--	---

## GCC 4.1.2

Demonstration of the <code>__cdecl</code> calling convention using GCC 4.1.2 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt;  #define __cdecl __attribute__((cdecl))  int __cdecl func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main(int argc, char* argv[]) {     printf("%x\n", func(6, 7, "Hello w00z13"));     exit(EXIT_SUCCESS); }         </pre>	<pre> main proc near     var_20= dword ptr -20h     var_1C= dword ptr -1Ch     var_18= dword ptr -18h     arg_0= byte ptr 4      lea     ecx, [esp+arg_0]     and     esp, 0FFFFFFF0h     push   dword ptr [ecx-4]     push   ebp     mov     ebp, esp     push   ecx     sub     esp, 14h     mov     [esp+20h+var_18], offset aHello00z13     mov     [esp+20h+var_1C], 7     mov     [esp+20h+var_20], 6     call   func     mov     [esp+20h+var_1C], eax     mov     [esp+20h+var_20], offset asc_804853D     call   _printf     mov     [esp+20h+var_20], 0     call   _exit main endp  func proc near     var_8= dword ptr -8     arg_0= dword ptr 8     arg_4= dword ptr 0Ch     arg_8= dword ptr 10h      push   ebp     mov     ebp, esp     push   edi     sub     esp, 4     mov     eax, [ebp+arg_4]     add     eax, [ebp+arg_0]     mov     edx, eax     mov     eax, [ebp+arg_8]     mov     ecx, 0FFFFFFFh     mov     [ebp+var_8], eax     mov     eax, 0     mov     edi, [ebp+var_8]     repne scasb     mov     eax, ecx     not     eax     sub     eax, 1     lea     eax, [edx+eax]     add     esp, 4     pop     edi     pop     ebp     <b>retn</b> func endp         </pre>

From the disassembled listings above we see that each compiler obeys to the rules of the `__cdecl` calling convention. All arguments are passed to the callee through the stack from right to left, and charge the caller with the clearance of the stack. This is exactly the behavior we would expect.

Interestingly enough the GCC compiler prefers to push the arguments onto the stack using the `mov` instruction rather than a `push`. It also addresses the stack pointer directly using the `ESP` register.

## The Pascal Calling Convention (`PASCAL`)

The Pascal calling convention directs you to sent arguments to the stack from left to right in the order in which they are declared. It is the responsibility of the callee (the called function) to clean the stack. Nowadays the `PASCAL` keyword is regarded to be out-of-date, and has gone out of use.

Microsoft Visual C++ no longer supports the `PASCAL` call type. Instead it uses the similar `WINAPI` call type defined in the `windef.h` file, which is included by `windows.h`.

from `windef.h`:

```
.....
#ifdef _MAC
#define CALLBACK      PASCAL
#define WINAPI        CDECL
#define WINAPIV       CDECL
#define APIENTRY      WINAPI
#define APIPRIVATE    CDECL
#ifdef _68K_
#define PASCAL        __pasca1
#else
#define PASCAL
#endif
#elif (_MSC_VER >= 800) || defined(_STDCALL_SUPPORTED)
#define CALLBACK      __stdcall
#define WINAPI        __stdcall
#define WINAPIV       __cdecl
#define APIENTRY      WINAPI
#define APIPRIVATE    __stdcall
#define PASCAL        __stdcall
#else
#define CALLBACK
#define WINAPI
#define WINAPIV
#define APIENTRY      WINAPI
#define APIPRIVATE
#define PASCAL        pasca1
#endif
.....
```

## Microsoft Visual C++ 7

Demonstration of the <code>PASCAL</code> calling convention using Microsoft Visual C++ 7 and IDA 5.2	
<pre>#include &lt;windows.h&gt; /* for PASCAL */ #include &lt;stdio.h&gt; #include &lt;string.h&gt;  int PASCAL func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main() {     printf("%x\n", func(6, 7, "Hello w00z13"));     return 0; }</pre>	<pre>main proc near     push    ebp     mov     ebp, esp     sub     esp, 40h     push    ebx     push    esi     push    edi     push    offset aHello00z13     push    7     push    6     call   j_func     push    eax     push    offset asc_42401C ; "%x\n"     call   j_printf     add     esp, 8     xor     eax, eax     pop     edi     pop     esi     pop     ebx     mov     esp, ebp     pop     ebp     retn  main endp  func proc near     arg_0= dword ptr 8</pre>

	<pre> arg_4= dword ptr 0Ch arg_8= dword ptr 10h  push    ebp mov     ebp, esp sub     esp, 40h push    ebx push    esi push    edi mov     esi, [ebp+arg_0] add     esi, [ebp+arg_4] mov     eax, [ebp+arg_8] push    eax call   j_strlen add     esp, 4 add     eax, esi pop     edi pop     esi pop     ebx mov     esp, ebp pop     ebp retn   0Ch func endp </pre>
--	--

From the argument passing in the disassembly listing above we can spot that the code produced from the Microsoft Visual C++ 7 compiler does not use the `__pascal` calling convention. This comes as no surprise as the `PASCAL` macro in `windef.h` is defined as `#define PASCAL __stdcall`. We therefore simply deal with the standard calling convention (see next section).

Microsoft specific details for obsolete calling conventions can be found at:  
[http://msdn2.microsoft.com/en-us/library/wda6h6df\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/wda6h6df(VS.80).aspx)

## Borland C++ 5.5

Demonstration of the <code>__pascal</code> calling convention using Borland C++ 5.5 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;string.h&gt;  int __pascal func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main() {     printf("%x\n", func(6, 7, "Hello w00z13"));     return 0; } </pre>	<pre> _main proc near      argc= dword ptr 8     argv= dword ptr 0Ch     envp= dword ptr 10h      push    ebp     mov     ebp, esp     push    6                ; int     push    7                ; int     push    offset aHelloW00z13     call   sub_401150     push    eax     push    offset format    ; "%x\n"     call   _printf     add     esp, 8     xor     eax, eax     pop     ebp     retn  _main endp  sub_401150 proc near      s= dword ptr 8     arg_4= dword ptr 0Ch     arg_8= dword ptr 10h      push    ebp     mov     ebp, esp     push    [ebp+s]          ; s     call   _strlen     pop     ecx     mov     edx, [ebp+arg_8]     add     edx, [ebp+arg_4]     add     eax, edx     pop     ebp     retn   0Ch sub_401150 endp </pre>

As we can clearly see from the code generated above, using the `__pascal` calling convention, the parameters are pushed on the stack from left to right and the callee is responsible for clearing the stack.

## GCC 4.1.2

To my knowledge GCC does not support the `PASCAL` calling convention, thus no examples are given for GCC in this section.

## The Standard Convention (`__stdcall`)

The standard calling convention is a hybrid of the C and Pascal convention. Arguments are pushed onto the stack from right to left as is the case with the C convention. However, the callee (called function) is responsible for clearing the stack. The `this` pointer (in C++ programs) is transferred via the stack last.

The names of the functions are preceded by the `_` (underscore) character and followed by the `@` character and the number of bytes (in decimal) in the argument list. Therefore, the function declared as `int func(int a, double b)` is decorated as follows: `_func@12`.

The standard calling convention is also used by Microsoft's WinAPI functions.

## Microsoft Visual C++ 7

Demonstration of the <code>__stdcall</code> calling convention using Microsoft Visual C++ 7 and IDA 5.2	
<pre>#include &lt;stdio.h&gt; #include &lt;string.h&gt;  int __stdcall func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main() {     printf("%x\n", func(6, 7, "Hello w00z13"));     return 0; }</pre>	<pre>main proc near     push    ebp     mov     ebp, esp     sub     esp, 40h     push    ebx     push    esi     push    edi     push    offset aHello00z13     push    7     push    6     call   j_func     push    eax     push    offset asc_42301c ; "%x\n"     call   j_printf     add     esp, 8     xor     eax, eax     pop     edi     pop     esi     pop     ebx     mov     esp, ebp     pop     ebp     retn  main endp  func proc near      arg_0= dword ptr 8     arg_4= dword ptr 0ch     arg_8= dword ptr 10h      push    ebp     mov     ebp, esp     sub     esp, 40h     push    ebx     push    esi     push    edi     mov     esi, [ebp+arg_0]     add     esi, [ebp+arg_4]     mov     eax, [ebp+arg_8]     push    eax     call   j_strlen     add     esp, 4     add     eax, esi     pop     edi     pop     esi     pop     ebx</pre>

	<pre> mov     esp, ebp pop     ebp retn   0Ch func endp         </pre>
--	--

## Borland C++ 5.5

Demonstration of the <code>__stdcall</code> calling convention using Borland C++ 5.5 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;string.h&gt;  int __stdcall func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main() {     printf("%x\n", func(6, 7, "Hello w00z13"));     return 0; }         </pre>	<pre> _main proc near      argc= dword ptr 8     argv= dword ptr 0Ch     envp= dword ptr 10h      push     ebp     mov     ebp, esp     push   offset aHello00z13     push   7 ; int     push   6 ; int     call   sub_401150     push   eax     push   offset format ; "%x\n"     call   _printf     add    esp, 8     xor    eax, eax     pop    ebp     retn  _main endp  sub_401150 proc near      arg_0= dword ptr 8     arg_4= dword ptr 0Ch     s= dword ptr 10h      push     ebp     mov     ebp, esp     push   [ebp+s] ; s     call   _strlen     pop    ecx     mov    edx, [ebp+arg_0]     add    edx, [ebp+arg_4]     add    eax, edx     pop    ebp     retn   0Ch sub_401150 endp         </pre>

## GCC 4.1.2

Demonstration of the <code>__stdcall</code> calling convention using GCC 4.1.2 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt;  #define __stdcall __attribute__((stdcall))  int __stdcall func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main(int argc, char* argv[]) {     printf("%x\n", func(6, 7, "Hello w00z13"));     exit(EXIT_SUCCESS); }         </pre>	<pre> main proc near      var_20= dword ptr -20h     var_1C= dword ptr -1Ch     var_18= dword ptr -18h     arg_0= byte ptr 4      lea    ecx, [esp+arg_0]     and    esp, 0FFFFFFFh     push  dword ptr [ecx-4]     push  ebp     mov   ebp, esp     push  ecx     sub   esp, 14h     mov   [esp+20h+var_18], offset aHello00z13     mov   [esp+20h+var_1C], 7     mov   [esp+20h+var_20], 6     call  func     sub   esp, 0Ch     mov   [esp+20h+var_1C], eax         </pre>

	<pre> mov [esp+20h+var_20], offset asc_804853D call _printf mov [esp+20h+var_20], 0 call _exit main endp  func proc near  var_8= dword ptr -8 arg_0= dword ptr 8 arg_4= dword ptr 0Ch arg_8= dword ptr 10h  push ebp mov ebp, esp push edi sub esp, 4 mov eax, [ebp+arg_4] add eax, [ebp+arg_0] mov edx, eax mov eax, [ebp+arg_8] mov ecx, 0FFFFFFFh mov [ebp+var_8], eax mov eax, 0 cld mov edi, [ebp+var_8] repne scasb mov eax, ecx not eax sub eax, 1 lea eax, [edx+eax] add esp, 4 pop edi pop ebp retn 0Ch func endp </pre>
--	---

No surprises here. As expected all three compilers sticks to the rules and pushes the arguments onto the stack from right to left and the callee is in charge of clearing the stack.

Again the GCC compiler uses the `mov` instruction rather than a `push` to pass arguments.

### The *Fastcall* convention (`__fastcall`)

The `__fastcall` convention dictates that you transfer the arguments via registers if possible. Compilers from Microsoft and Borland support the `__fastcall` keyword, but they interpret it differently.

The names of the functions that adhere to the `__fastcall` convention are preceded by the “@” character, which is automatically inserted by the compiler. The number of bytes (in decimal) in the parameter list (including the register parameters) is suffixed to the function names (e.g. `@MyFunc@20`)

### Microsoft Visual C++ 7

Demonstration of the <code>__fastcall</code> calling convention using Microsoft Visual C++ 7 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;string.h&gt;  int __fastcall func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main() {     printf("%x\n", func(6, 7, "Hello w00z13"));     return 0; } </pre>	<pre> main proc near push ebp mov ebp, esp sub esp, 40h push ebx push esi push edi push offset aHelloW00z13 mov edx, 7 mov ecx, 6 call j_func push eax push offset asc_42401C ; "%x\n" call j_printf add esp, 8 xor eax, eax pop edi pop esi </pre>

	<pre> pop     ebx mov     esp, ebp pop     ebp retn main endp  func proc near      var_8= dword ptr -8     var_4= dword ptr -4     arg_0= dword ptr 8      push    ebp     mov     ebp, esp     sub     esp, 48h     push    ebx     push    esi     push    edi     mov     [ebp+var_8], edx     mov     [ebp+var_4], ecx     mov     esi, [ebp+var_4]     add     esi, [ebp+var_8]     mov     eax, [ebp+arg_0]     push    eax     call   j_strlen     add     esp, 4     add     eax, esi     pop     edi     pop     esi     pop     ebx     mov     esp, ebp     pop     ebp     retn   4 func endp </pre>
--	--

As mentioned above arguments are transferred to the calling function via registers if possible. The first two `DWORD` or smaller arguments are passed in `ECX` and `EDX` registers; all other arguments are passed from right to left via the stack. The called function is responsible for clearing the stack and pops the arguments from the stack.

We can see from the disassemble listing above that the Microsoft Visual C++ compiler stores the arguments passed to the function in `ECX` and `EDX` in local variables (`mov [ebp+var_8], ecx` and `mov [ebp+var_14], edx`). This seems rather stupid. After all, addressing the memory negates all the benefits of the `__fastcall` convention. However, this behavior can be circumvented by the use of compiler optimization flags.

## Borland C++ 5.5

Demonstration of the <code>__fastcall</code> calling convention using Borland C++ 5.5 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;string.h&gt;  int __fastcall func(int a, int b, char* c) {     return (a + b + strlen(c)); }  int main() {     printf("%x\n", func(6, 7, "Hello w00z13"));     return 0; } </pre>	<pre> _main proc near      argc= dword ptr 8     argv= dword ptr 0Ch     envp= dword ptr 10h      push    ebp     mov     ebp, esp     mov     ecx, offset aHelloW00z13     mov     edx, 7     mov     eax, 6     call   sub_401150     push    eax     push    offset format ; "%x\n"     call   _printf     add     esp, 8     xor     eax, eax     pop     ebp     retn _main endp  sub_401150 proc near push    ebp mov     ebp, esp push    ebx </pre>

	<pre> push esi push edi mov edi, ecx mov esi, edx mov ebx, eax push edi ; s call _strlen pop ecx add esi, ebx add eax, esi pop edi pop esi pop ebx pop ebp retn sub_401150 endp </pre>
--	--

The arguments are evaluated from left to right and the first three arguments are passed through the EAX, EDX and ECX register, if possible. All other arguments are pushed onto the stack (also from left to right).

## GCC 4.1.2

Demonstration of the <code>__fastcall</code> calling convention using GCC 4.1.2 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt; #define __fastcall __attribute__((fastcall)) int __fastcall func(int a, int b, char* c) {     return (a + b + strlen(c)); } int main(int argc, char* argv[]) {     printf("%x\n", func(6, 7, "Hello w00z13"));     exit(EXIT_SUCCESS); } </pre>	<pre> main proc near     var_20= dword ptr -20h     var_1C= dword ptr -1Ch     arg_0= byte ptr 4     lea ecx, [esp+arg_0]     and esp, 0FFFFFFFh     push dword ptr [ecx-4]     push ebp     mov ebp, esp     push ecx     sub esp, 14h     mov [esp+20h+var_20], offset aHello00z13 ; "Hello w00z13"     mov edx, 7     mov ecx, 6     call func     sub esp, 4     mov [esp+20h+var_1C], eax     mov [esp+20h+var_20], offset asc_804853D ; "%x\n"     call _printf     mov [esp+20h+var_20], 0     call _exit main endp  func proc near     var_10= dword ptr -10h     var_C= dword ptr -0Ch     var_8= dword ptr -8     arg_0= dword ptr 8     push ebp     mov ebp, esp     push edi     sub esp, 0Ch     mov [ebp+var_8], ecx     mov [ebp+var_C], edx     mov eax, [ebp+var_C]     add eax, [ebp+var_8]     mov edx, eax     mov eax, [ebp+arg_0]     mov ecx, 0FFFFFFFh     mov [ebp+var_10], eax     mov eax, 0     cld     mov edi, [ebp+var_10]     repne scasb     mov eax, ecx     not eax     sub eax, 1     lea eax, [edx+eax]     add esp, 0Ch </pre>

	<pre> pop     edi pop     ebp retn    4 func endp </pre>
--	--

GCC's implementation of the `__fastcall` is similar to Microsoft's. On the Intel 386, the `__fastcall` attribute causes the compiler to pass the first argument (if of integral type) in the `ECX` register and the second argument (if of integral type) in the `EDX` register. Subsequent and other typed arguments are passed on the stack. The called function will pop the arguments off the stack. If the number of arguments is variable all arguments are pushed on the stack. Arguments are passed from right to left.

## ***The thiscall Calling Convention***

This calling convention is used for calling C++ non-static member functions. There are two primary versions of `thiscall` used depending on the compiler and whether or not the function uses variable arguments.

The `thiscall` calling convention can only be explicitly specified on Microsoft Visual C++ 2005 (VC8) and later. On any other compiler `__thiscall` is not a keyword.

Because this calling convention applies only to C++, there is no C name decoration scheme.

Examples are given in the section "The Default Convention".

## **Microsoft Visual C++ 7**

For the Microsoft Visual C++ 7 compiler this is the default calling convention used by C++ member functions that do not use variable arguments. Under `thiscall`, the callee cleans the stack. Arguments are pushed on the stack from right to left, with the `this` pointer being passed via the `ECX` register. The `thiscall` calling convention cannot be explicitly specified in a program, because `thiscall` is not a keyword (it is however a keyword for VC8).

Member functions with variable arguments use the `__cdecl` calling convention. All function arguments are pushed on the stack, with the `this` pointer placed on the stack last.

## **Borland C++ 5.5**

Although some sources state that by default the Borland C++ 5.5 compiler uses the `EAX` register to pass the `this` pointer of a class instance to the member function, I could not render this to be true. Using the Borland C++ compiler 5.5 and IDA 5.2 shows that by default the `this` pointer is passed through the stack. All other arguments are also pushed on the stack from right to left and the clearance of the stack is performed by the calling function.

## **GCC 4.1.2**

For the GCC compiler, the `thiscall` calling conventions is almost identical to `__cdecl`. The calling function is in charge of clearing the stack, and the parameters are passed from right to left. The difference is the addition of the `this` pointer, which is pushed onto the stack last, as if it were the first parameter of the function prototype.

This is actually the same behavior we already know from the Borland C++ 5.5 compiler.

## ***The Default Convention***

If there is no explicit declaration of the call type, the compiler usually uses its own conventions and chooses them at its own discretion. The `this` pointer is the most influenced – by default, some compilers transfer it via a register whereas others prefer the stack.

The Microsoft Visual C++ 7 compiler uses the `ecx` for passing the `this` pointer. Although some documents claim that Borland's C++ compiler uses the `eax` register for passing the `this` pointer, analysis have shown that it is actually pushed onto the stack. This is also the case with GCC.

Other arguments can be pushed onto the stack or can be transferred via registers if the optimizer of the compiler considers this a better way. The mechanism of transferring arguments and the logic of sampling them is different in different compilers. It is also unpredictable.

## Microsoft Visual C++ 7

Demonstration of the default calling convention using Microsoft Visual C++ 7 and IDA 5.2	
<pre>#include &lt;stdio.h&gt; #include &lt;string.h&gt;  class Demo { public:     int func(int a, int b, char* c)     {         return (a + b + strlen(c));     }; };  int main(int argc, char* argv[]) {     Demo* d = new Demo();     printf("%x\n", d-&gt;func(6, 7, "Hello w00z13"));     delete d;     return 0; }</pre>	<pre>main proc near     var_54 = dword ptr -54h     var_50 = dword ptr -50h     var_4C = dword ptr -4Ch     var_48 = dword ptr -48h     var_4 = dword ptr -4      push    ebp     mov     ebp, esp     sub     esp, 54h     push    ebx     push    esi     push    edi     mov     [ebp+var_50], 1     mov     eax, [ebp+var_50]     push    eax     call   j_operator_new     add     esp, 4     mov     [ebp+var_4C], eax     cmp     [ebp+var_4C], 0     jz      short loc_412F93     mov     ecx, [ebp+var_50]     xor     eax, eax     mov     edi, [ebp+var_4C]     mov     edx, ecx     shr     ecx, 2     rep    stosd     mov     ecx, edx     and     ecx, 3     rep    stosb     mov     eax, [ebp+var_4C]     mov     [ebp+var_54], eax     jmp     short loc_412F9A loc_412F93:     mov     [ebp+var_54], 0 loc_412F9A:     mov     ecx, [ebp+var_54]     mov     [ebp+var_4], ecx     push    offset aHello00z13     push    7     push    6     mov     ecx, [ebp+var_4]     call   j_Demo__func     push    eax     push    offset asc_42301C ; "%x\n"     call   j_printf     add     esp, 8     mov     eax, [ebp+var_4]     mov     [ebp+var_48], eax     mov     ecx, [ebp+var_48]     push    ecx     call   j_operator_delete     add     esp, 4     xor     eax, eax     pop     edi     pop     esi     pop     ebx     mov     esp, ebp     pop     ebp     retn</pre>

	<pre> main endp  Demo__func proc near      var_4= dword ptr -4     arg_0= dword ptr 8     arg_4= dword ptr 0Ch     arg_8= dword ptr 10h      push    ebp     mov     ebp, esp     sub     esp, 44h     push    ebx     push    esi     push    edi     mov     [ebp+var_4], ecx     mov     esi, [ebp+arg_0]     add     esi, [ebp+arg_4]     mov     eax, [ebp+arg_8]     push    eax     call   j_strlen     add     esp, 4     add     eax, esi     pop     edi     pop     esi     pop     ebx     mov     esp, ebp     pop     ebp     retn   0Ch Demo__func endp         </pre>
--	---

The Microsoft Visual C++ 7 compiler uses a mixture of the `__stdcall` and `__fastcall`. Arguments are passed to the called function using the stack from right to left but the `this` pointer for the class instance is passed through the `ecx` register. Stack clearance is done by the callee.

## Borland C++ 5.5

Unfortunately I was not able to use the same sample source code with the Borland C++ Compiler. The sample code was simply too short and the Borland C++ 5.5 compiler inlined the call to the member function of the `Demo` class instance. Thus the provided sample source code given in this example is rather complex but still shows how parameters are passed.

I have also omitted the disassembled listing of the function itself, as it is not necessary for understanding the default behavior of the Borland C++ compiler 5.5.

Demonstration of the default calling convention using Borland C++ 5.5 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt; #include &lt;time.h&gt;  class Demo { public:     int func(int a, int b, char* c)     {         srand(time(NULL));          int r = rand();         int t = 0;          if (a &lt;= b)         {             printf("func: a &lt;= 0\n");             t = (b - a) + strlen(c);             for (int i = 0; i &lt; t; i++)             {                 r += t * i + i;             }         }         else         {             printf("func: a &gt; b\n");         }     } };         </pre>	<pre> _main proc near      argc= dword ptr 8     argv= dword ptr 0Ch     envp= dword ptr 10h      push    ebp     mov     ebp, esp     push    ebx     push    8 ; size     call   @\$bnew\$qui ; new(uint)     pop     ecx     mov     ebx, eax     push    offset aHello00z131     push    15h     push    0Bh     push    ebx ; this     call   sub_40118C     add     esp, 10h     push    eax     push    offset format ; "%x\n"     call   _printf     add     esp, 8     push    ebx ; handle     call   __rtl__close     pop     ecx     xor     eax, eax         </pre>

<pre> t = (a - b) + strlen(c); for (int i = 0; i &lt; t; i++) {     r += t * i + i; } return r; };  int main() {     Demo* d1 = new Demo();     printf("%x\n", d1-&gt;func(11, 21, "Hello w00z13 1"));     delete d1;     return 0; } </pre>	<pre> pop     ebx pop     ebp retn _main endp  sub_40118C proc near      ....     ....  sub_40118C endp </pre>
--	--

### GCC 4.1.2

Demonstration of the default calling convention using GCC 4.1.2 and IDA 5.2	
<pre> #include &lt;stdio.h&gt; #include &lt;string.h&gt;  class Demo { public:     int func(int a, int b, char* c)     {         return (a + b + strlen(c));     }; };  int main(int argc, char* argv[]) {     Demo* d = new Demo();     printf("%x\n", d-&gt;func(6, 7, "Hello w00z13"));     delete d;     return 0; } </pre>	<pre> main proc near      var_30= dword ptr -30h     var_2C= dword ptr -2Ch     var_28= dword ptr -28h     var_24= dword ptr -24h     var_20= dword ptr -20h     arg_0= byte ptr 4      lea     ecx, [esp+arg_0]     and     esp, 0FFFFFFF0h     dword ptr [ecx-4]     push   ebp     mov     ebp, esp     push   ecx     sub     esp, 24h     mov     [esp+30h+var_30], 1     call   ___znwj     mov     [ebp-8], eax     mov     [esp+2Ch+var_20], offset aHello00z13     mov     [esp+2Ch+var_24], 7     mov     [esp+2Ch+var_28], 6     mov     eax, [ebp-8]     mov     [esp+2Ch+var_2C], eax     call   _ZN4Demo4funcEiiPc     mov     [esp+2Ch+var_28], eax     mov     [esp+2Ch+var_2C], offset asc_804865D     call   _printf     mov     eax, [ebp-8]     mov     [esp+2Ch+var_2C], eax     call   ___zd1Pv     mov     eax, 0     add     esp, 24h     pop     ecx     pop     ebp     lea     esp, [ecx-4]     retn  main endp  _ZN4Demo4funcEiiPc proc near      var_8= dword ptr -8     arg_4= dword ptr 0Ch     arg_8= dword ptr 10h     arg_C= dword ptr 14h      push   ebp     mov     ebp, esp     push   edi     sub     esp, 4     mov     eax, [ebp+arg_8]     add     eax, [ebp+arg_4]     mov     edx, eax </pre>

	<pre> mov     eax, [ebp+arg_C] mov     ecx, 0FFFFFFFh mov     [ebp+var_8], eax mov     eax, 0 cld mov     edi, [ebp+var_8] repne  scasb mov     eax, ecx not     eax sub     eax, 1 lea    eax, [edx+eax] add     esp, 4 pop     edi pop     ebp retn _ZN4Demo4funcEiiPc endp </pre>
--	--

GCC obviously uses the `_cdecl` calling convention by default. Although the `this` pointer is stored in `EAX` it is passed to the callee through the stack. The calling function is also in charge of clearing the stack.

### Conclusion

All the compilers tested share similar behavior for `_cdecl` and `_stdcall`.

However, their interpretation of the `_fastcall` calling convention varies. The Microsoft Visual C++ 7 compiler and GCC use the `ECX` and `EDX` registers for the first two integral arguments of a function and then the stack. The arguments are passed from right to left. Borland's C++ compiler uses `EAX`, `EDX` and `ECX` before pushing further arguments onto the stack. Arguments are passed from left to right.

Additionally the passing of the `this` pointer varies for the different compilers. The Microsoft Visual C++ 7 compiler is the only one, among the three tested, which uses a register by default. All other prefer to push the `this` pointer onto the stack.

It's also worth noting that the Borland C++ compiler is the only compiler among the three tested, which supports the `PASCAL` calling convention.

### References

Microsoft: <http://msdn2.microsoft.com/en-us/library/k2b2ssfy%28VS.80%29.aspx>

Borland: <http://cc.codegear.com/ProdCat.aspx?prodid=2&catid=9>

GCC: <http://gcc.gnu.org/>

Kris Kasperski – Hacker Disassembling Uncovered – A-LIST 2003 – ISBN: 1-931769-22-2

## ***GNU Free Documentation License***

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.  
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### 1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### 2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ascii without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

### 3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also

clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the

Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

## 7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have

the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

## **ADDENDUM: How to use this License for your documents**

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.