

CSC 583 Advanced Topics in Computer Security

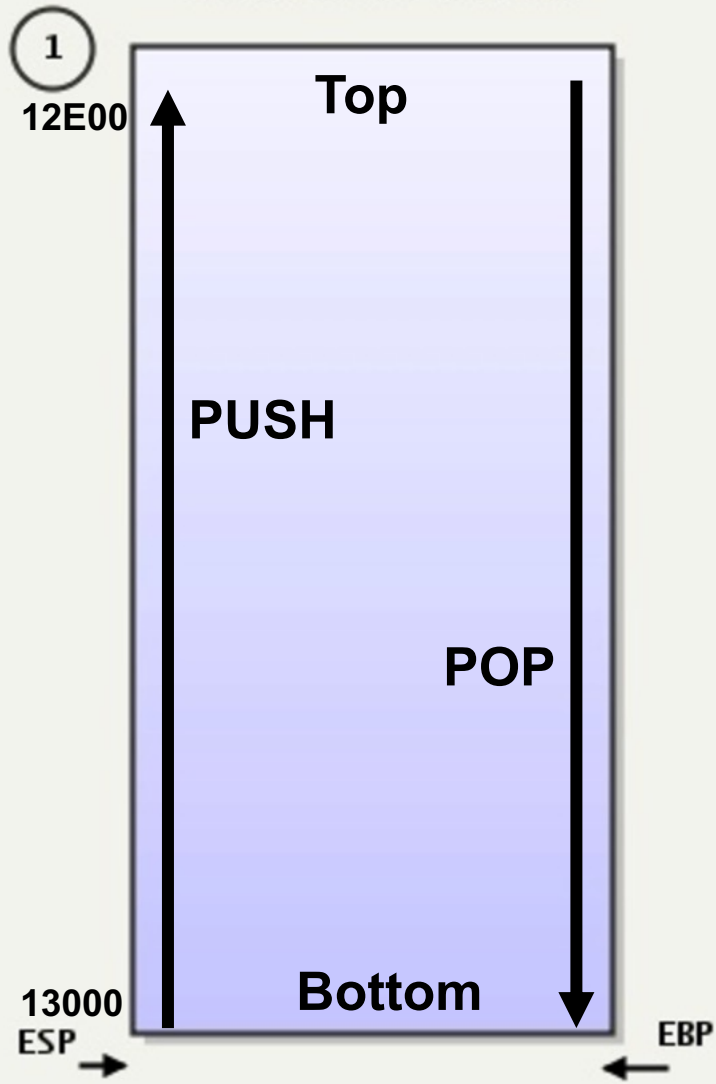
Stack and Stack Frame

Si Chen (schen@wcupa.edu)



The Stack

Stack frame details



Stack:

- A special region of your computer's memory that **stores temporary variables** created by each functions
- The stack is a "**LIFO**" (last in, first out) data structure
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

Properties:

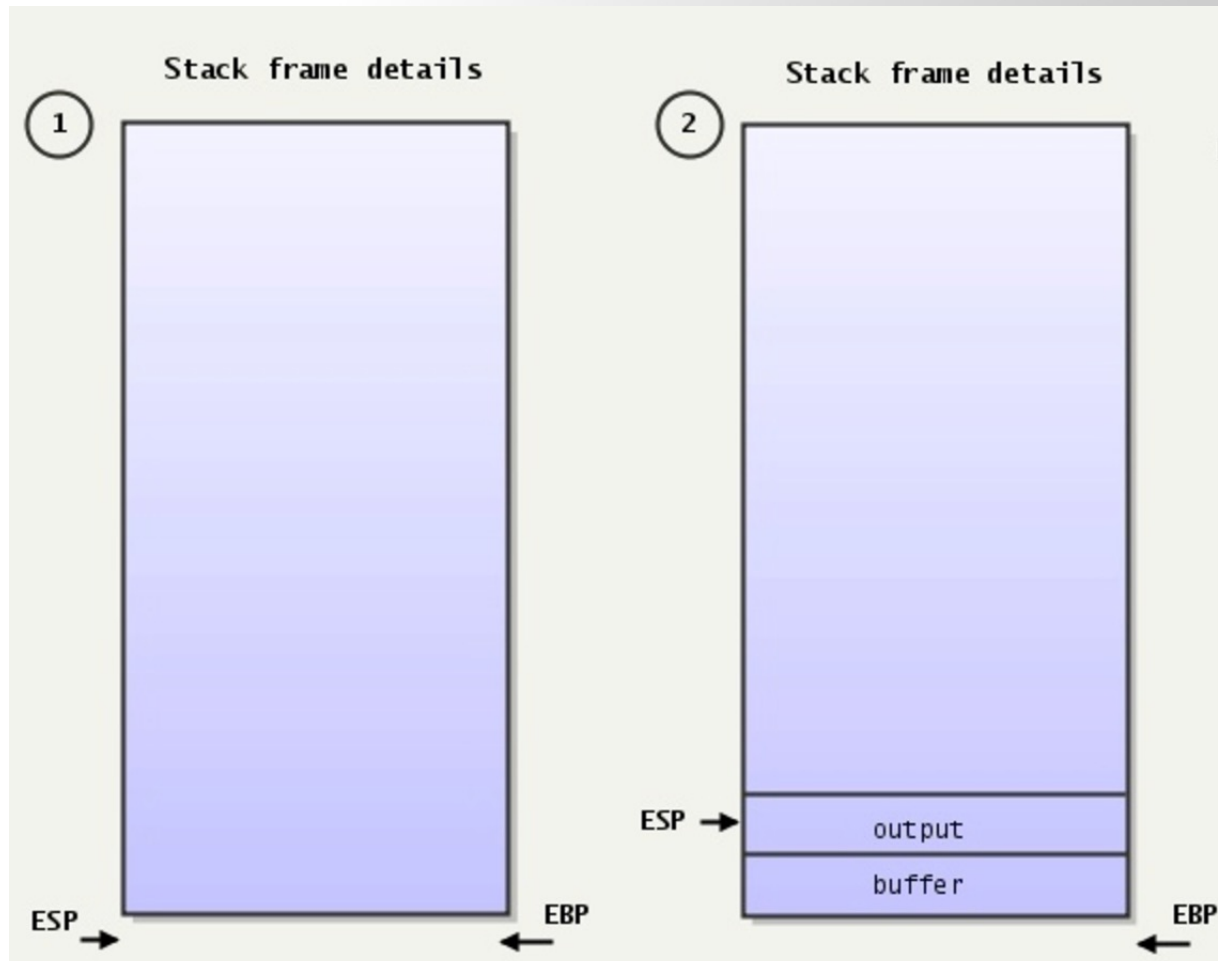
- the stack grows and shrinks as functions **push and pop** local **variables**
- there is no need to manage the memory yourself, variables are allocated and freed **automatically**
- the **stack has size limits**
- stack variables only exist while the function that created them, is running

EBP—Pointer to data on the stack
ESP—Stack pointer

The Stack

Stack:

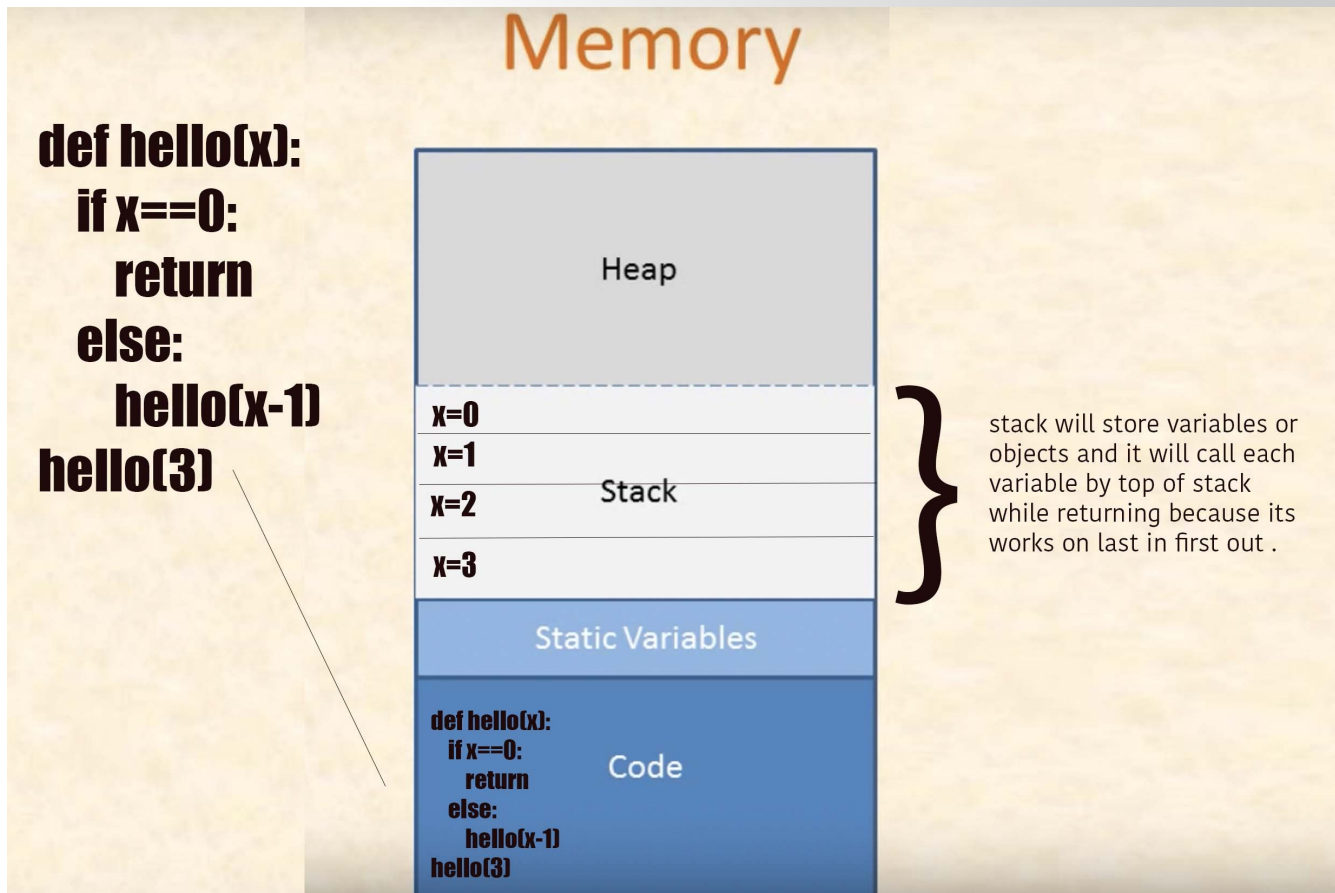
- A special region of your computer's memory that **stores temporary variables** created by each functions
- The stack is a "**LIFO**" (last in, first out) data structure
- Once a stack variable is freed, that region of memory becomes available for other stack variables.



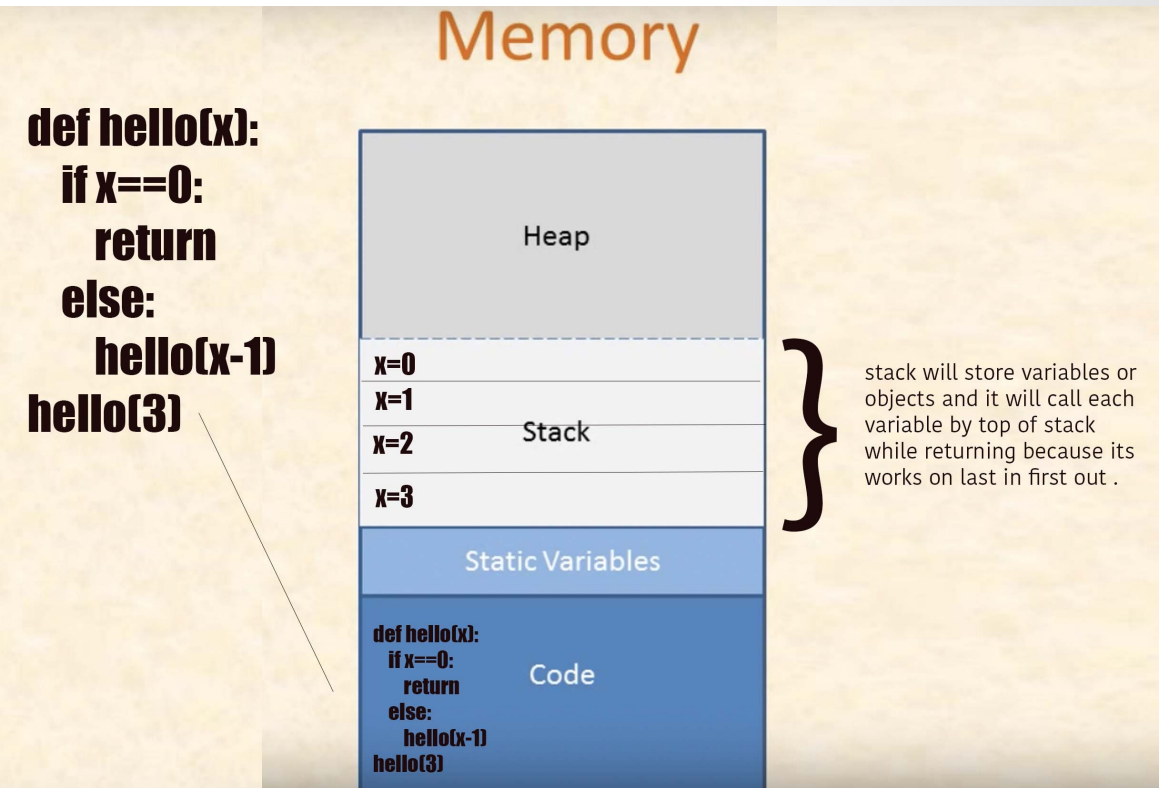
Stack Frame

Stack Frame

- A stack frame is **a frame of data that gets pushed onto the stack.**
- In the case of a **call stack**, a stack frame would represent **a function call and its argument data.**



Stack Frame



```
1 def hello(x):  
2     if x == 0:  
3         return  
4     else:  
5         hello(x-1)  
6  
7 hello(9999999)
```

```
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)  
File "stack.py", line 5, in hello  
    hello(x-1)
```

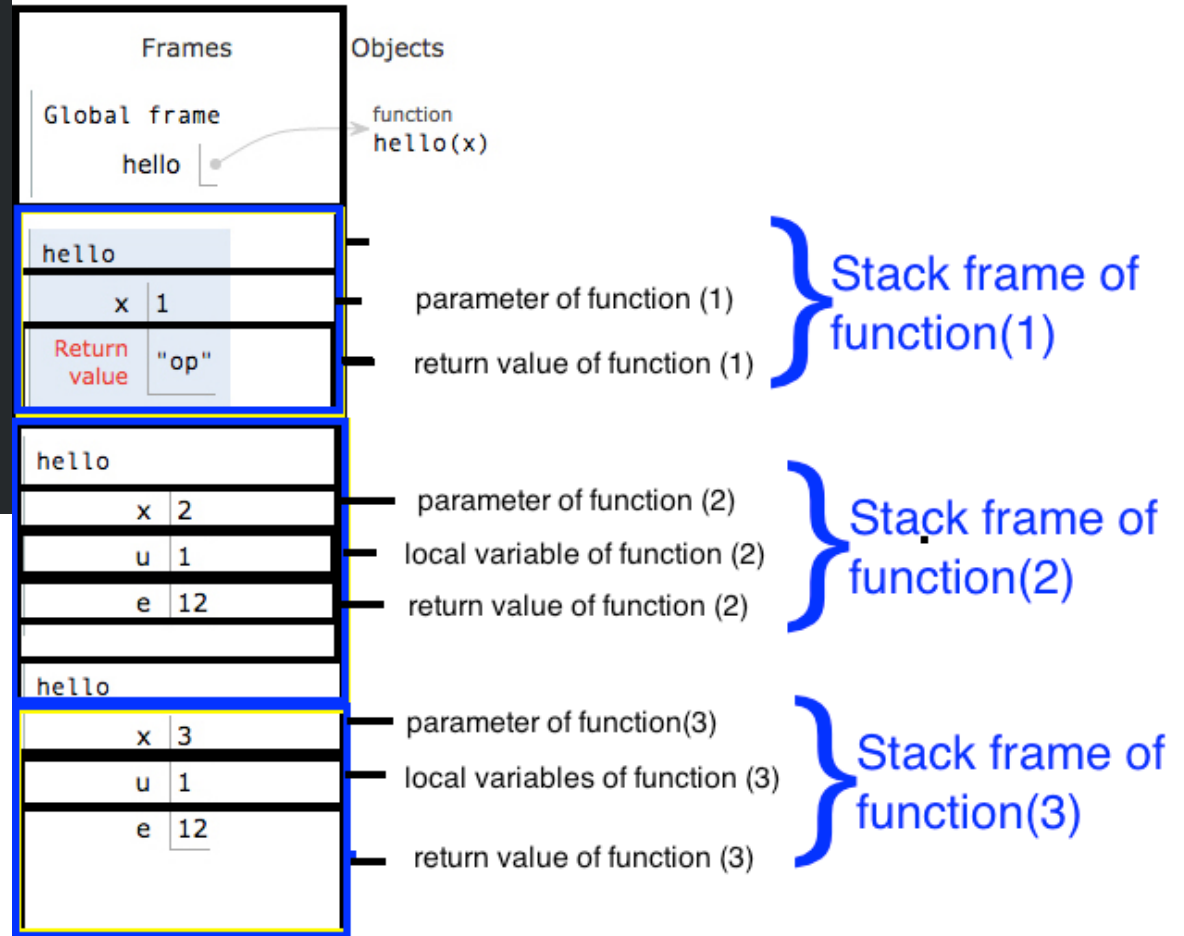
RuntimeError: maximum recursion depth exceeded

quake0day@quakes-iMac

- Pass arguments
- Save the return address
- Save **local variable**

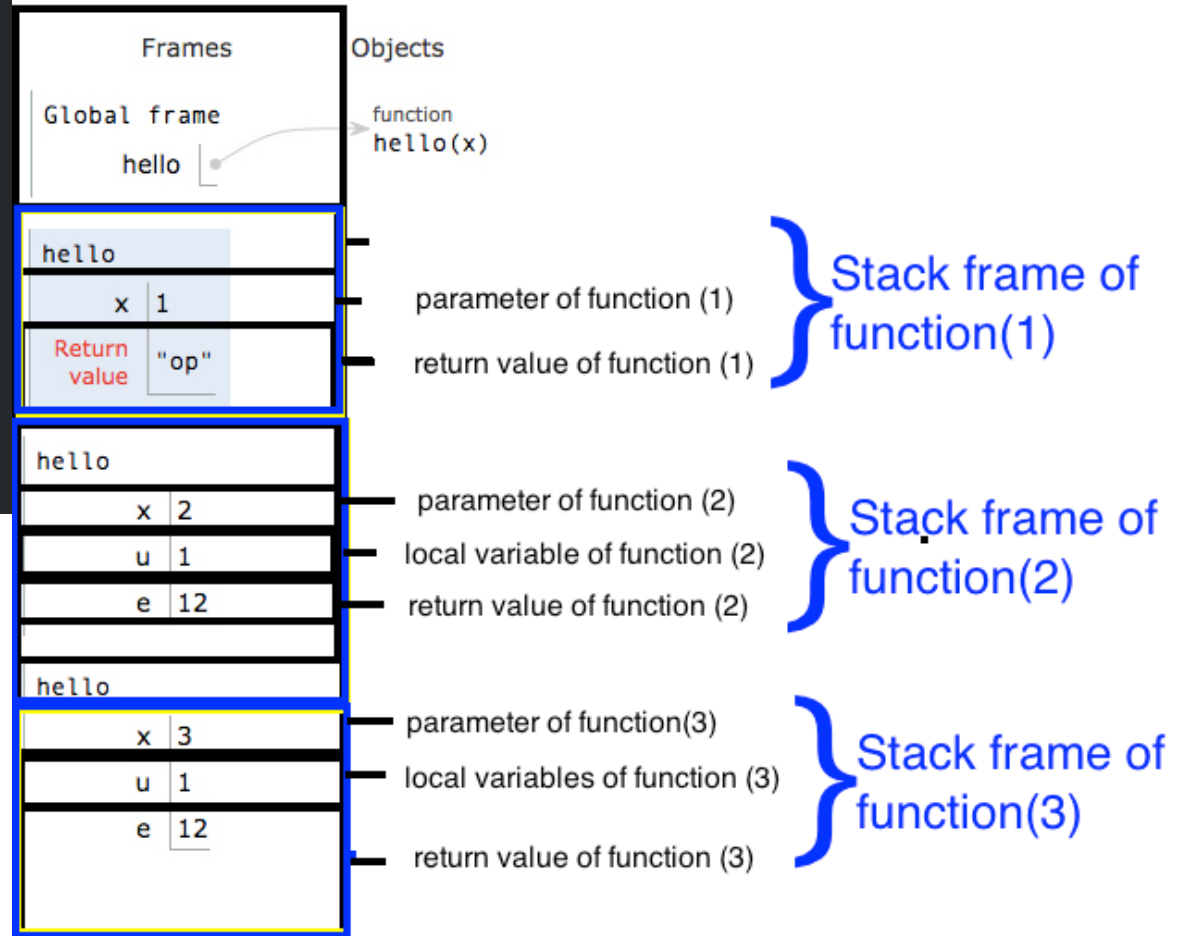
Stack Frame

```
1 def hello(x):  
2     if x == 1:  
3         return "op"  
4     else:  
5         u = 1  
6         e = 12  
7         s = hello(x - 1)  
8         e += 1  
9         print(s)  
10        print(x)  
11        u += 1  
12    return e  
13  
14  
15 hello(3)
```



Stack Frame

```
1 def hello(x):
2     if x == 1:
3         return "op"
4     else:
5         u = 1
6         e = 12
7         s = hello(x - 1)
8         e += 1
9         print(s)
10        print(x)
11        u += 1
12    return e
13
14
15 hello(3)
```



Local Variable

- Limited Register(s) → Store *Local Variable* in stack
 - Use *esp* and *ebp* to define a stack frame for current function
 - Use relative position of *esp* or *ebp* for retrieving and storing data
 - e.g. `mov eax, [esp+124]`
- Very easy to do recursive call

Functions and Frames

Each function call results in a new frame being created on the stack.

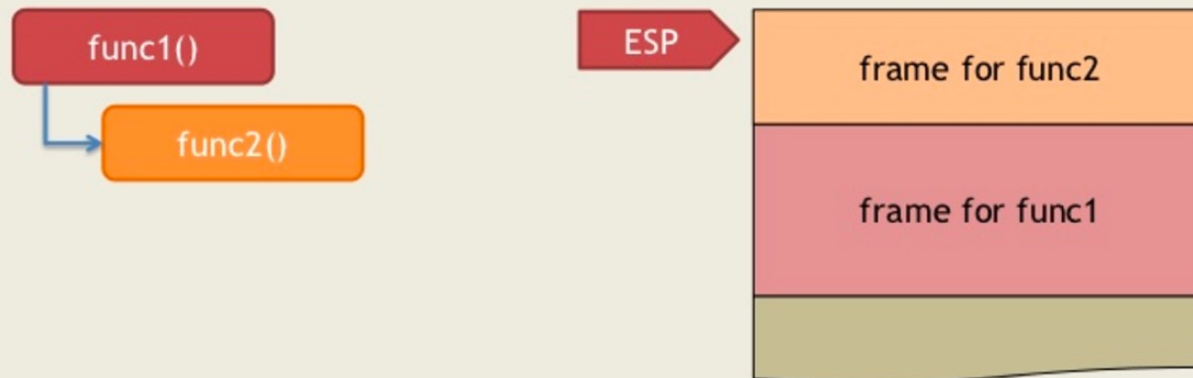
func1()

ESP

frame for func1

Functions and Frames

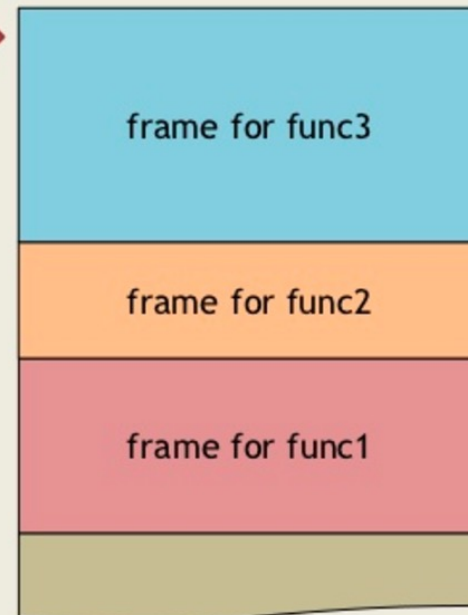
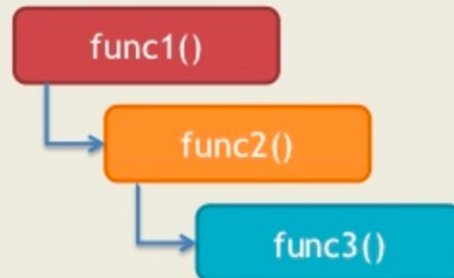
Each function call results in a new frame being created on the stack.



Functions and Frames

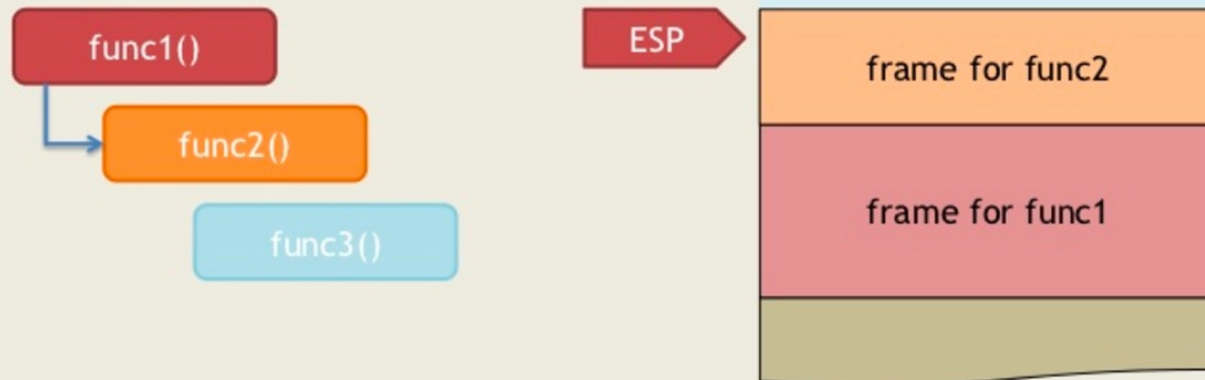
Each function call results in a new frame being created on the stack.

ESP



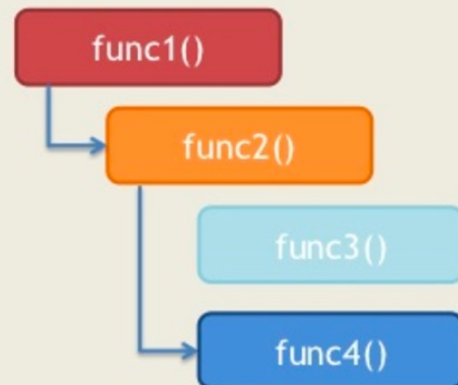
Functions and Frames

When a function returns, the frame is "unwound" or "collapsed".

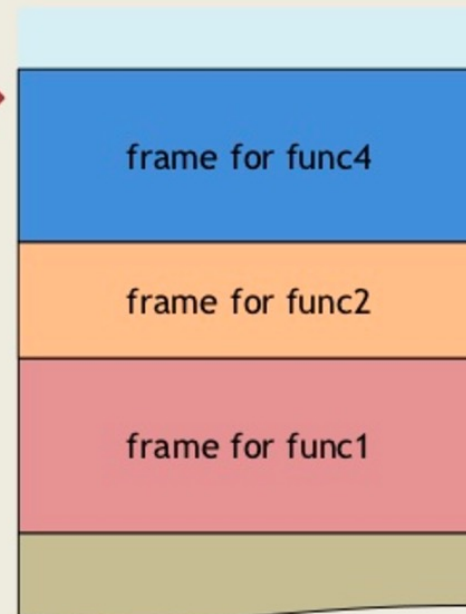


Functions and Frames

And as new functions get invoked, new frames get created.



ESP



Stack Frame

File Edit View Terminal Tabs Help

```
PUSH EBP      ; start of the func (save current EBP to stack)
MOV EBP, ESP  ; save current ESP to EBP

.....      ; function body
              ; no matter how ESP changes, the EBP remains unchanged

MOV ESP, EBP  ; move the saved function start addr back to ESP
POP EBP       ; before return the func, pop the stored EBP
RETN          ; end of the func
```

-- INSERT --

12,1

All

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

StackFrame.exe

OllyDbg - StackFrame.exe - [CPU - main thread, module StackFra]

File View Debug Plugins Options Window Help

LEMTWHC / KBR ... S

Address	Disassembly	Comment
00401000	PUSH EBP	# add()
00401001	MOV EBP,ESP	
00401003	SUB ESP,8	
00401006	MOV EAX,DWORD PTR SS:[EBP+8]	[EBP+8] => param 'a'
00401009	MOV DWORD PTR SS:[EBP-8],EAX	[EBP-8] => local 'x'
0040100C	MOV ECX,DWORD PTR SS:[EBP+C]	[EBP+C] => param 'b'
0040100F	MOV DWORD PTR SS:[EBP-4],ECX	[EBP-4] => local 'y'
00401012	MOV EAX,DWORD PTR SS:[EBP-8]	
00401015	ADD EAX,DWORD PTR SS:[EBP-4]	
00401018	MOV ESP,EBP	
0040101A	POP EBP	
0040101B	RETN	
0040101C	INT3	
0040101D	INT3	
0040101E	INT3	
0040101F	INT3	
00401020	PUSH EBP	# main()
00401021	MOV EBP,ESP	
00401023	SUB ESP,8	
00401026	MOV DWORD PTR SS:[EBP-4],1	[EBP-4] => local 'a'
0040102D	MOV DWORD PTR SS:[EBP-8],2	[EBP-8] => local 'b'
00401034	MOV EAX,DWORD PTR SS:[EBP-8]	
00401037	PUSH EAX	Arg2
00401038	MOV ECX,DWORD PTR SS:[EBP-4]	Arg1
0040103B	PUSH ECX	add()
0040103C	CALL StackFra.00401000	
00401041	ADD ESP,8	
00401044	PUSH EAX	
00401045	PUSH StackFra.0040B384	ASCII "%d"
0040104A	CALL StackFra.00401067	printf()
0040104F	ADD ESP,8	

00401067=StackFra.00401067

```

1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }

```

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0	ST1	ST2	ST3	ST4	ST5	ST6	ST7
empty	empty	empty	empty	empty	empty	empty	empty
-UNORM	+UNORM	0.0	0.0	0.0	0.0	0.0	0.0
BCBC	006E						
01050104	0069002E						
00300003	00670006						

3 2 1 0 E S P U 0

FST 0000 Cond 0 0 0 0 Err 0 0 0 0

FCW 027F Prec NEAR,53 Mask 1 1 1

EBP - n: Local vars
EBP + n: Parameters

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

StackFrame.exe

OllyDbg - StackFrame.exe - [CPU - main thread, module StackFra]

File View Debug Plugins Options Window Help

LEMTWHC / KBR...S

00401000 \$ 55 PUSH EBP
00401001 . 8BEC MOV EBP,ESP
00401003 . 83EC 08 SUB ESP,8
00401006 . 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
00401009 . 8945 F8 MOV DWORD PTR SS:[EBP-8],EAX
0040100C . 8B4D 0C MOV ECX,DWORD PTR SS:[EBP+C]
0040100F . 894D FC MOV DWORD PTR SS:[EBP-4],ECX
00401012 . 8B45 F8 MOV EAX,DWORD PTR SS:[EBP-8]
00401015 . 0345 FC ADD EAX,DWORD PTR SS:[EBP-4]
00401018 . 8BE5 MOV ESP,EBP
0040101A . 5D POP EBP
0040101B . C3 RETN
0040101C . CC INT3
0040101D . CC INT3
0040101E . CC INT3
0040101F . CC INT3
00401020 \$ 55 PUSH EBP
00401021 . 8BEC MOV EBP,ESP
00401023 . 83EC 08 SUB ESP,8
00401026 . C745 FC 010000 MOV DWORD PTR SS:[EBP-4],1
0040102D . C745 F8 020000 MOV DWORD PTR SS:[EBP-8],2
00401034 . 8B45 F8 MOV EAX,DWORD PTR SS:[EBP-8]

add()
[EBP+8] => param 'a'
[EBP-8] => local 'x'
[EBP+C] => param 'b'
[EBP-4] => local 'y'

main()
[EBP-4] => local 'a'
[EBP-8] => local 'b'

EBP=0012FFC0
local call from 0040124B

Registers (FPU)
EAX 00342F40
ECX 00000001
EDX 0040D748 StackFra.0040D748
EBX 7FFD7000
ESP 0012FF7C
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228
EIP 00401020 StackFra.00401020
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BCBC 01050104 00300003
ST1 empty +UNORM 006E 0069002E 00670006
ST2 empty 0.0

0012FF7C 00401250 RETURN to StackFra.00401250 from StackFra.0
0012FF80 00000001
0012FF84 00342EE0
0012FF88 00342F40
0012FF8C 6945F8F1
0012FF90 7C910228 ntdll.7C910228
0012FF94 FFFFFFFF
0012FF98 7FFD7000
0012FF9C 0012FFAC

Address	Hex dump	ASCII
0040C000	01 00 00 00 31 07 57 69	0...1.Wi
0040C008	CE F8 A8 96 00 00 00 00	ÿzu...
0040C010	A0 DA 40 00 00 00 00 00	áre....
0040C018	A0 DA 40 00 01 01 00 00	áre.00..
0040C020	00 00 00 00 00 00 00 00	
0040C028	00 10 00 00 00 00 00 00	.>.....
0040C030	00 00 00 00 00 00 00 00	
0040C038	00 00 00 00 02 00 00 00	0...

Breakpoint at StackFra.00401020

Paused

StackFrame.exe

OllyDbg - StackFrame.exe - [CPU - main thread, module StackFra]

File View Debug Plugins Options Window Help

LEMTWHC / KBR...S

Address	Hex	Assembly	Comment
00401000	55	PUSH EBP	# add()
00401001	8BEC	MOV EBP,ESP	
00401003	83EC 08	SUB ESP,8	
00401006	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	[EBP+8] => param 'a'
00401009	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	[EBP-8] => local 'x'
0040100C	8B4D 0C	MOV ECX,DWORD PTR SS:[EBP+C]	[EBP+C] => param 'b'
0040100F	894D FC	MOV DWORD PTR SS:[EBP-4],ECX	[EBP-4] => local 'y'
00401012	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
00401015	0345 FC	ADD EAX,DWORD PTR SS:[EBP-4]	
00401018	8BE5	MOV ESP,EBP	
0040101A	5D	POP EBP	
0040101B	C3	RETN	
0040101C	CC	INT3	
0040101D	CC	INT3	
0040101E	CC	INT3	
0040101F	CC	INT3	
00401020	55	PUSH EBP	# main()
00401021	8BEC	MOV EBP,ESP	
00401023	83EC 08	SUB ESP,8	
00401026	C745 FC 010000	MOV DWORD PTR SS:[EBP-4],1	[EBP-4] => local 'a'
0040102D	C745 F8 020000	MOV DWORD PTR SS:[EBP-8],2	[EBP-8] => local 'b'
00401034	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	

ESP=0012FF78

Address	Hex	dump	ASCII
0040C000	01 00 00 00	31 07 57 69	0...1.Wi
0040C008	CE F8 A8 96	00 00 00 00	†°žû....
0040C010	A0 DA 40 00	00 00 00 00	áŕ@.....
0040C018	A0 DA 40 00	01 01 00 00	áŕ@.00..
0040C020	00 00 00 00	00 00 00 00	
0040C028	00 10 00 00	00 00 00 00	▶.....
0040C030	00 00 00 00	00 00 00 00	
0040C038	00 00 00 00	02 00 00 00	0...

Address	Hex	dump	Comment
0012FF78	0012FFC0	00401250	RETURN to StackFra.00401250 from StackFra.0
0012FF7C	00000001	00342EE0	
0012FF80	00342F40	6945F8F1	
0012FF84	7C910228	ntdll.7C910228	
0012FF88	FFFFFFF		
0012FF8C	7FFD7000		
0012FF90			
0012FF94			
0012FF98			

Registers (FPU)	
EAX	00342F40
ECX	00000001
EDX	0040D748 StackFra.0040D748
EBX	7FFD7000
ESP	0012FF78
EBP	0012FF78
ESI	FFFFFFFF
EDI	7C910228 ntdll.7C910228
EIP	00401023 StackFra.00401023
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDF000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_SUCCESS (00000000)
EFL	00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0	empty -UNORM BCBC 01050104 00300003
ST1	empty +UNORM 006E 0069002E 00670000
ST2	empty 0.0

Paused

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 long add(long a, long b)
6 {
7     long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

StackFrame.exe

OllyDbg - StackFrame.exe - [CPU - main thread, module StackFra]

File View Debug Plugins Options Window Help

LEMTWHC / KBR...S

Address	Hex	Assembly	Comment
00401000	55	PUSH EBP	# add()
00401001	8BEC	MOV EBP,ESP	
00401003	83EC 08	SUB ESP,8	
00401006	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	[EBP+8] => param 'a'
00401009	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	[EBP-8] => local 'x'
0040100C	8B4D 0C	MOV ECX,DWORD PTR SS:[EBP+C]	[EBP+C] => param 'b'
0040100F	894D FC	MOV DWORD PTR SS:[EBP-4],ECX	[EBP-4] => local 'y'
00401012	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
00401015	0345 FC	ADD EAX,DWORD PTR SS:[EBP-4]	
00401018	8BE5	MOV ESP,EBP	
0040101A	5D	POP EBP	
0040101B	C3	RETN	
0040101C	CC	INT3	
0040101D	CC	INT3	
0040101E	CC	INT3	
0040101F	CC	INT3	
00401020	55	PUSH EBP	# main()
00401021	8BEC	MOV EBP,ESP	
00401023	83EC 08	SUB ESP,8	
00401026	C745 FC 010000	MOV DWORD PTR SS:[EBP-4],1	[EBP-4] => local 'a'
0040102D	C745 F8 020000	MOV DWORD PTR SS:[EBP-8],2	[EBP-8] => local 'b'
00401034	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	

ESP=0012FF78

Address	Hex dump	ASCII
0040C000	01 00 00 00 31 07 57 69	0...1.Wi
0040C008	CE F8 A8 96 00 00 00 00	°zu...
0040C010	A0 DA 40 00 00 00 00 00	ar....
0040C018	A0 DA 40 00 01 01 00 00	ar.00..
0040C020	00 00 00 00 00 00 00 00	
0040C028	00 10 00 00 00 00 00 00	
0040C030	00 00 00 00 00 00 00 00	
0040C038	00 00 00 00 02 00 00 00	0...

Address	Hex	Comment
0012FF78	0012FFC0	RETURN to StackFra.00401250 from StackFra.00401250
0012FF7C	00401250	
0012FF80	00000001	
0012FF84	00342EE0	
0012FF88	00342F40	
0012FF8C	6945F8F1	
0012FF90	7C910228	ntdll.7C910228
0012FF94	FFFFFFFF	
0012FF98	7FFD7000	

Registers (FPU)

EAX 00342F40
ECX 00000001
EDX 0040D748 StackFra.0040D748
EBX 7FFD7000
ESP 0012FF78
EBP 0012FF78
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228
EIP 00401023 StackFra.00401023

C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BCBC 01050104 00300003
ST1 empty +UNORM 006E 0069002E 00670000
ST2 empty 0.0

Paused

Create space for 'a' and 'b' → long → 4 byte

StackFrame.exe

OllyDbg - StackFrame.exe - [CPU - main thread, module StackFra]

File View Debug Plugins Options Window Help

LEMTWHC / KBR...S

Address	Hex	Assembly	Comment
00401000	55	PUSH EBP	# add()
00401001	8BEC	MOV EBP,ESP	
00401003	83EC 08	SUB ESP,8	
00401006	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	[EBP+8] => param 'a'
00401009	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	[EBP-8] => local 'x'
0040100C	8B4D 0C	MOV ECX,DWORD PTR SS:[EBP+C]	[EBP+C] => param 'b'
0040100F	894D FC	MOV DWORD PTR SS:[EBP-4],ECX	[EBP-4] => local 'y'
00401012	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
00401015	0345 FC	ADD EAX,DWORD PTR SS:[EBP-4]	
00401018	8BE5	MOV ESP,EBP	
0040101A	5D	POP EBP	
0040101B	C3	RETN	
0040101C	CC	INT3	
0040101D	CC	INT3	
0040101E	CC	INT3	
0040101F	CC	INT3	
00401020	55	PUSH EBP	# main()
00401021	8BEC	MOV EBP,ESP	
00401023	83EC 08	SUB ESP,8	
00401026	C745 FC 010000	MOV DWORD PTR SS:[EBP-4],1	[EBP-4] => local 'a'
0040102D	C745 F8 020000	MOV DWORD PTR SS:[EBP-8],2	[EBP-8] => local 'b'
00401034	8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	

ESP=0012FF78

Address	Hex	dump	ASCII
0040C000	01 00 00 00	31 07 57 69	0...1.Wi
0040C008	CE F8 A8 96	00 00 00 00	°zu...
0040C010	A0 DA 40 00	00 00 00 00	ar....
0040C018	A0 DA 40 00	01 01 00 00	ar.00..
0040C020	00 00 00 00	00 00 00 00	
0040C028	00 10 00 00	00 00 00 00	
0040C030	00 00 00 00	00 00 00 00	
0040C038	00 00 00 00	02 00 00 00	0...

Registers (FPU)
EAX 00342F40
ECX 00000001
EDX 0040D748 StackFra.0040D748
EBX 7FFD7000
ESP 0012FF78
EBP 0012FF78
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228
EIP 00401023 StackFra.00401023
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BCBC 01050104 00300003
ST1 empty +UNORM 006E 0069002E 00670006
ST2 empty 0.0

Address	Hex	dump	Comment
0012FF78	0012FFC0	00401250	RETURN to StackFra.00401250 from StackFra.00401023
0012FF7C	00000001		
0012FF80	00342EE0		
0012FF84	00342F40		
0012FF88	6945F8F1		
0012FF8C	7C910228	ntdll.7C910228	
0012FF90	FFFFFFFF		
0012FF94	7FFD7000		
0012FF98			

Paused

Create space for 'a' and 'b' → long → 4 byte

StackFrame.exe

```
00401026 | . C745 FC 010000 MOV DWORD PTR SS:[EBP-4],1      [EBP-4] => local 'a'
0040102D | . C745 F8 020000 MOV DWORD PTR SS:[EBP-8],2      [EBP-8] => local 'b'
```

Assembly	C	Type Conversion
DWORD PTR SS:[EBP-4]	*(DWORD*)(EBP-4)	DWORD (4 byte)
WORD PTR SS:[EBP-4]	*(WORD*)(EBP-4)	WORD (2 byte)
BYTE PTR SS:[EBP-4]	*(BYTE*)(EBP-4)	1 byte

4 Byte memory space at address [EBP-4]

0012FF70	00000002	
0012FF74	00000001	
0012FF78	0012FFC0	
0012FF7C	00401250	RETURN to StackFra.00401250 from StackFra.00
0012FF80	00000001	
0012FF84	00342EE0	
0012FF88	00342F40	
0012FF8C	6945F8F1	
0012FF90	7C910228	ntdll.7C910228

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

00401034	. 8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
00401037	. 50	PUSH EAX	Arg2
00401038	. 8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]	
0040103B	. 51	PUSH ECX	Arg1
0040103C	. E8 BFFFFFFF	CALL StackFra.00401000	add()

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

00401000	\$ 55	PUSH EBP	# add()
00401001	. 8BEC	MOV EBP,ESP	

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

00401000	\$ 55	PUSH EBP	# add()
00401001	. 8BEC	MOV EBP,ESP	
00401003	. 83EC 08	SUB ESP,8	
00401006	. 8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	[EBP+8] => param 'a'
00401009	. 8945 F8	MOV DWORD PTR SS:[EBP-8],EAX	[EBP-8] => local 'x'
0040100C	. 8B4D 0C	MOV ECX,DWORD PTR SS:[EBP+C]	[EBP+C] => param 'b'
0040100F	. 894D FC	MOV DWORD PTR SS:[EBP-4],ECX	[EBP-4] => local 'y'

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

00401012	. 8B45 F8	MOV EAX,DWORD PTR SS:[EBP-8]	
00401015	. 0345 FC	ADD EAX,DWORD PTR SS:[EBP-4]	

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

00401018	. 8BE5	MOV ESP,EBP
0040101A	. 5D	POP EBP
0040101B	. C3	RETN

00401041	. 83C4 08	ADD ESP,8	Clean Stack	ASCII "%d" printf()
00401044	. 50	PUSH EAX		
00401045	. 68 84B34000	PUSH StackFra.0040B384		
0040104A	. E8 18000000	CALL StackFra.00401067		

00401041	. 83C4 08	ADD ESP,8	Clean Stack	ASCII "%d" printf()
00401044	. 50	PUSH EAX		
00401045	. 68 84B34000	PUSH StackFra.0040B384		
0040104A	. E8 18000000	CALL StackFra.00401067		

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

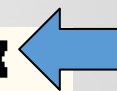
00401044	. 50	PUSH EAX	
00401045	. 68 84B34000	PUSH StackFra.0040B384	ASCII "%d\n"
0040104A	. E8 18000000	CALL StackFra.00401067	printf()
0040104F	. 83C4 08	ADD ESP,8	

StackFrame.exe

```
1 // StackFrame.cpp
2
3 #include "stdio.h"
4
5 Long add(Long a, Long b)
6 {
7     Long x = a, y = b;
8     return (x + y);
9 }
10
11 int main(int argc, char* argv[])
12 {
13     Long a = 1, b = 2;
14     printf("%d\n", add(a, b));
15     return 0;
16 }
```

00401052	. 33C0
00401054	. 8BE5
00401056	. 5D

XOR EAX,EAX
MOV ESP,EBP
POP EBP



Set EAX → 0
Faster than
MOV EAX,0

Calling Convention

Two Questions

- Q: When a function finished, how to handle the parameter left in the stack.

0012FF70	00000002	
0012FF74	00000001	
0012FF78	0012FFC0	
0012FF7C	00401250	RETURN to StackFra.00401250 from StackFra.00
0012FF80	00000001	
0012FF84	00342EE0	
0012FF88	00342F40	
0012FF8C	6945F8F1	
0012FF90	7C910228	ntdll.7C910228

A: We don't care...

- Q: When a function finished, how change the ESP value?

A: ESP should be restored to the previous value

Standard C Calling Conventions

- **Calling conventions** are a standardized method for functions to be implemented and called by the machine.
- A calling convention specifies the method that a compiler sets up to access a subroutine.
- There are three major calling conventions that are used with the C language on 32-bit x86 processors:
 - CDECL
 - STDCALL,
 - FASTCALL.

- The C language, by default, uses the CDECL calling convention
- In the CDECL calling convention the following holds:
 - Arguments are passed on the stack in Right-to-Left order, and return values are passed in eax.
 - The **calling function cleans the stack**. This allows CDECL functions to have *variable-length argument lists*.

STDCALL

```
_cdecl int MyFunction1(int a, int b)
{
    return a + b;
}
```

and the following function call:

```
x = MyFunction1(2, 3);
```

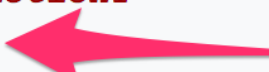
- The C language, by default
- In the CDECL calling convention
 - Arguments are passed on the stack and are passed in `eax`.
 - The **calling function cleans up** the stack after the function call.

These would produce the following assembly listings, respectively:

```
_MyFunction1:
push ebp
mov ebp, esp
mov eax, [ebp + 8]
mov edx, [ebp + 12]
add eax, edx
pop ebp
ret
```

and

```
push 3
push 2
call _MyFunction1
add esp, 8
```



- STDCALL, also known as "WINAPI" (and a few other names, depending on where you are reading it) is used almost exclusively by Microsoft as the standard calling convention for the Win32 API.
 - STDCALL passes arguments right-to-left, and returns the value in eax.
 - The **called function cleans the stack**, unlike CDECL. This means that STDCALL doesn't allow variable-length argument lists.

STDCALL

- STDCALL, also known as "WINAPI" (on where you are reading it) is used as the standard calling convention for Windows API functions.
 - STDCALL passes arguments right-to-left.
 - The **called function cleans the stack** after it returns. STDCALL doesn't allow variable-length arguments.

RET 8 → RET + POP 8 Byte

Consider the following C function:

```
_stdcall int MyFunction2(int a, int b)
{
    return a + b;
}
```

and the calling instruction:

```
x = MyFunction2(2, 3);
```

These will produce the following respective assembly code fragments:

```
:_MyFunction2@8
push ebp
mov ebp, esp
mov eax, [ebp + 8]
mov edx, [ebp + 12]
add eax, edx
pop ebp
ret 8
```



and

```
push 3
push 2
call _MyFunction2@8
```

- The FASTCALL calling convention **is not completely standard** across all compilers, so it should be used with caution.
- The calling function most frequently is responsible for cleaning the stack, if needed.

Q & A

