Class 7

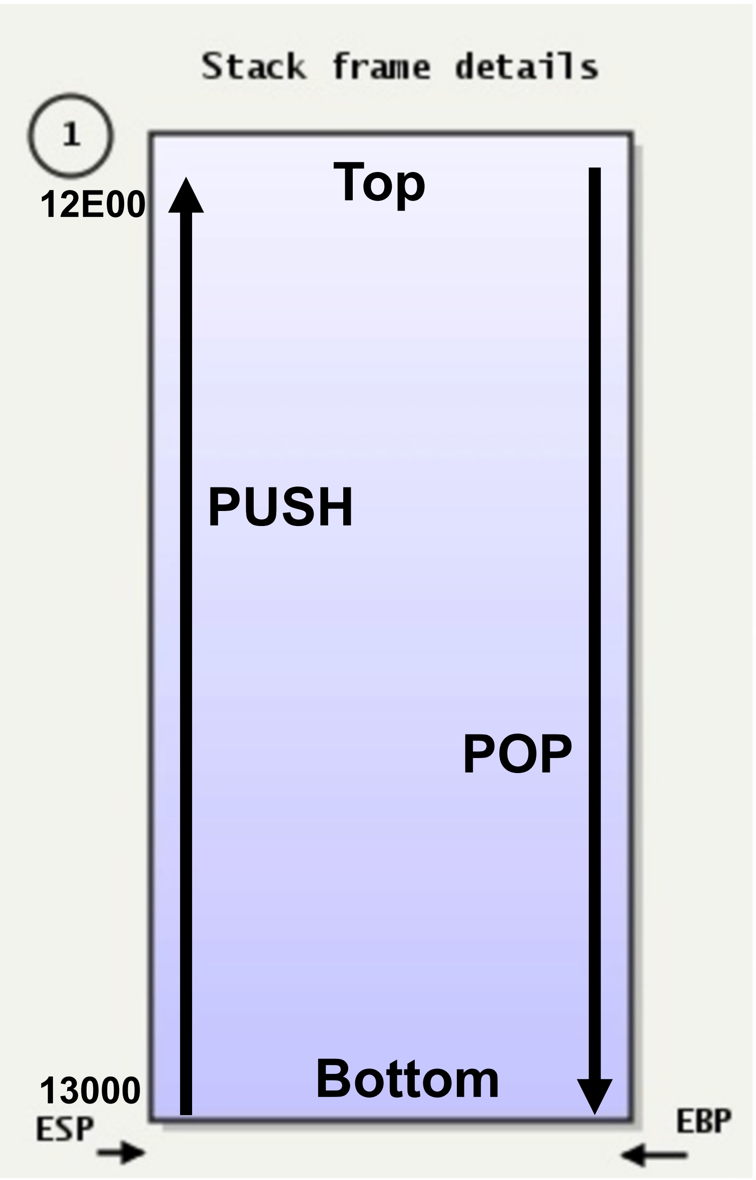
CSC 471 Modern Malware Analysis
Stack and Stack Frame

Si Chen (schen@wcupa.edu)

Assembly 101 Example: abexcm1-voiees.exe

- Assembly 101 Example: abexcm1-voiees.exe
- Open Assembly 101 Example: abexcm1-voiees.exe in your windows XP VM.
- Use Ollydbg to open the file (Desktop/toys/abexcm1-voiees.exe)
  - Dynamic Analysis

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.

Properties:

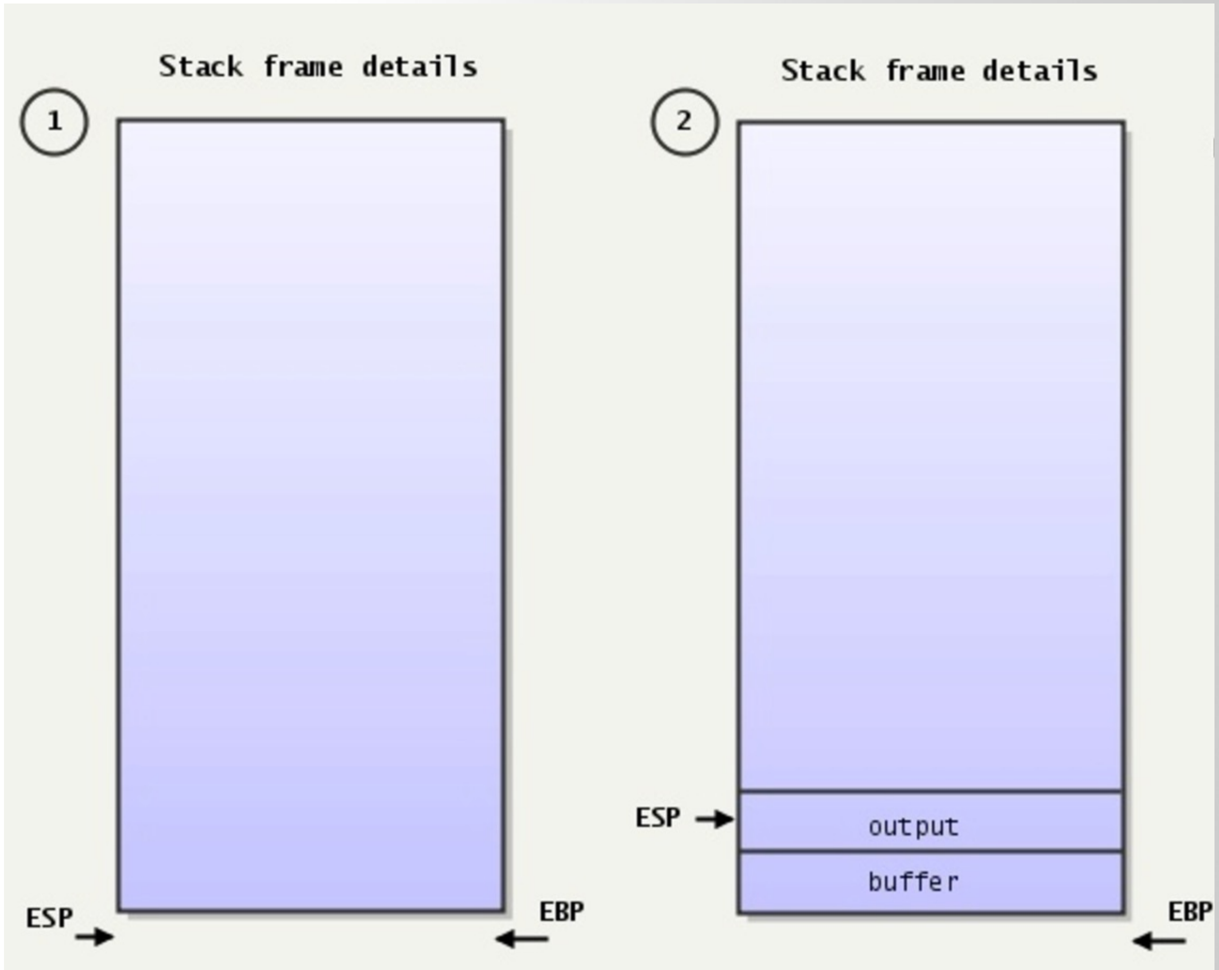
- the stack grows and shrinks as functions **push and pop** local **variables**
- there is no need to manage the memory yourself, variables are allocated and freed **automatically**
- the **stack has size limits**
- stack variables only exist while the function that created them, is running

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

The Stack

Stack:

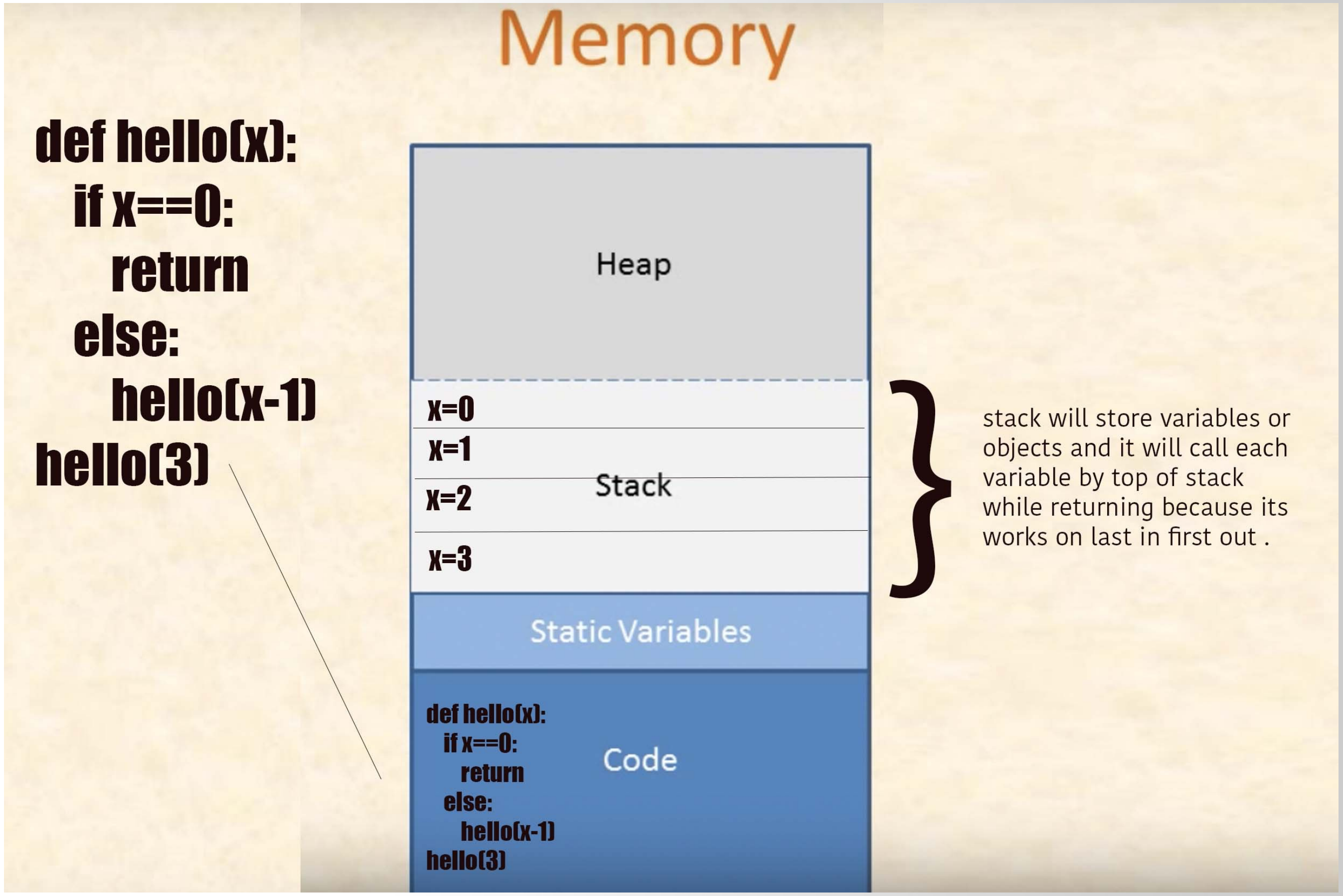
- A special region of your computer's memory that **stores temporary variables** created by each functions
- The stack is a "**LIFO**" (last in, first out) data structure
- Once a stack variable is freed, that region of memory becomes available for other stack variables.



Stack Frame

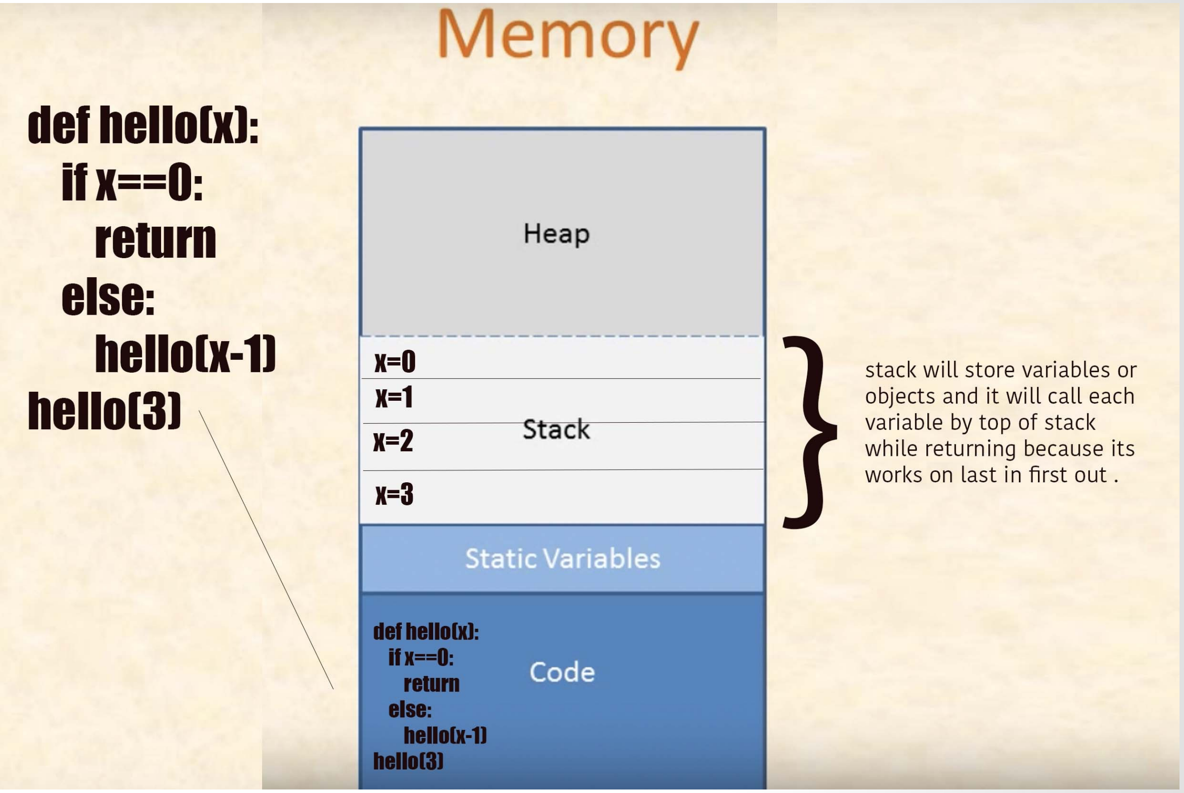
Stack Frame

- 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**.


Memory
Heap
x=0
x=1
x=2
x=3
Stack
Static Variables
Code
stack will store variables or objects and it will call each variable by top of stack while returning because its works on last in first out .

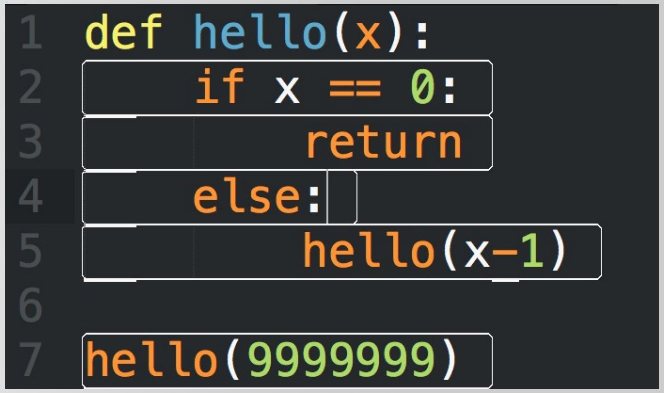
```
def hello(x):
    if x==0:
        return
    else:
        hello(x-1)
hello(3)
```

Stack Frame

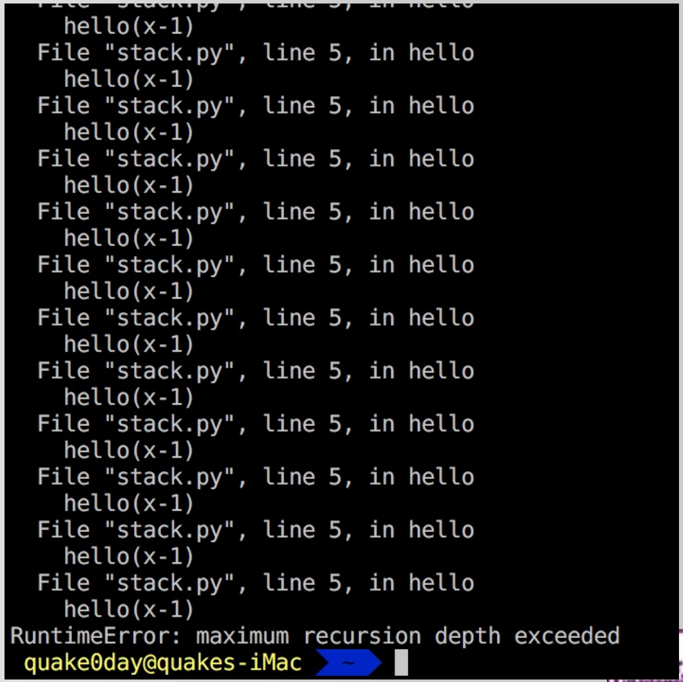


```
def hello(x):
    if x==0:
        return
    else:
        hello(x-1)
hello(3)
```

stack will store variables or objects and it will call each variable by top of stack while returning because its works on last in first out .



```
def hello(x):
    if x == 0:
        return
    else:
        hello(x-1)

hello(9999999)
```

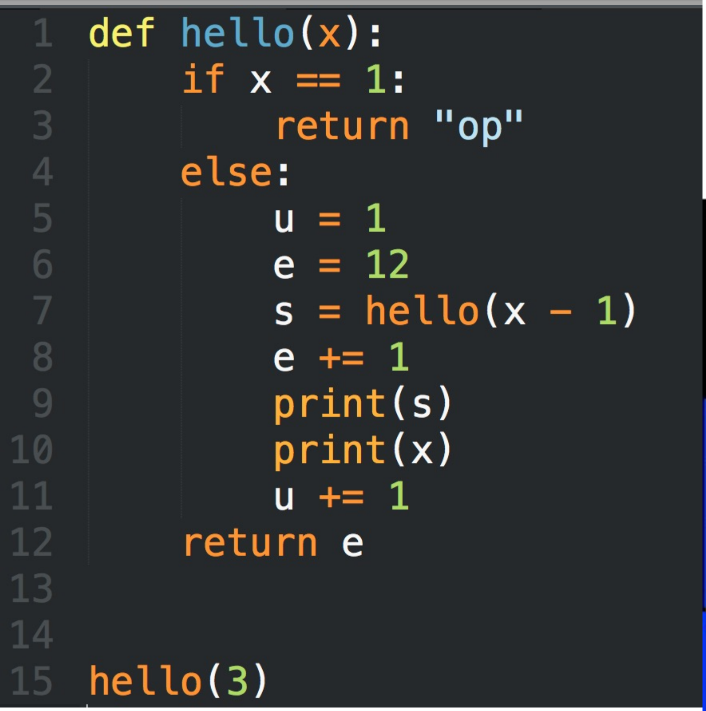


```
    hello(x-1)
  File "stack.py", line 5, in hello
    hello(x-1)
  File "stack.py", line 5, in hello
    hello(x-1)
  File "stack.py", line 5, in hello
    hello(x-1)
  File "stack.py", line 5, in hello
    hello(x-1)
  File "stack.py", line 5, in hello
    hello(x-1)
  File "stack.py", line 5, 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)
  File "stack.py", line 5, in hello
    hello(x-1)
RuntimeError: maximum recursion depth exceeded
 quake0day@quakes-iMac  ~
```

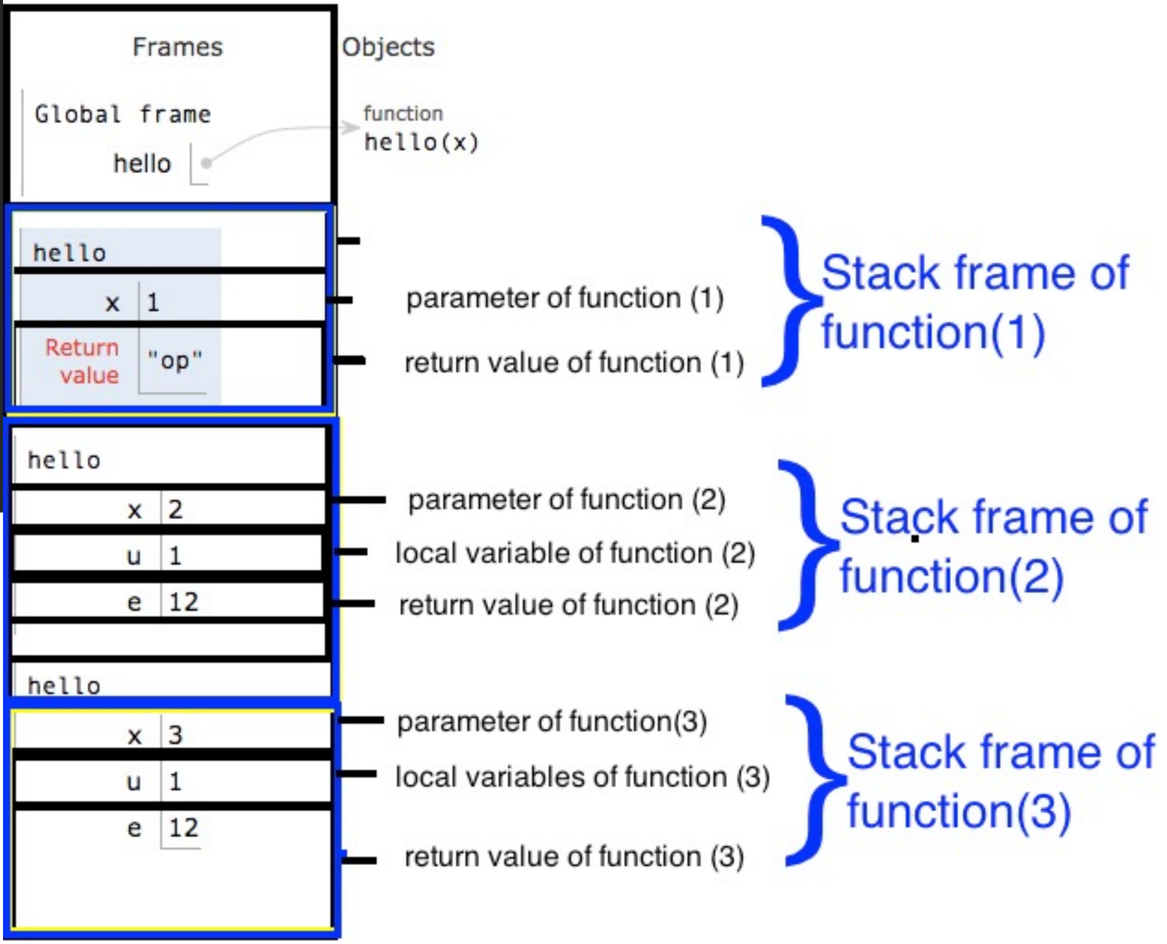
Stack

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

Stack Frame

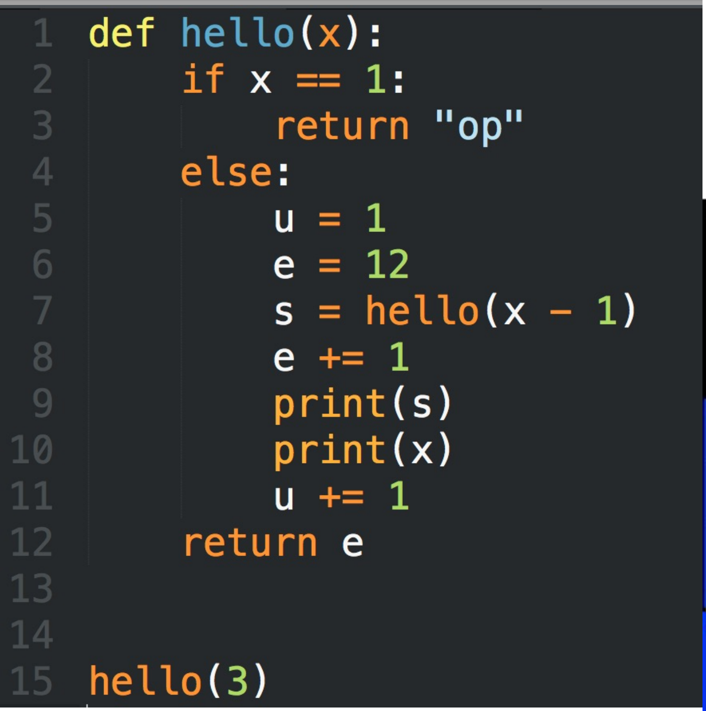


```python
def hello(x):
    if x == 1:
        return "op"
    else:
        u = 1
        e = 12
        s = hello(x - 1)
        e += 1
        print(s)
        print(x)
        u += 1
    return e


hello(3)
```

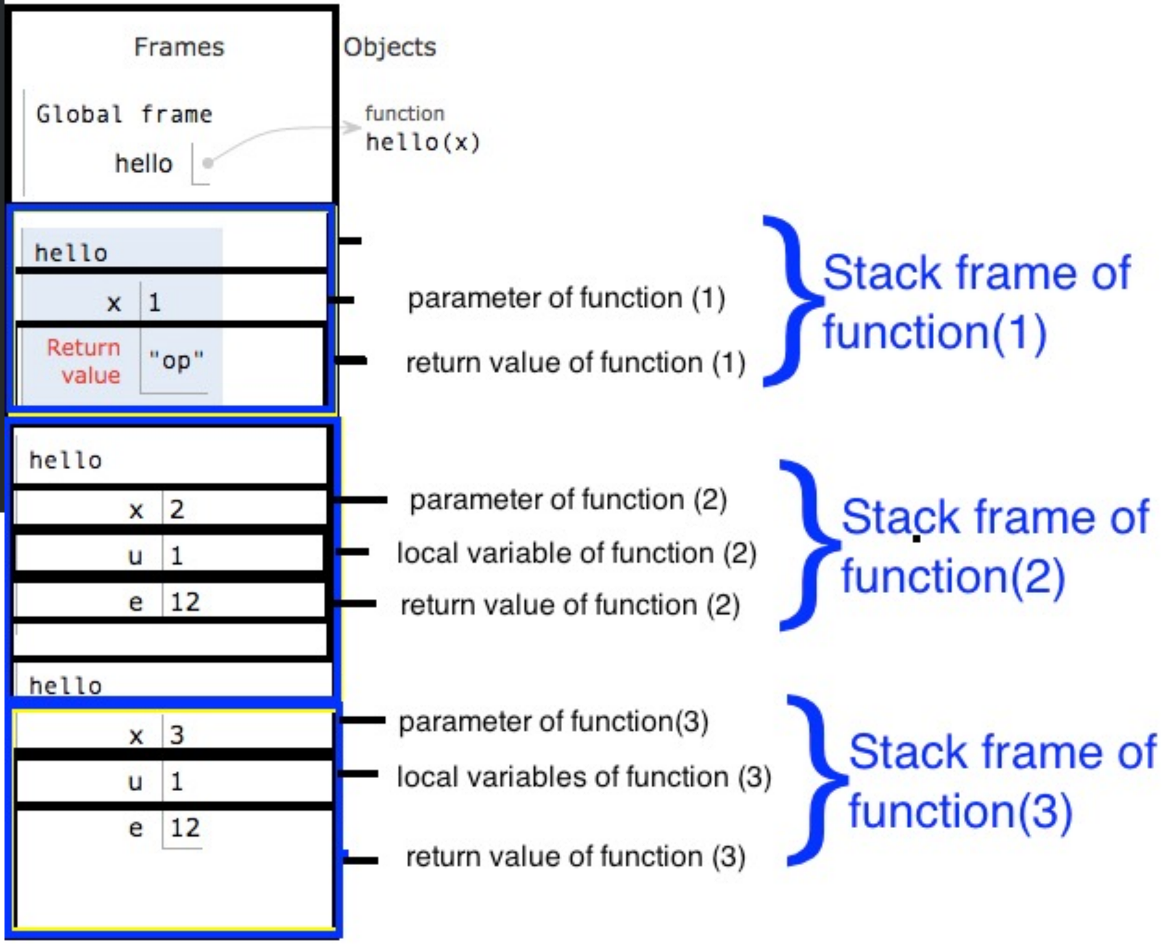


Frames
Objects
Global frame
hello
function
hello(x)
hello
x 1
Return value "op"
parameter of function (1)
return value of function (1)
Stack frame of function(1)
hello
x 2
u 1
e 12
parameter of function (2)
local variable of function (2)
return value of function (2)
Stack frame of function(2)
hello
x 3
u 1
e 12
parameter of function(3)
local variables of function (3)
return value of function (3)
Stack frame of function(3)

Stack Frame



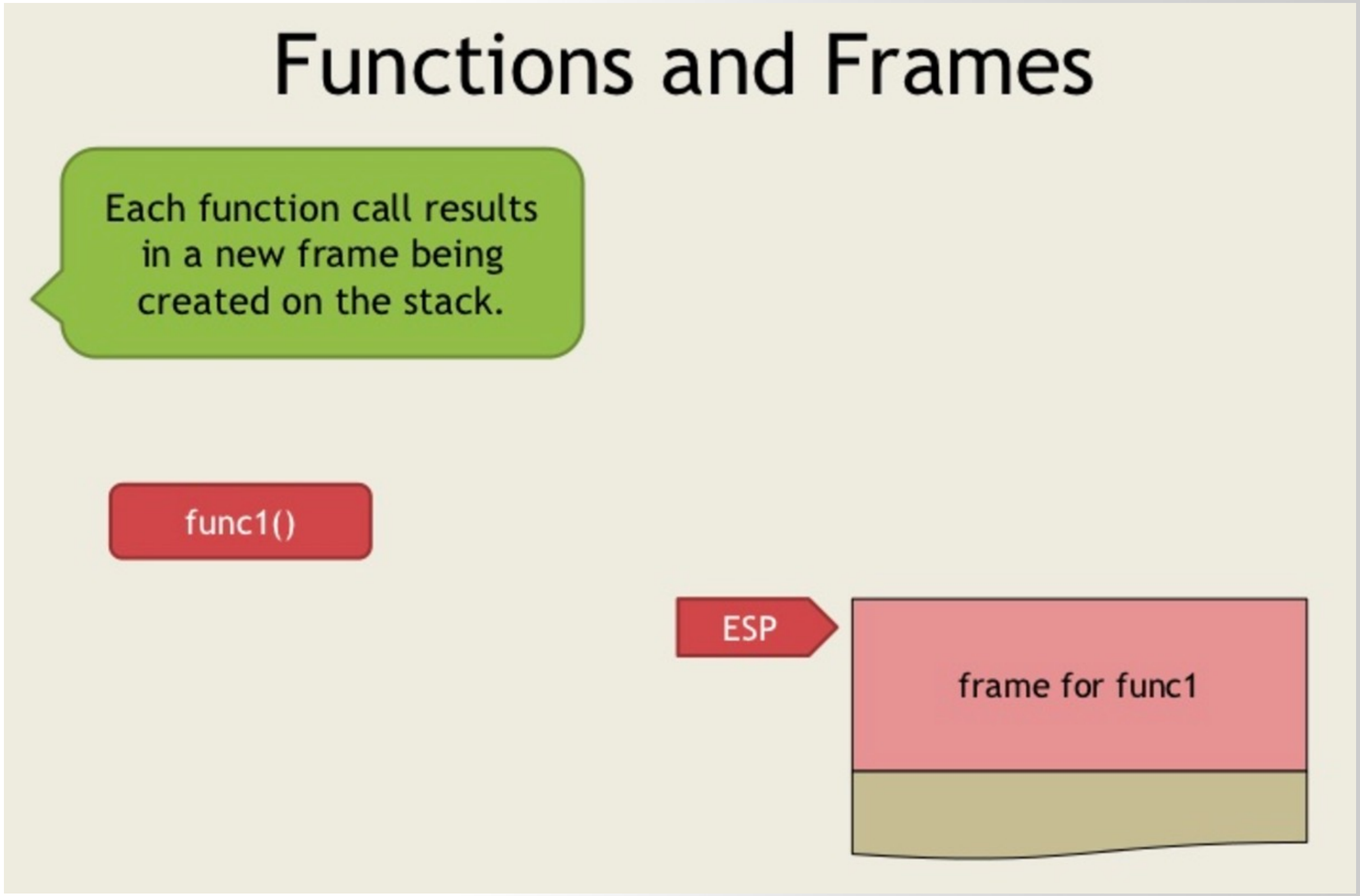
```
def hello(x):
    if x == 1:
        return "op"
    else:
        u = 1
        e = 12
        s = hello(x - 1)
        e += 1
        print(s)
        print(x)
        u += 1
    return e


hello(3)
```



Frames
Objects
Global frame
hello
function
hello(x)
hello
x 1
Return value "op"
parameter of function (1)
return value of function (1)
Stack frame of function(1)
hello
x 2
u 1
e 12
parameter of function (2)
local variable of function (2)
return value of function (2)
Stack frame of function(2)
hello
x 3
u 1
e 12
parameter of function(3)
local variables of function (3)
return value of function (3)
Stack frame of function(3)

Local Variable

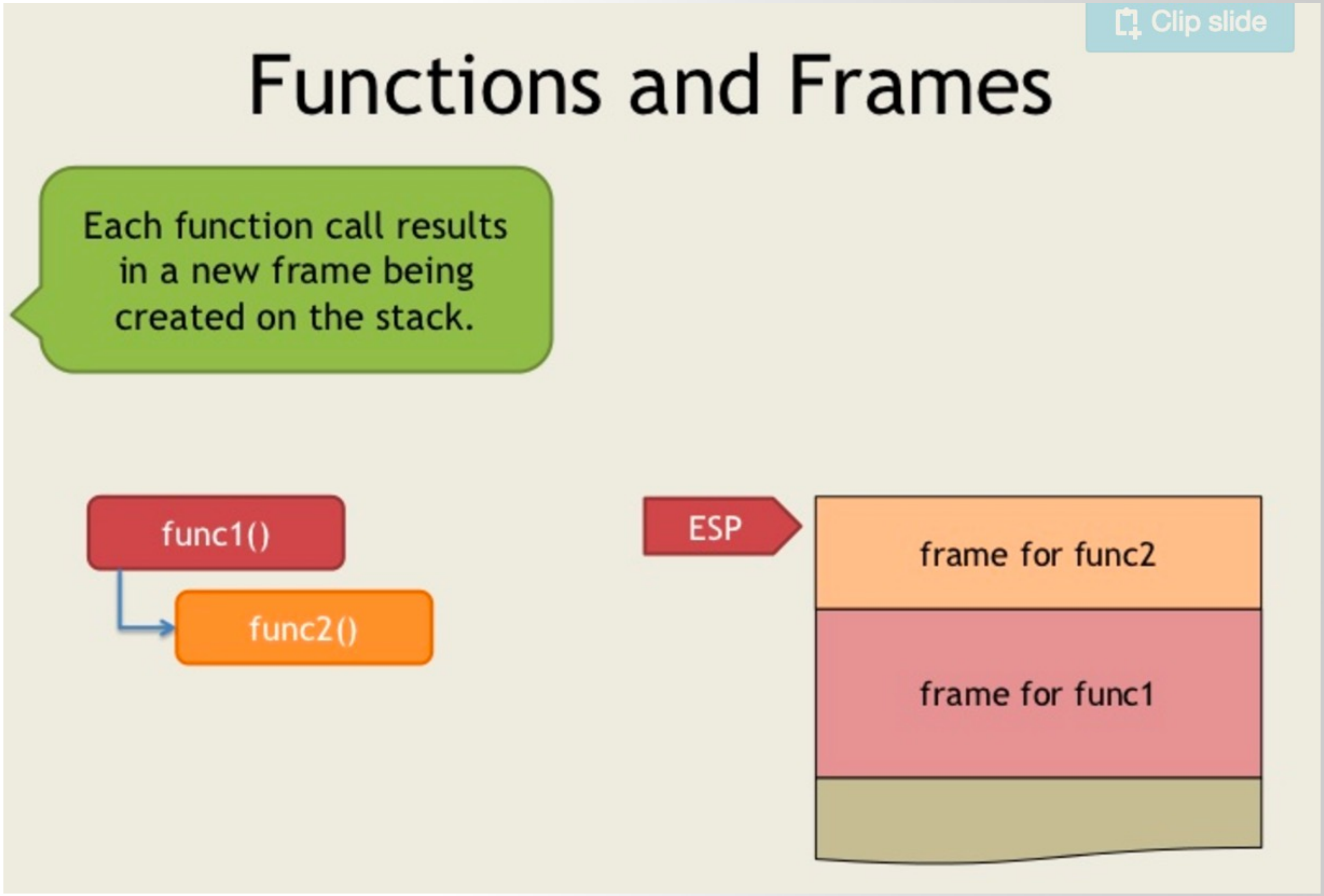
- Limited Register(s) → Store *Local Variable* in stack
  - Use *esp* and *ebp* to define a stack frame for current function
  - Use relative position of *esp* or *ebp* for retrieving and storing data
    - e.g. mov eax, [esp+124]
- Very easy to do recursive call



Functions and Frames

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

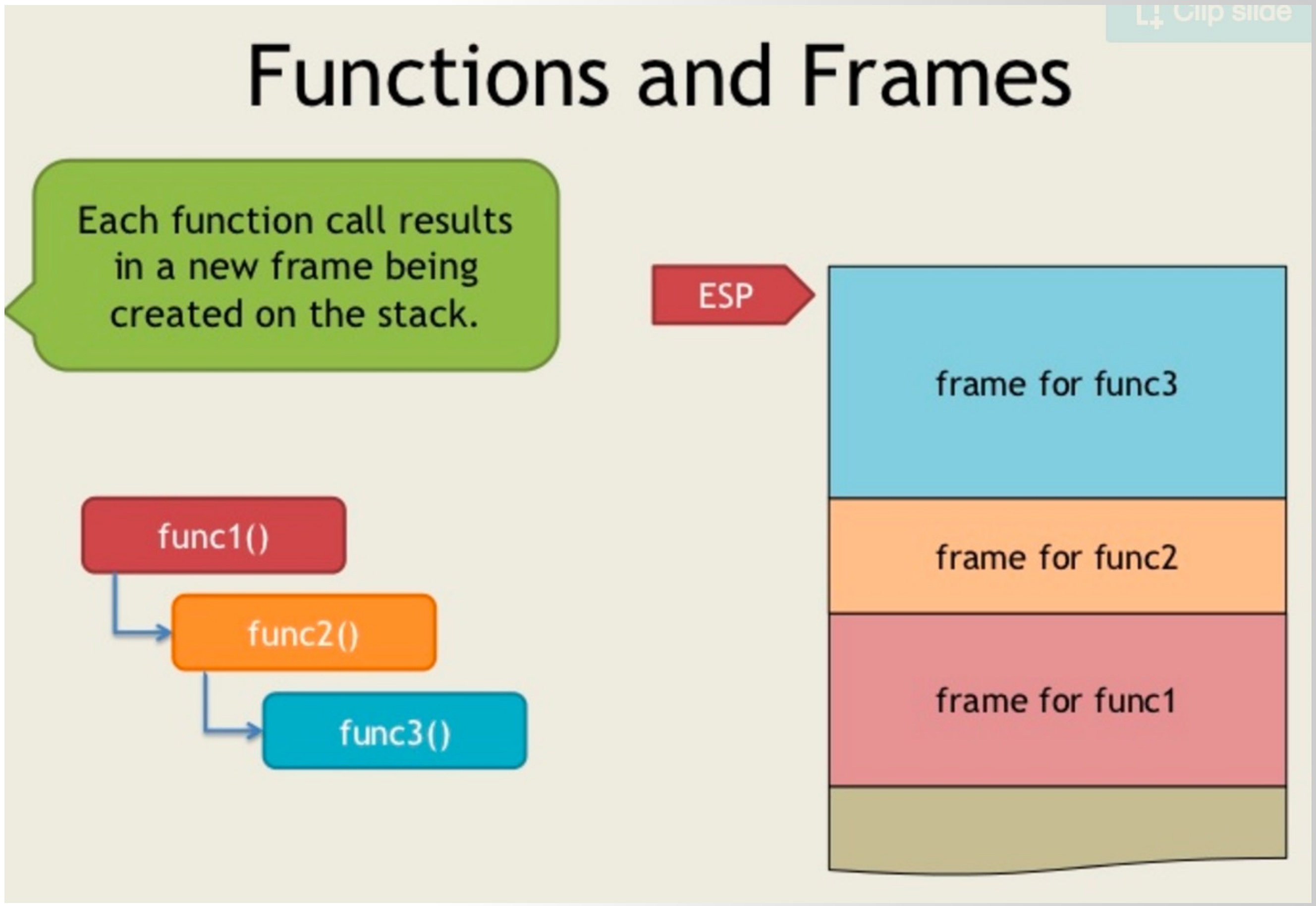
https://www.slideshare.net/saumilshah/how-functions-work-7776073



Functions and Frames

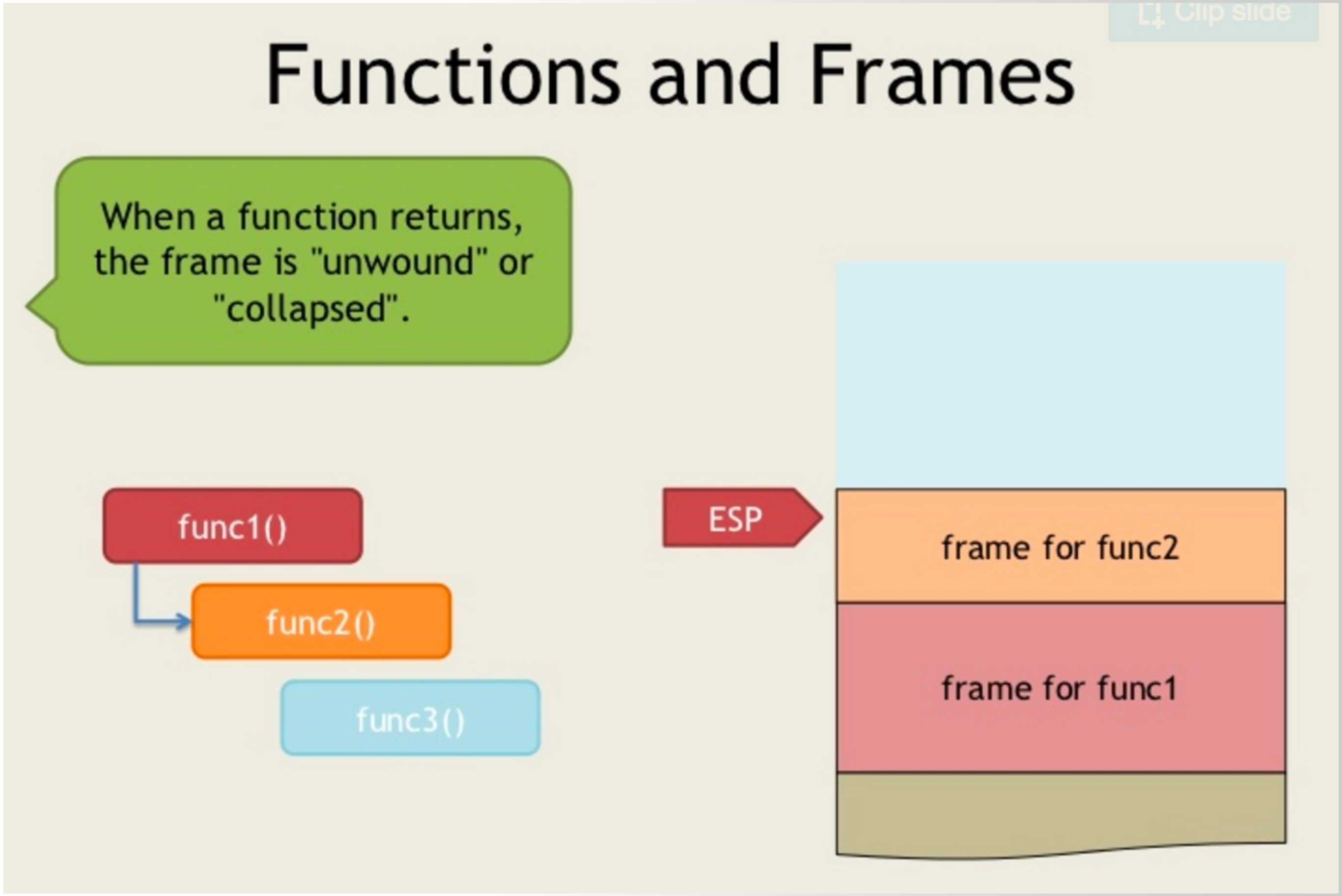
Each function call results in a new frame being created on the stack.

https://www.slideshare.net/saumilshah/how-functions-work-7776073



Functions and Frames

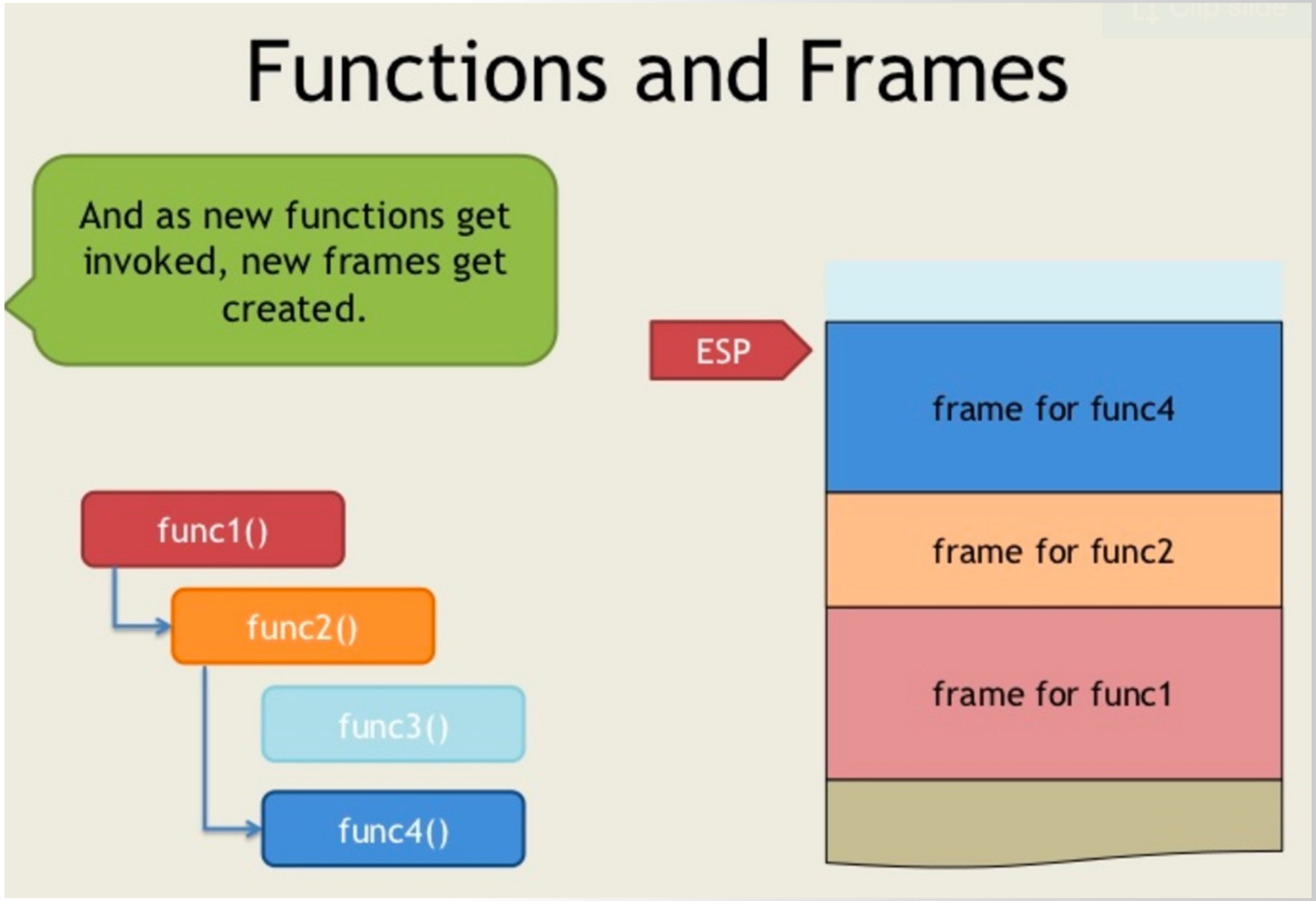
Each function call results in a new frame being created on the stack.



Functions and Frames

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

https://www.slideshare.net/saumilshah/how-functions-work-7776073

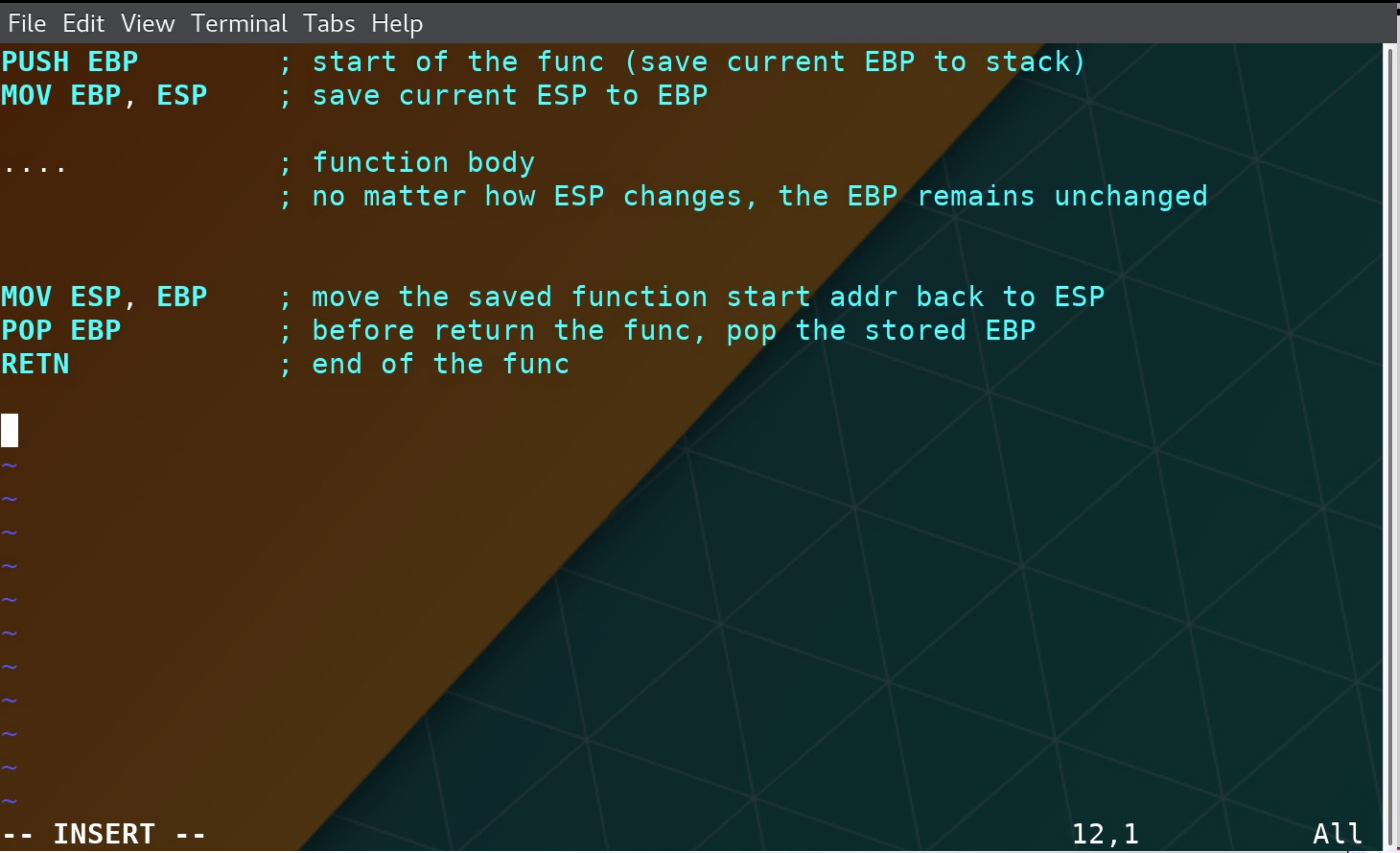


Functions and Frames

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

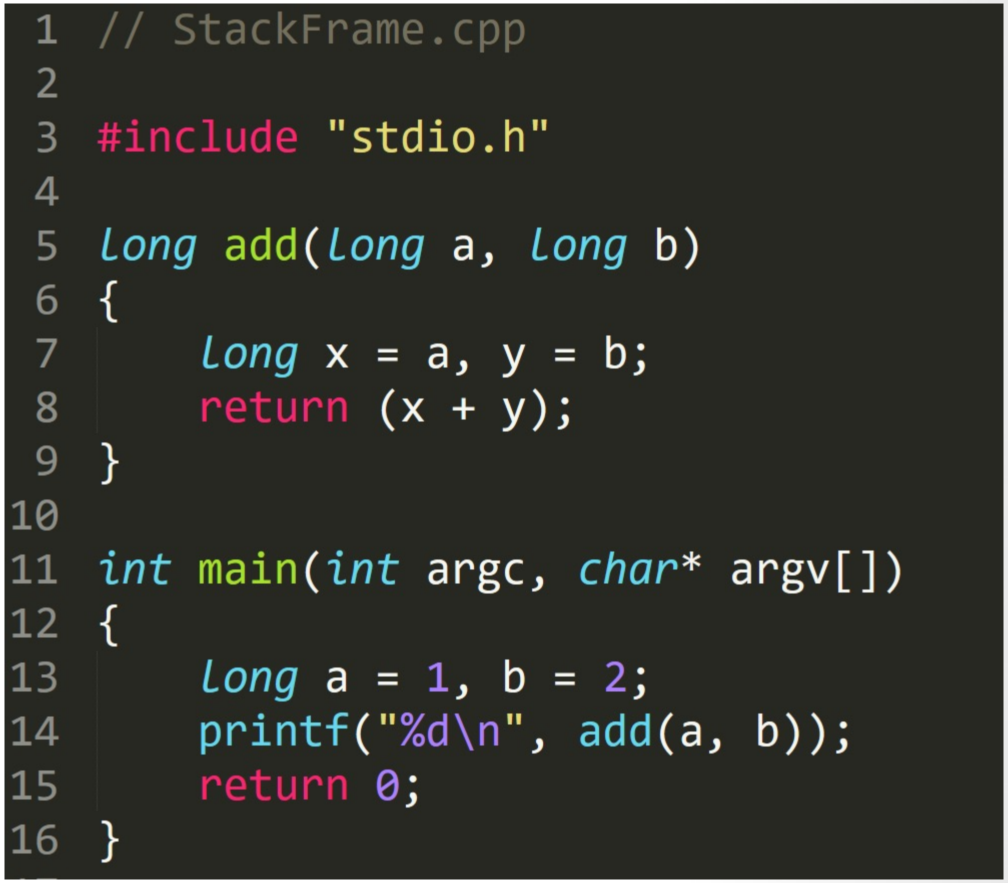
https://www.slideshare.net/saumilshah/how-functions-work-7776073

Stack Frame



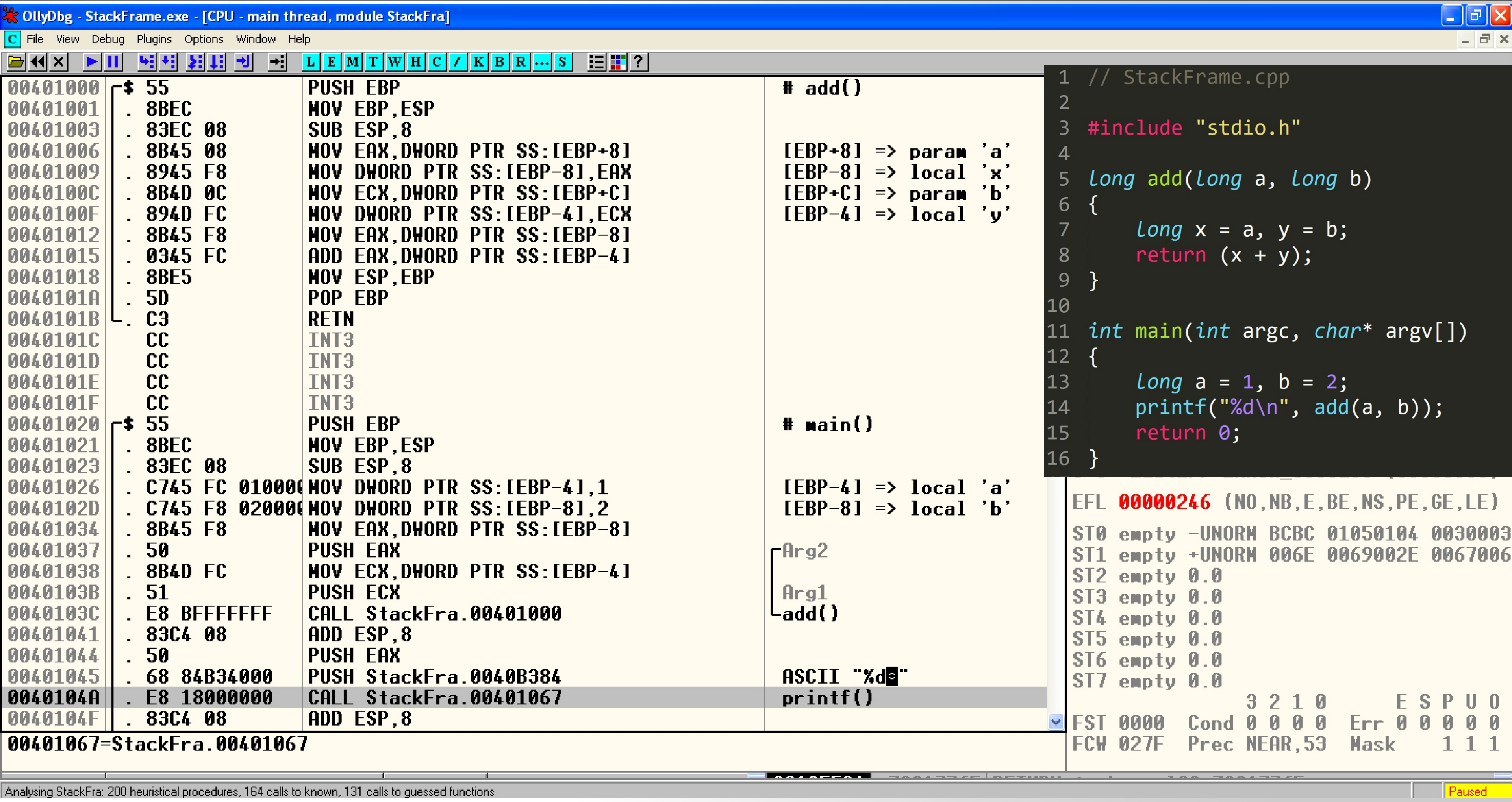
```
PUSH EBP        ; start of the func (save current EBP to stack)
MOV EBP, ESP    ; save current ESP to EBP

....            ; 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



```cpp
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
	long x = a, y = b;
	return (x + y);
}

int main(int argc, char* argv[])
{
	long a = 1, b = 2;
	printf("%d\n", add(a, b));
	return 0;
}
```

StackFrame.exe



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

```
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 010000MOV DWORD PTR SS:[EBP-4],1        [EBP-4] => local 'a'
0040102D │. C745 F8 020000MOV DWORD PTR SS:[EBP-8],2        [EBP-8] => local 'b'
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()
00401041 │. 83C4 08       ADD ESP,8
00401044 │. 50            PUSH EAX
00401045 │. 68 84B34000   PUSH StackFra.0040B384            ASCII "%d"
0040104A │. E8 18000000   CALL StackFra.00401067            printf()
0040104F │. 83C4 08       ADD ESP,8
00401067=StackFra.00401067
```

```cpp
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

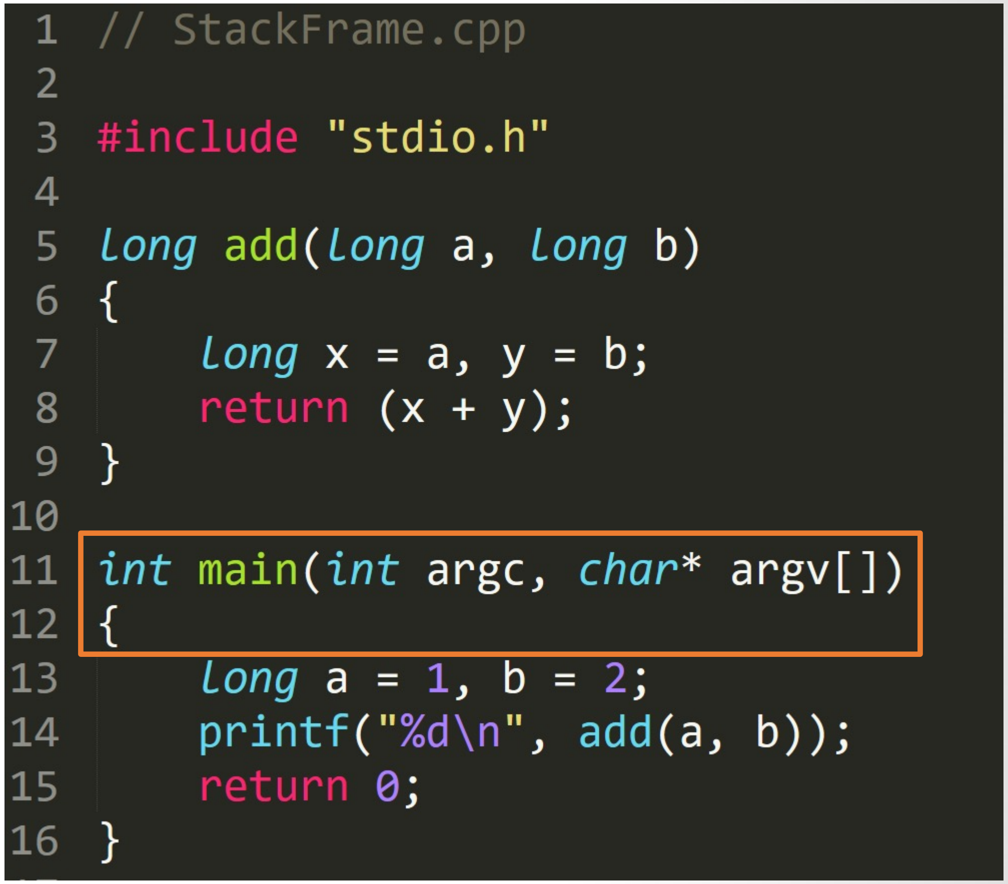
```
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BCBC 01050104 0030003
ST1 empty +UNORM 006E 0069002E 0067006
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1
```

Analysing StackFra: 200 heuristical procedures, 164 calls to known, 131 calls to guessed functions

Paused

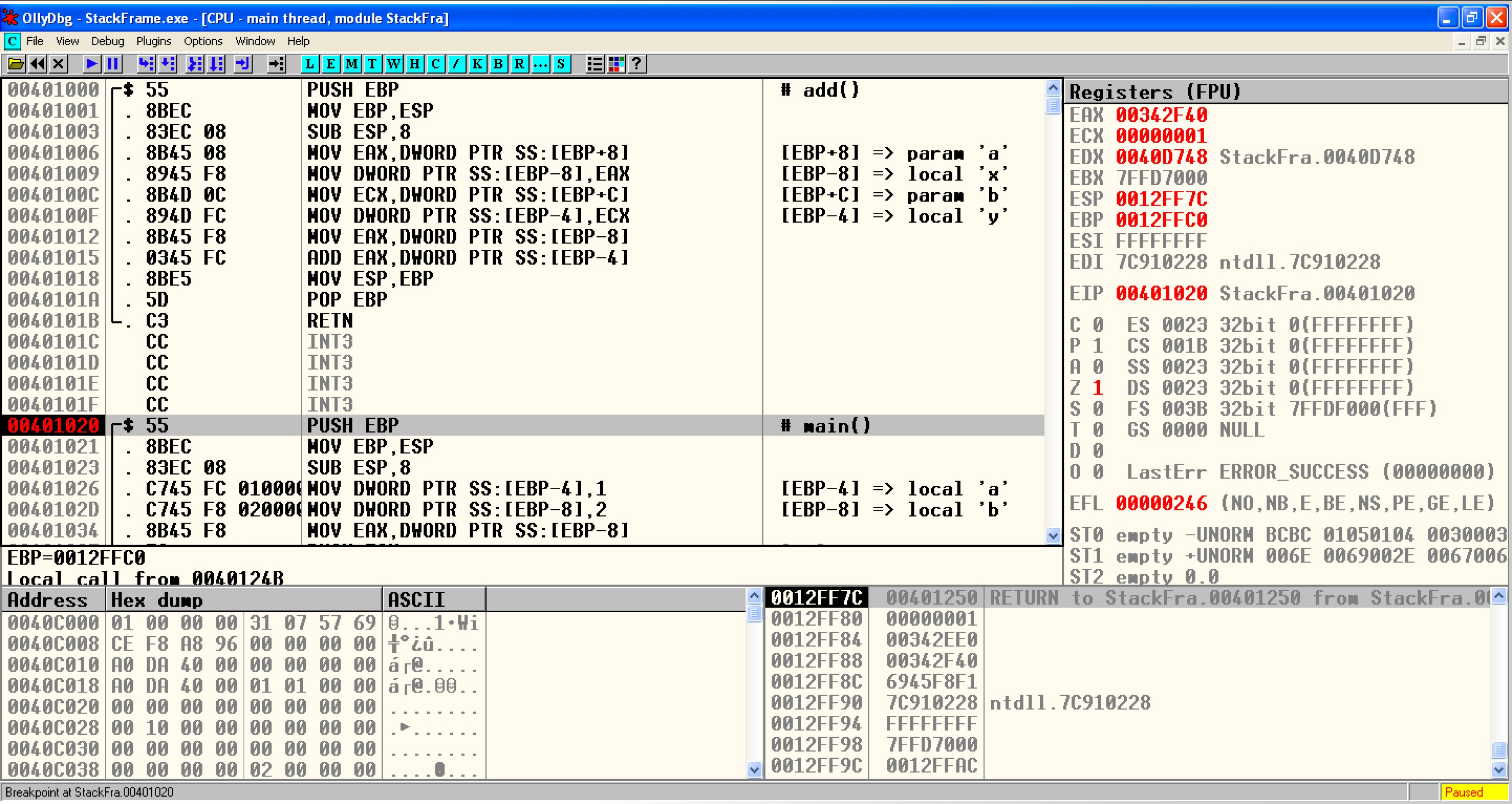
EBP - n: Local vars
EBP + n: Parameters

StackFrame.exe



```
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
	long x = a, y = b;
	return (x + y);
}

int main(int argc, char* argv[])
{
	long a = 1, b = 2;
	printf("%d\n", add(a, b));
	return 0;
}
```

StackFrame.exe



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

```
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 01000(MOV DWORD PTR SS:[EBP-4],1            [EBP-4] => local 'a'
0040102D │. C745 F8 02000(MOV DWORD PTR SS:[EBP-8],2            [EBP-8] => local 'b'
00401034 │. 8B45 F8       MOV EAX,DWORD PTR SS:[EBP-8]
```

```
EBP=0012FFC0
Local call from 0040124B
```

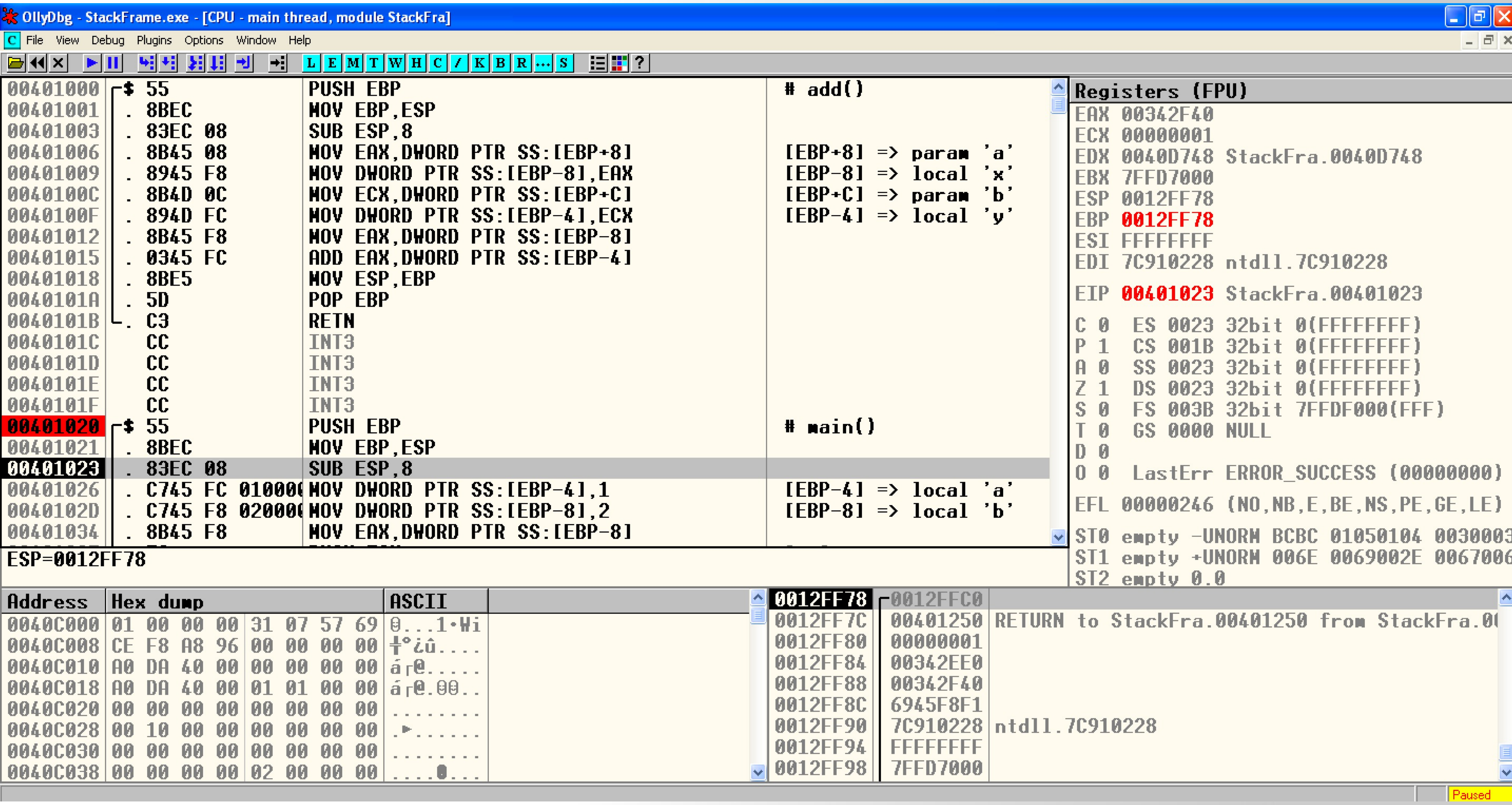
```
Registers (FPU)
EAX 00342F40
ECX 00000001
EDX 0040D748 StackFra.0040D748
EBX 7FFD7000
ESP 0012FF7C
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

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 0030003
ST1 empty +UNORM 006E 0069002E 0067006
ST2 empty 0.0
```

```
Address   Hex dump                 ASCII
0040C000  01 00 00 00 31 07 57 69  ☺...1•Wi
0040C008  CE F8 A8 96 00 00 00 00  ╬°¿û....
0040C010  A0 DA 40 00 00 00 00 00  á┌@.....
0040C018  A0 DA 40 00 01 01 00 00  á┌@.☺☺..
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  ....☻...
```

```
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
```

Breakpoint at StackFra.00401020   Paused

StackFrame.exe



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

```
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 01000 MOV DWORD PTR SS:[EBP-4],1          [EBP-4] => local 'a'
0040102D .  C745 F8 02000 MOV DWORD PTR SS:[EBP-8],2          [EBP-8] => local 'b'
00401034 .  8B45 F8       MOV EAX,DWORD PTR SS:[EBP-8]
ESP=0012FF78
```

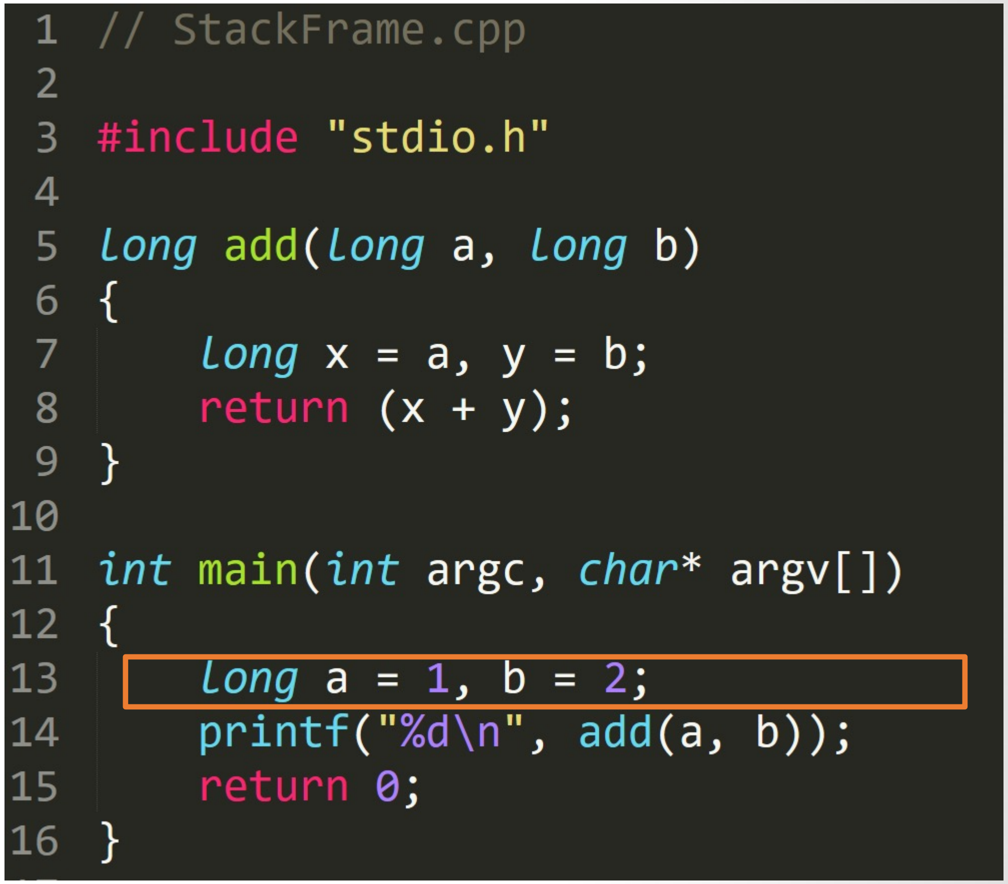
```
Registers (FPU)
EAX 00342F40
ECX 00000001
EDX 0040D748 StackFra.0040D748
EBX 7FFD7000
ESP 0012FF78
EBP 0012FF78
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

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 0030003
ST1 empty +UNORM 006E 0069002E 0067006
ST2 empty 0.0
```

```
Address  Hex dump                 ASCII
0040C000 01 00 00 00 31 07 57 69  ☺...1•Wi
0040C008 CE F8 A8 96 00 00 00 00  ╬°¿û....
0040C010 A0 DA 40 00 00 00 00 00  á┌@.....
0040C018 A0 DA 40 00 01 01 00 00  á┌@.☺☺..
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  ....☻...
```

```
0012FF78 ┌0012FFC0
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
```

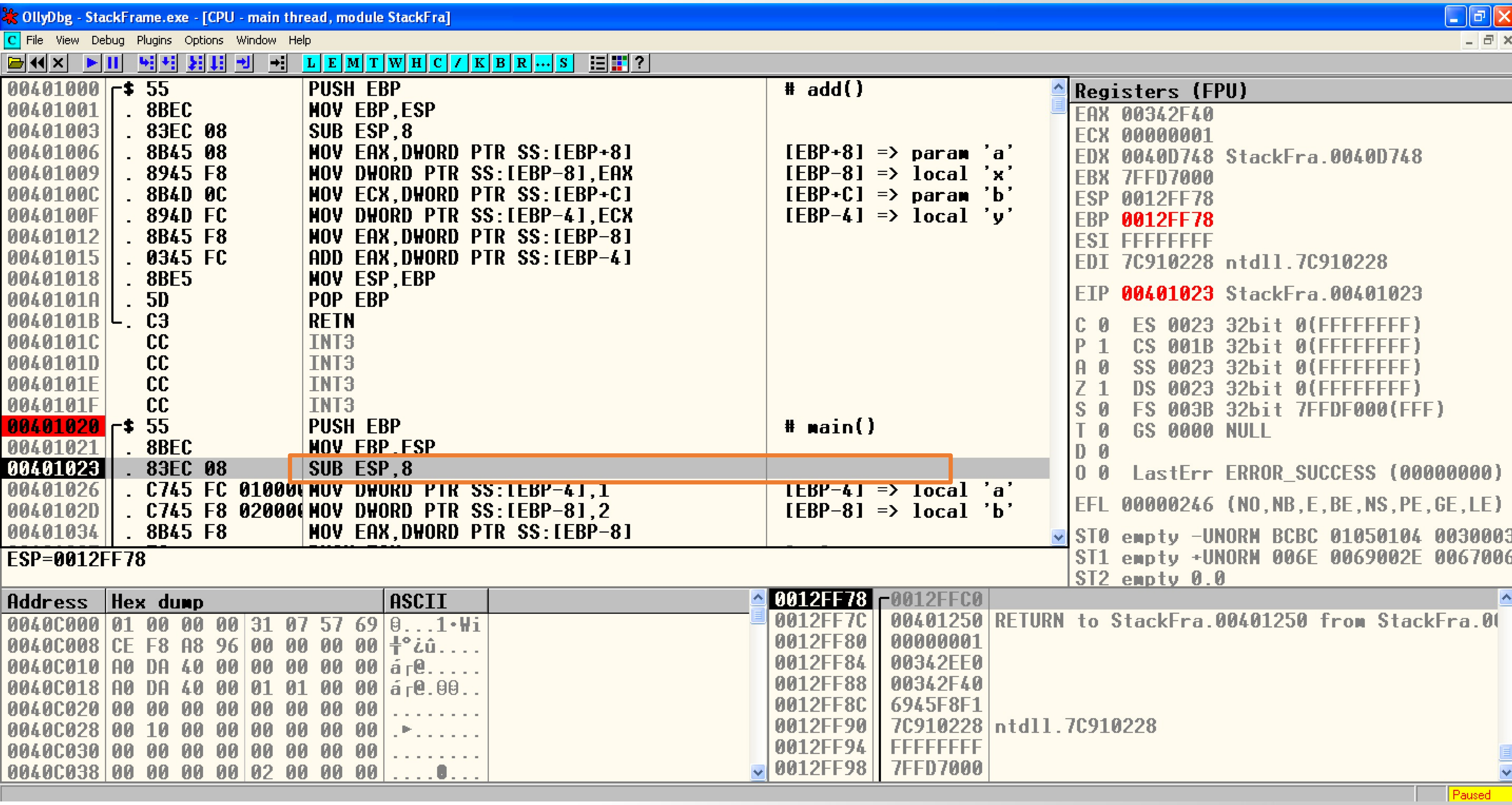
Paused

StackFrame.exe



```
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

StackFrame.exe


OllyDbg - StackFrame.exe - [CPU - main thread, module StackFra]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Paused

```
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 01000 MOV DWORD PTR SS:[EBP-4],1            [EBP-4] => local 'a'
0040102D  │. C745 F8 02000 MOV DWORD PTR SS:[EBP-8],2            [EBP-8] => local 'b'
00401034  │. 8B45 F8       MOV EAX,DWORD PTR SS:[EBP-8]
ESP=0012FF78
```

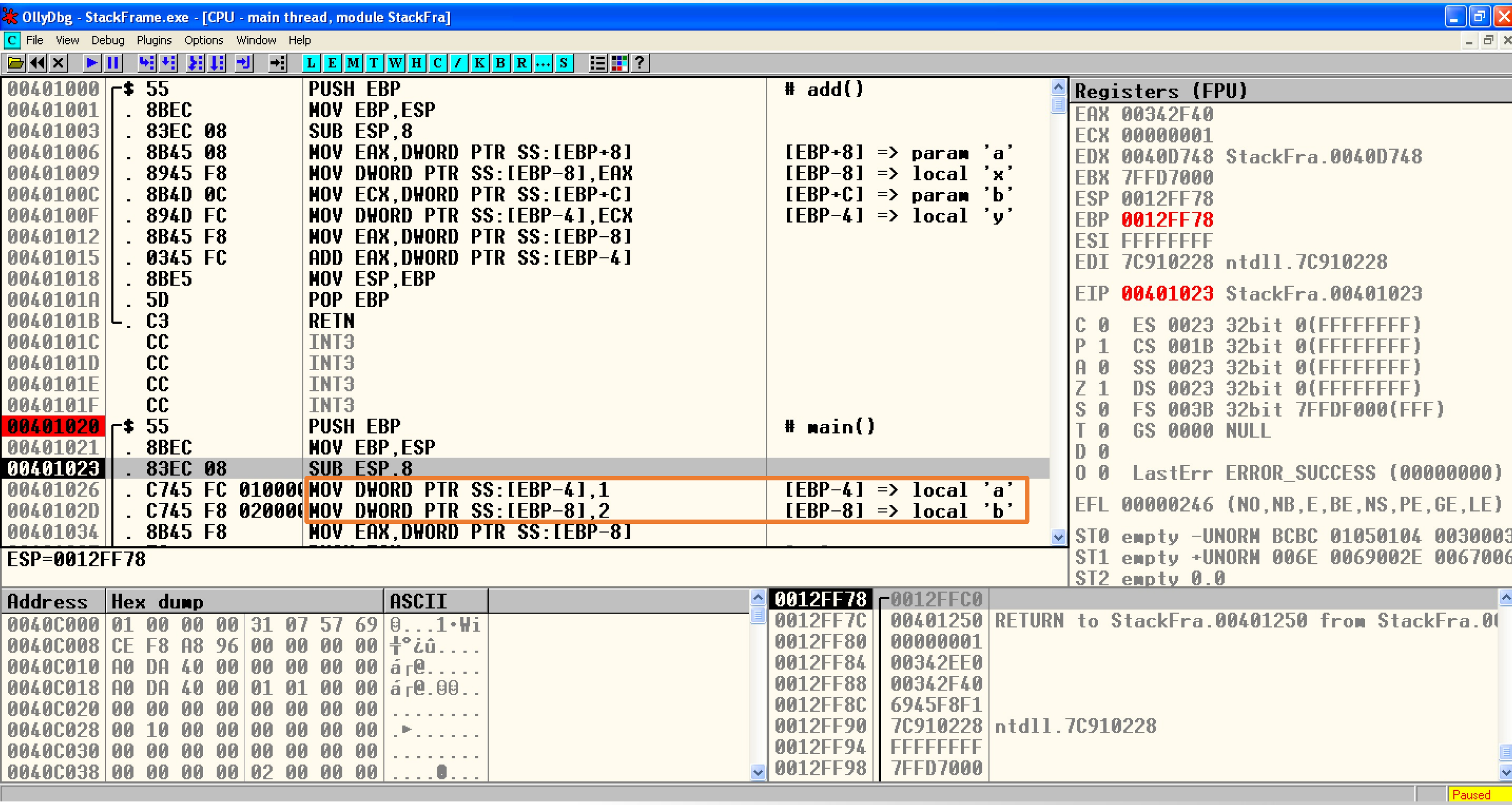
```
Registers (FPU)
EAX 00342F40
ECX 00000001
EDX 0040D748 StackFra.0040D748
EBX 7FFD7000
ESP 0012FF78
EBP 0012FF78
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

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 0030003
ST1 empty +UNORM 006E 0069002E 0067006
ST2 empty 0.0
```

```
Address   Hex dump                 ASCII
0040C000  01 00 00 00 31 07 57 69  ☺...1•Wi
0040C008  CE F8 A8 96 00 00 00 00  ╪°¿û....
0040C010  A0 DA 40 00 00 00 00 00  á┌@.....
0040C018  A0 DA 40 00 01 01 00 00  á┌@.☺☺..
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  ....☻...
```

```
0012FF78  ┌0012FFC0
0012FF7C  │00401250  RETURN to StackFra.00401250 from StackFra.00
0012FF80  │00000001
0012FF84  │00342EE0
0012FF88  │00342F40
0012FF8C  │6945F8F1
0012FF90  │7C910228  ntdll.7C910228
0012FF94  │FFFFFFFF
0012FF98  │7FFD7000
```

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
L E M T W H C / K B R ... S
Paused

```
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 01000MOV DWORD PTR SS:[EBP-4],1        [EBP-4] => local 'a'
0040102D │. C745 F8 02000MOV DWORD PTR SS:[EBP-8],2        [EBP-8] => local 'b'
00401034 │. 8B45 F8      MOV EAX,DWORD PTR SS:[EBP-8]
ESP=0012FF78
```

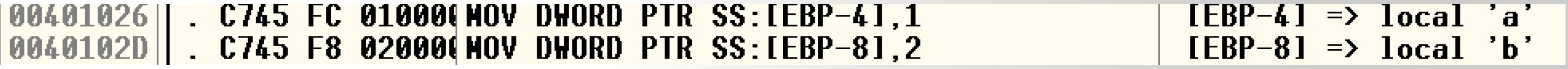
```
Registers (FPU)
EAX 00342F40
ECX 00000001
EDX 0040D748 StackFra.0040D748
EBX 7FFD7000
ESP 0012FF78
EBP 0012FF78
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

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 0030003
ST1 empty +UNORM 006E 0069002E 0067000
ST2 empty 0.0
```

```
Address  Hex dump                 ASCII
0040C000 01 00 00 00 31 07 57 69  ☺...1•Wi
0040C008 CE F8 A8 96 00 00 00 00  ╬°¿û....
0040C010 A0 DA 40 00 00 00 00 00  á┌@.....
0040C018 A0 DA 40 00 01 01 00 00  á┌@.☺☺..
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  ....☻...
```

```
0012FF78 ┌0012FFC0
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
```

Create space for ‘a’ and ‘b’ → long → 4 byte

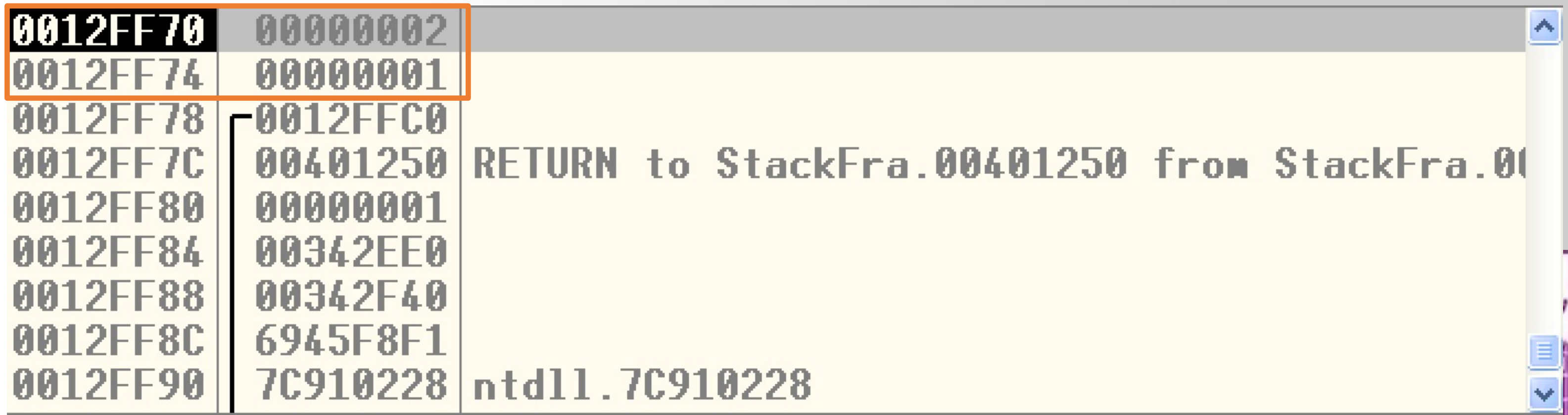
StackFrame.exe



```
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'
```

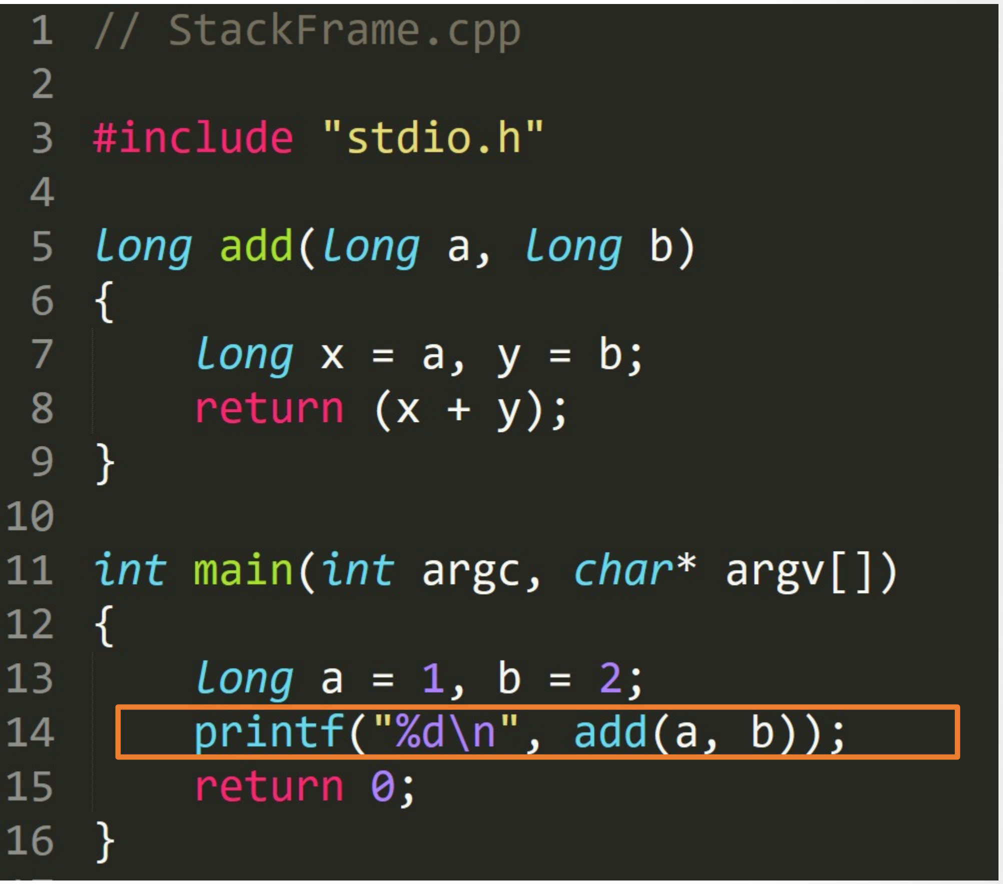
Assembly	C	Type Conversion
DWORD PTR SS:[EBP-4]	*(DWORD*)(EBP-4)	DWORD (4 byte)
WORD PTR SS:[EBP-4]	*(WORD*)(EBP-4)	WORD (2 byte)
BYTE PTR SS:[EBP-4]	*(BYTE*)(EBP-4)	1 byte

4 Byte memory space at address [EBP-4]

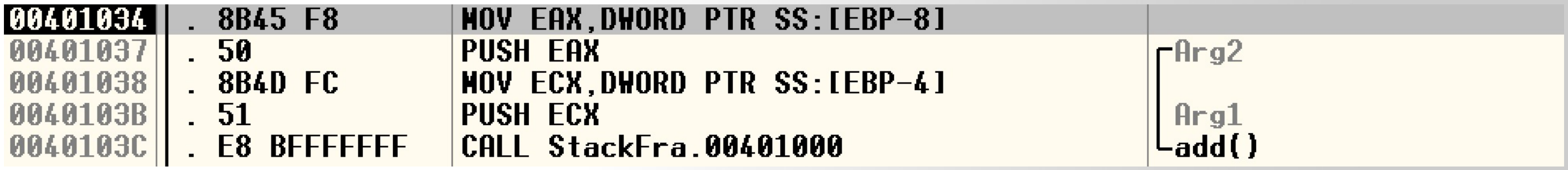


```
0012FF70   00000002
0012FF74   00000001
0012FF78  ┌0012FFC0
0012FF7C  │00401250  RETURN to StackFra.00401250 from StackFra.0(
0012FF80  │00000001
0012FF84  │00342EE0
0012FF88  │00342F40
0012FF8C  │6945F8F1
0012FF90  │7C910228  ntdll.7C910228
```

StackFrame.exe

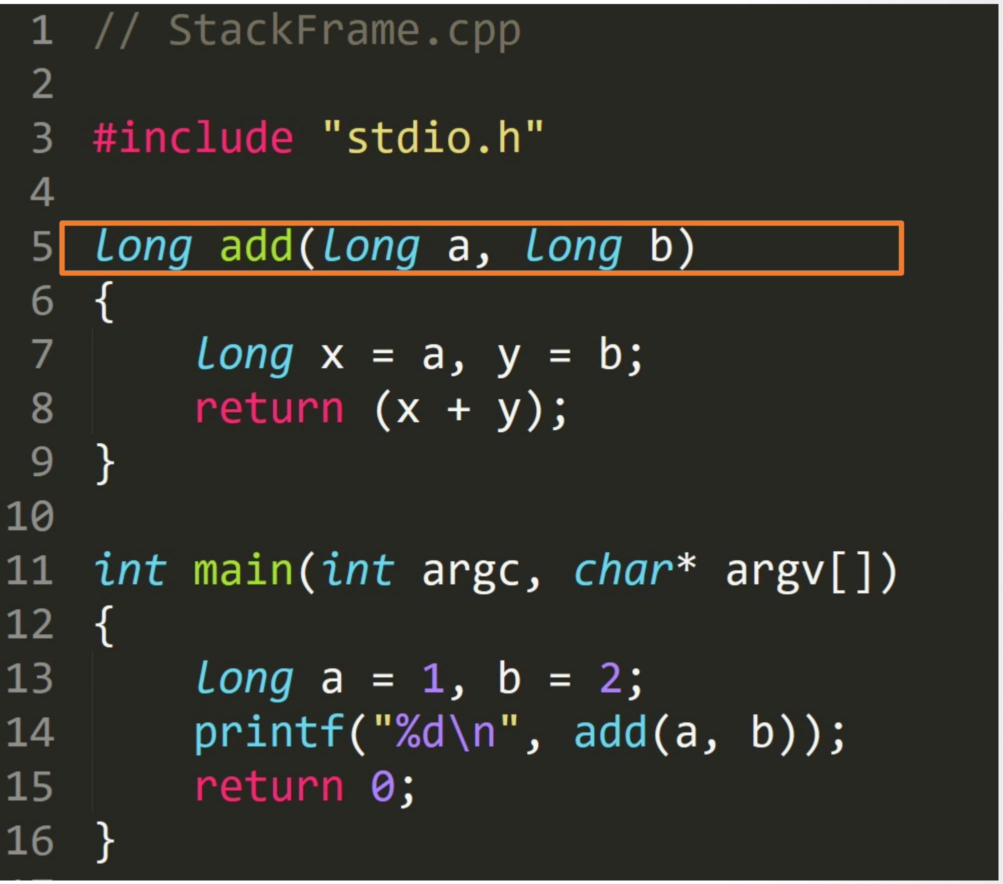


```cpp
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

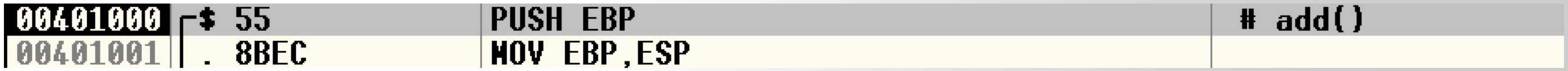


```
00401034  . 8B45 F8      MOV EAX,DWORD PTR SS:[EBP-8]
00401037  . 50           PUSH EAX                         ┌Arg2
00401038  . 8B4D FC      MOV ECX,DWORD PTR SS:[EBP-4]     │
0040103B  . 51           PUSH ECX                         │Arg1
0040103C  . E8 BFFFFFFF  CALL StackFra.00401000       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

StackFrame.exe

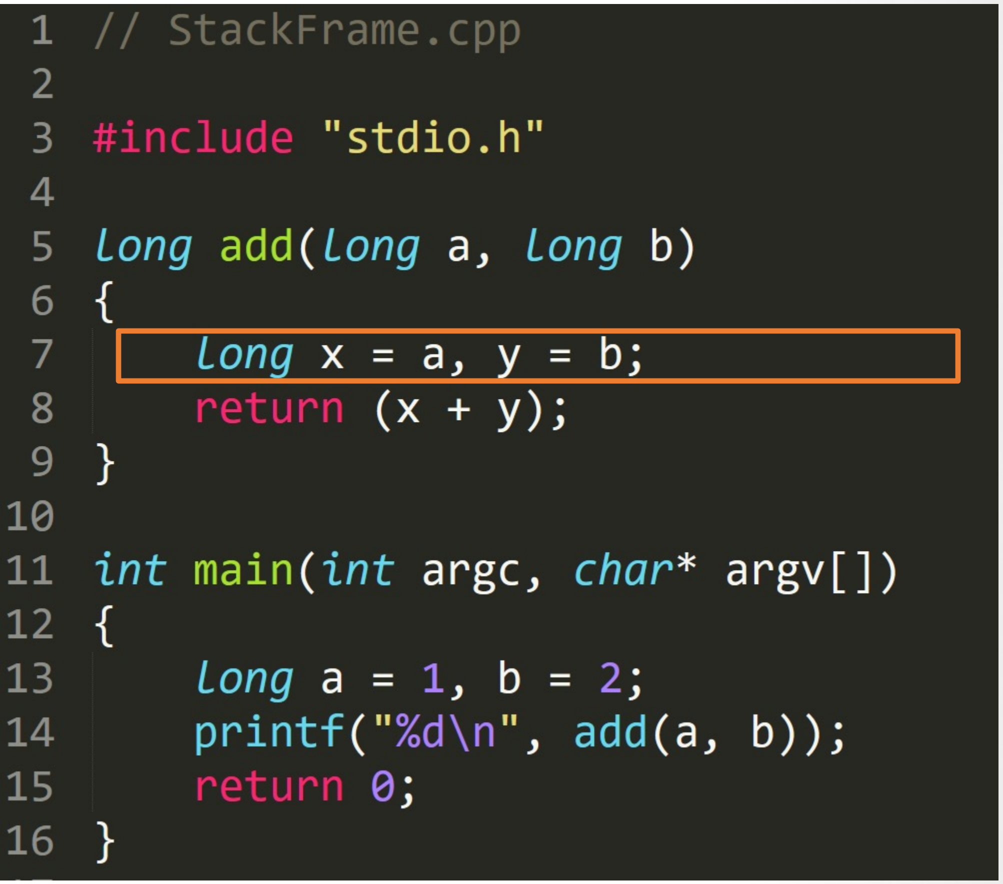


```
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

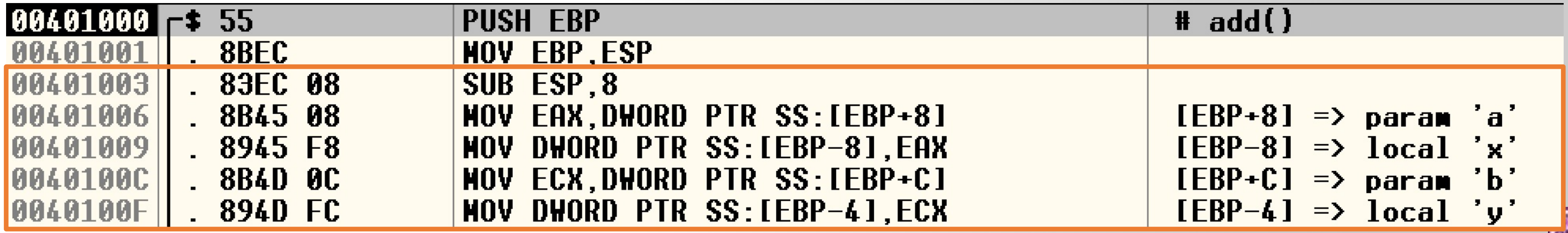


```
00401000  ┌$ 55      PUSH EBP          # add()
00401001  │. 8BEC    MOV EBP,ESP
```

StackFrame.exe

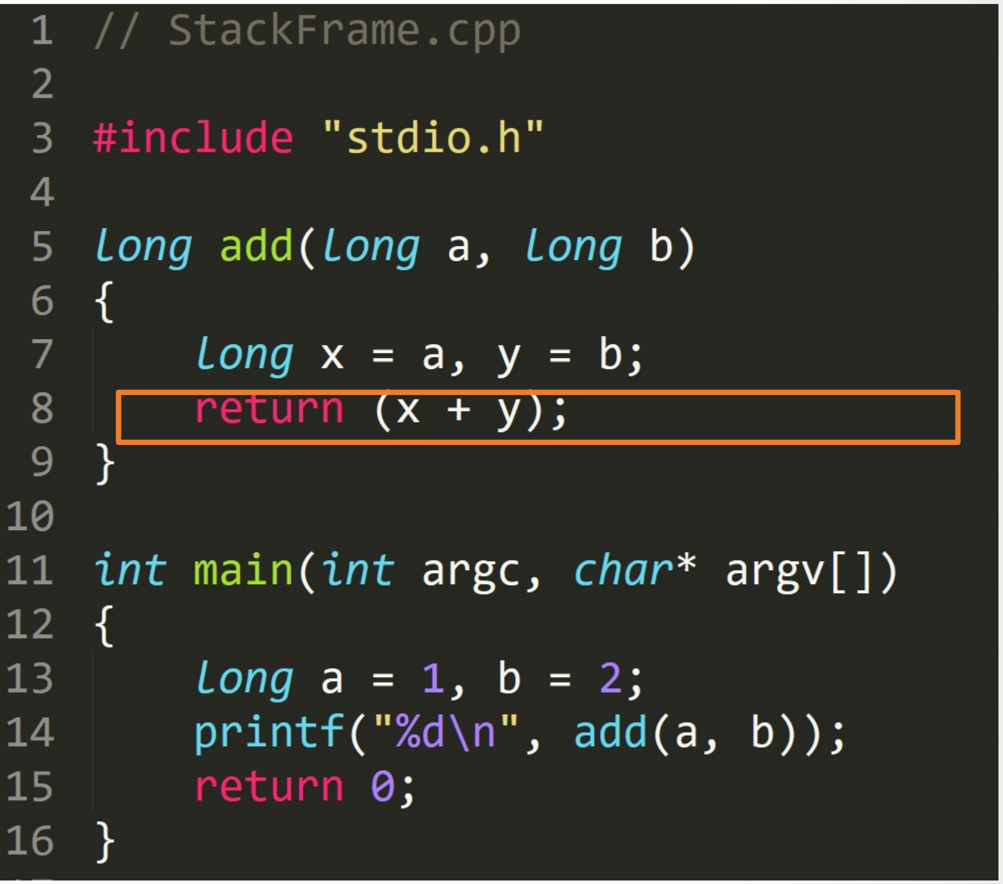


```cpp
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

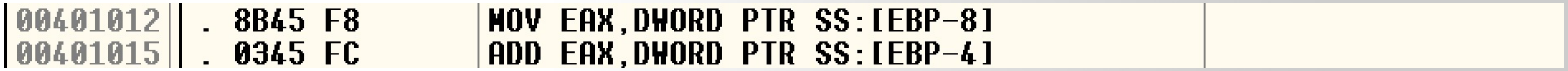


```
00401000  ┌$ 55         PUSH EBP                           # add()
00401001  │. 8BEC       MOV EBP,ESP
00401003  │. 83EC 08    SUB ESP,8
00401006  │. 8B45 08    MOV EAX,DWORD PTR SS:[EBP+8]       [EBP+8] => param 'a'
00401009  │. 8945 F8    MOV DWORD PTR SS:[EBP-8],EAX       [EBP-8] => local 'x'
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

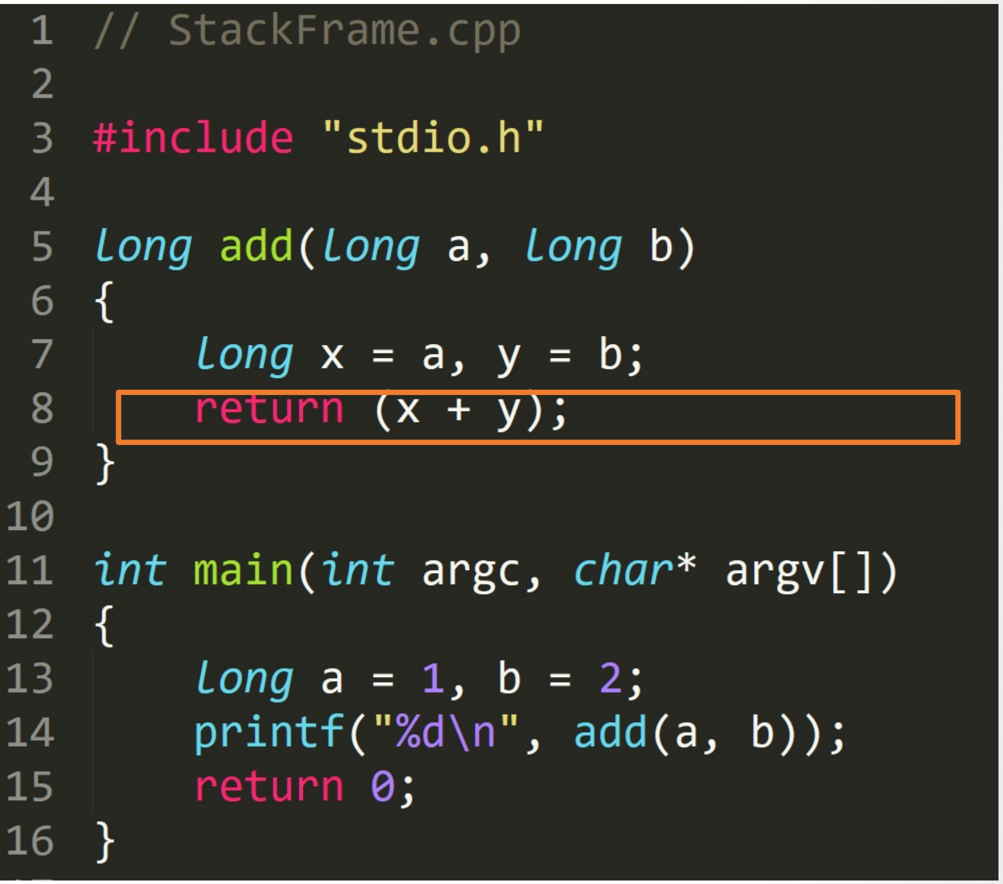


```
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

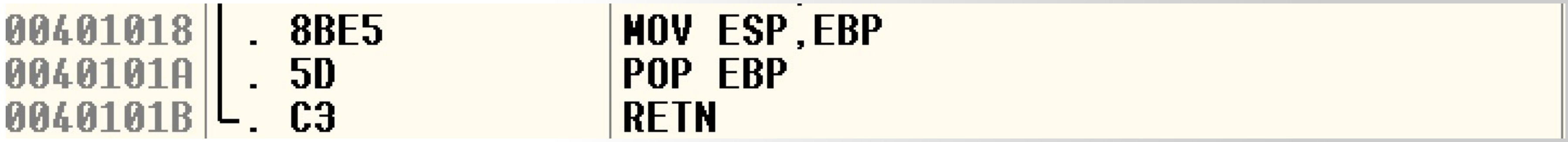


```
00401012  .  8B45 F8     MOV EAX,DWORD PTR SS:[EBP-8]
00401015  .  0345 FC     ADD EAX,DWORD PTR SS:[EBP-4]
```

StackFrame.exe

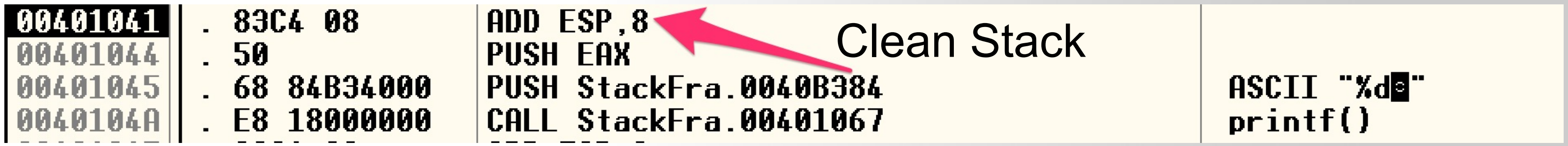


```
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```



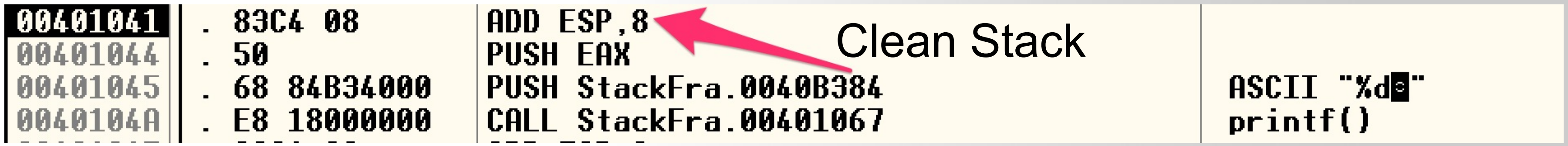
```
00401018 │ . 8BE5          MOV ESP,EBP
0040101A │ . 5D            POP EBP
0040101B └ . C3            RETN
```

StackFrame.exe



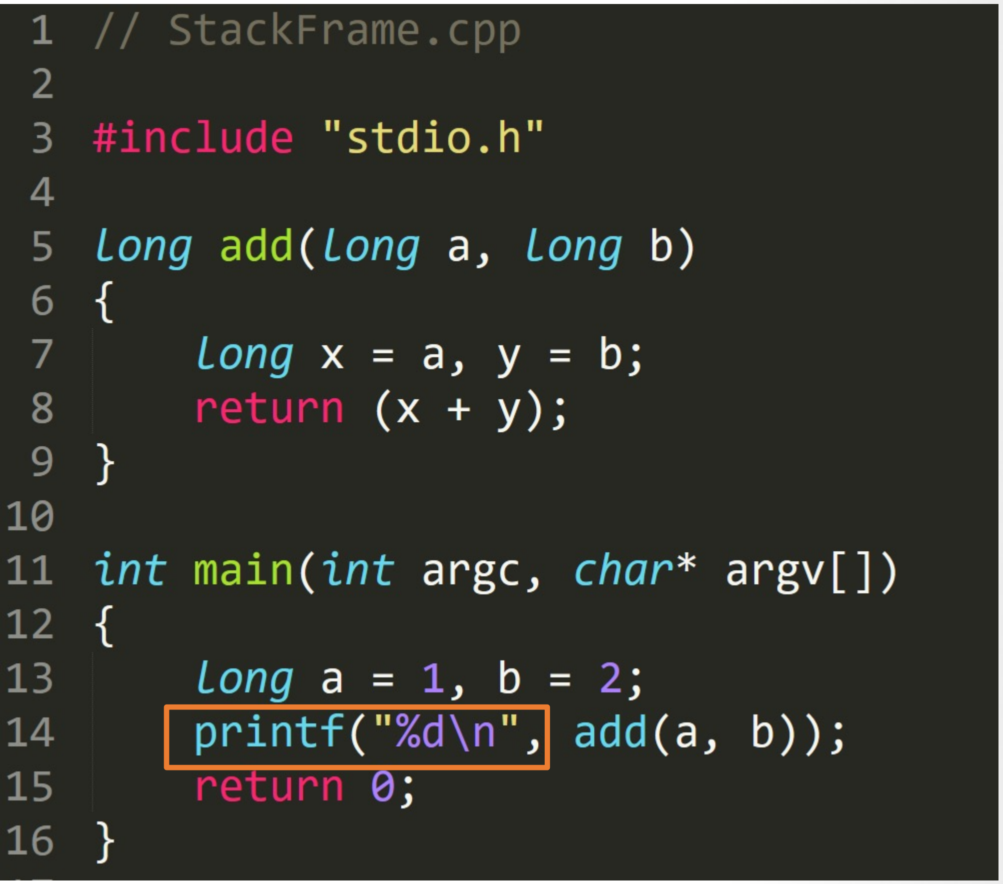
```
00401041  . 83C4 08        ADD ESP,8
00401044  . 50             PUSH EAX
00401045  . 68 84B34000    PUSH StackFra.0040B384              ASCII "%d"
0040104A  . E8 18000000    CALL StackFra.00401067              printf()
```

StackFrame.exe

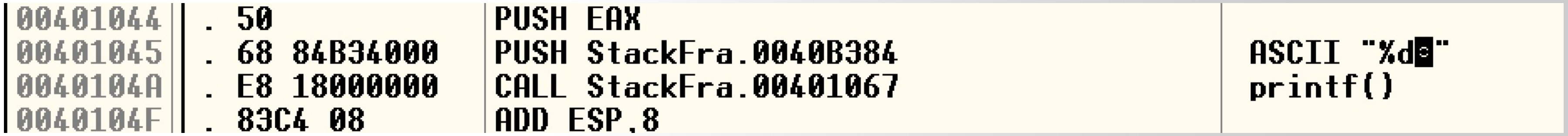


```
00401041  . 83C4 08      ADD ESP,8
00401044  . 50           PUSH EAX
00401045  . 68 84B34000  PUSH StackFra.0040B384                  ASCII "%d"
0040104A  . E8 18000000  CALL StackFra.00401067                  printf()
```

StackFrame.exe

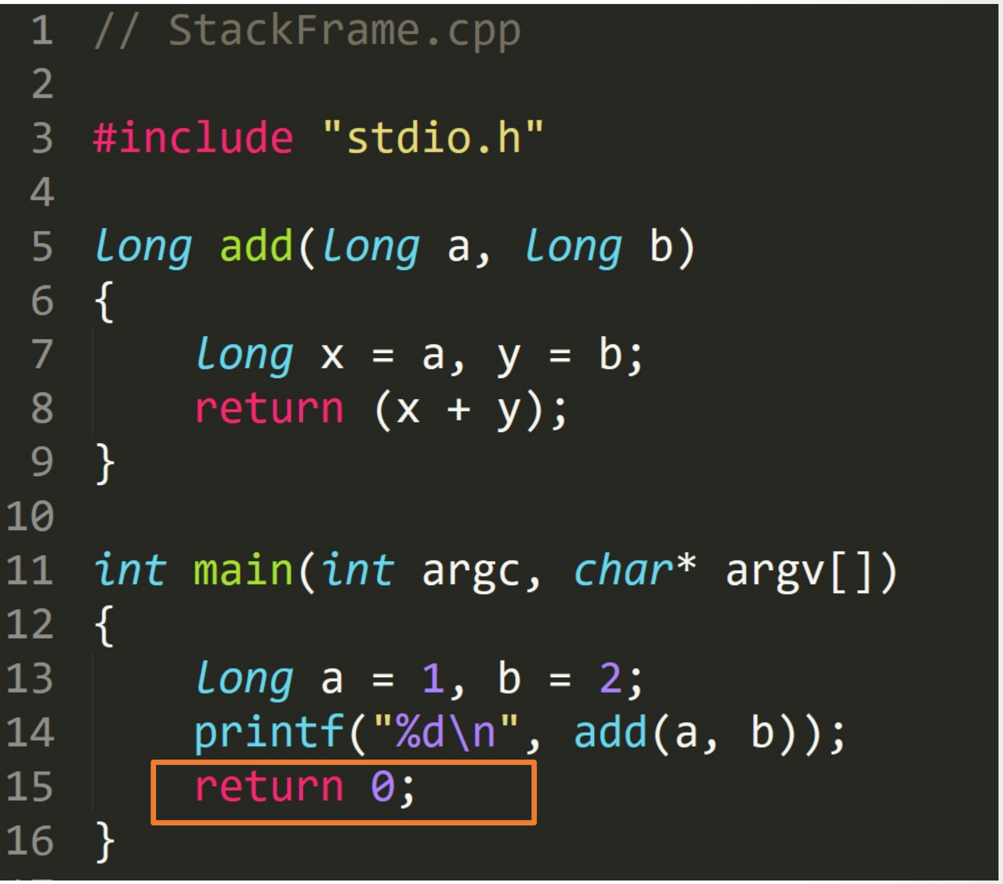


```cpp
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

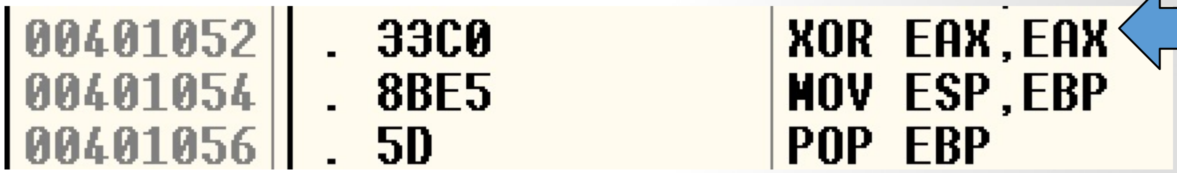


```
00401044 | . 50          PUSH EAX
00401045 | . 68 84B34000 PUSH StackFra.0040B384       ASCII "%d█"
0040104A | . E8 18000000 CALL StackFra.00401067       printf()
0040104F | . 83C4 08     ADD ESP,8
```

StackFrame.exe



```
// StackFrame.cpp

#include "stdio.h"

long add(long a, long b)
{
    long x = a, y = b;
    return (x + y);
}

int main(int argc, char* argv[])
{
    long a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```



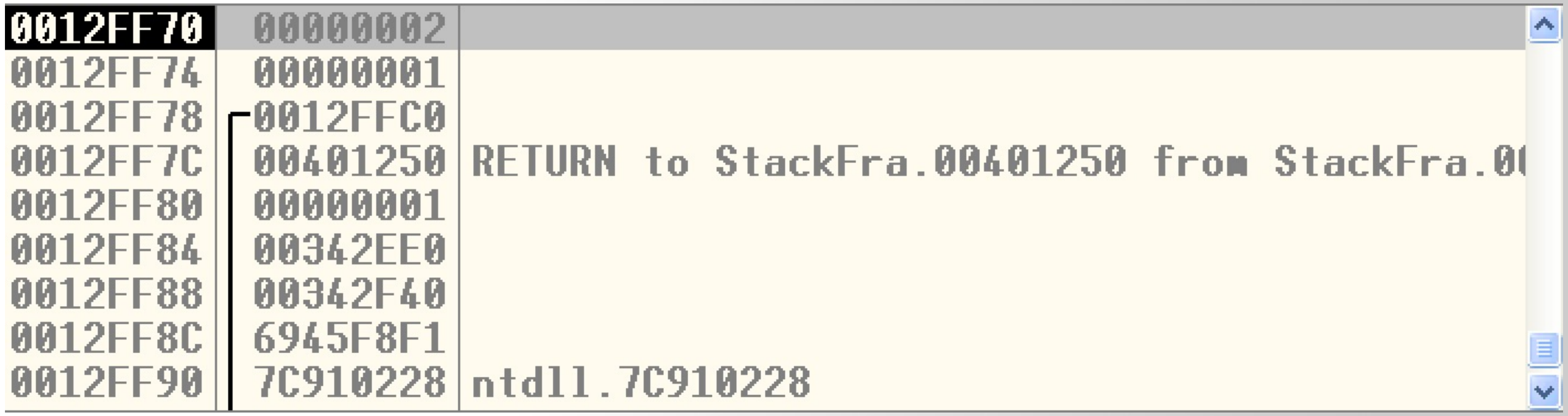
```
00401052 | . 33C0    XOR EAX,EAX
00401054 | . 8BE5    MOV ESP,EBP
00401056 | . 5D      POP EBP
```

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.0(
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

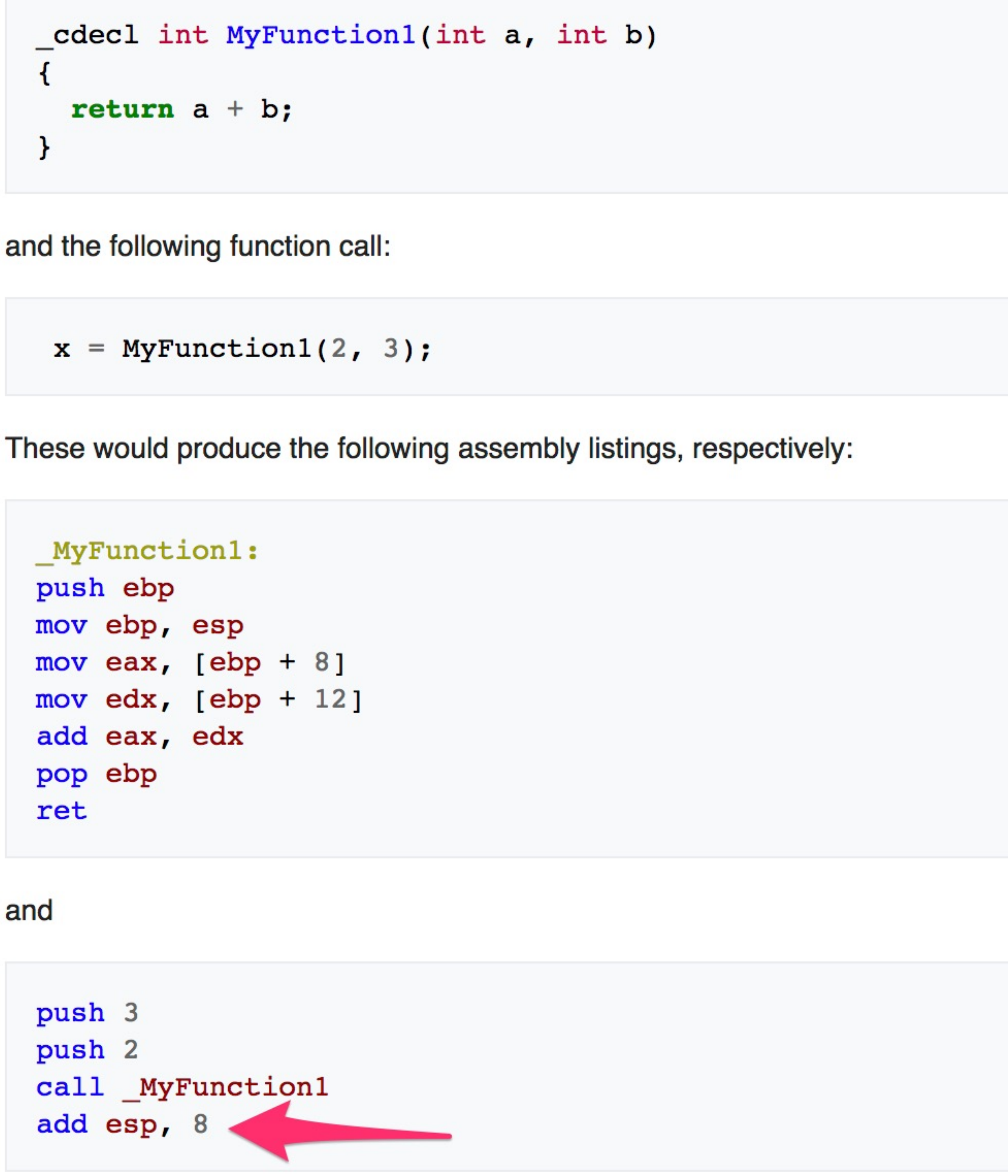
- **Calling conventions** are a standardized method for functions to be implemented and called by the machine.
- A calling convention specifies the method that a compiler sets up to access a subroutine.
- There are three major calling conventions that are used with the C language on 32-bit x86 processors: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

CDECL

- 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

- The C language, by defa
- In the CDECL calling cor
  - Arguments are passed oı passed in eax.
  - The ***calling* function cle** have *variable-length argu*



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

and the following function call:

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

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
```

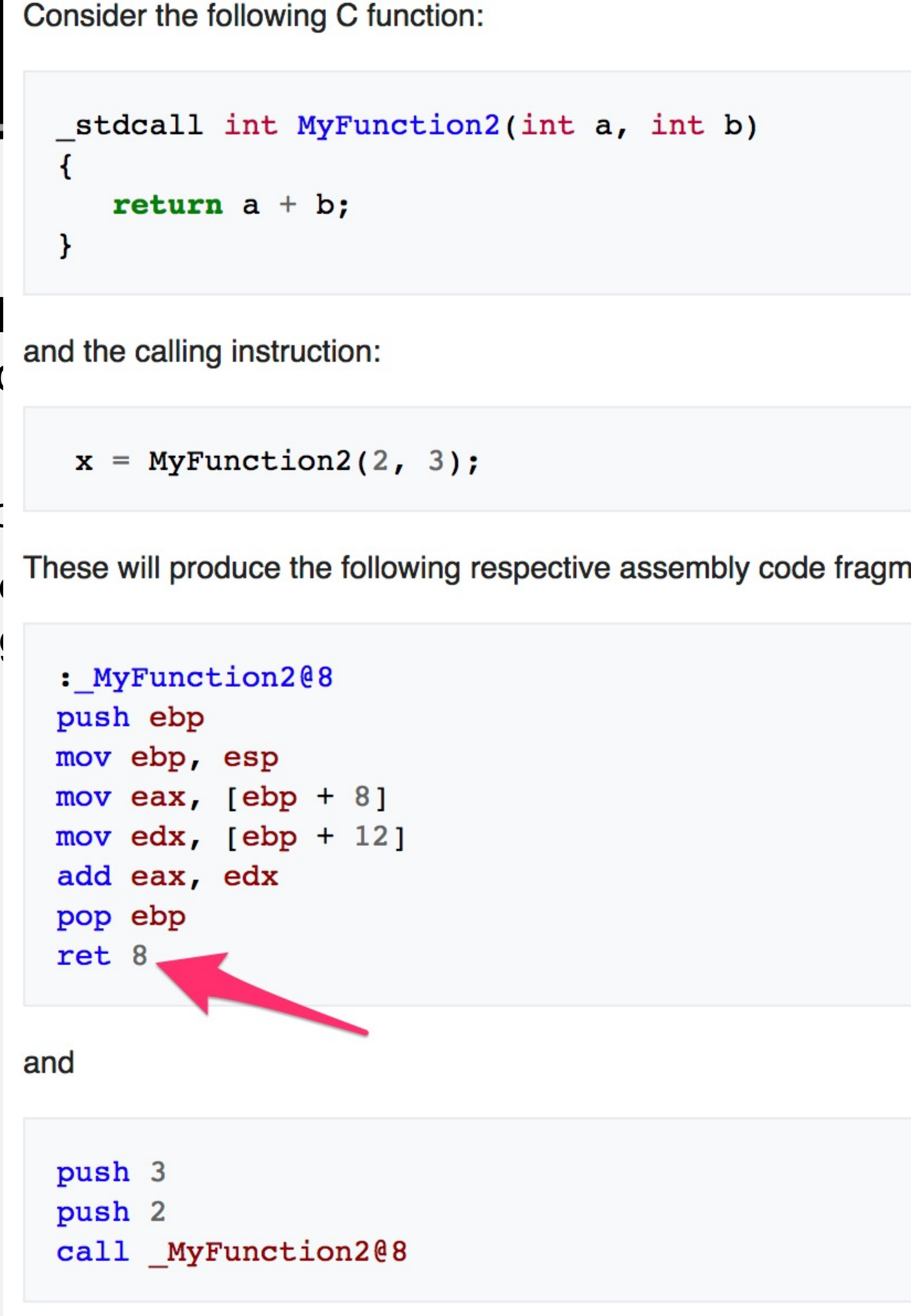
STDCALL

- 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 usec
  the standard calling convention for
  - STDCALL passes arguments right-tc
  - The **called function cleans the sta**
    STDCALL doesn't allow variable-len

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 fragm

```
:_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
```

FASTCALL

- 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