# Practical Foundations of Debugging

## Chapter 7

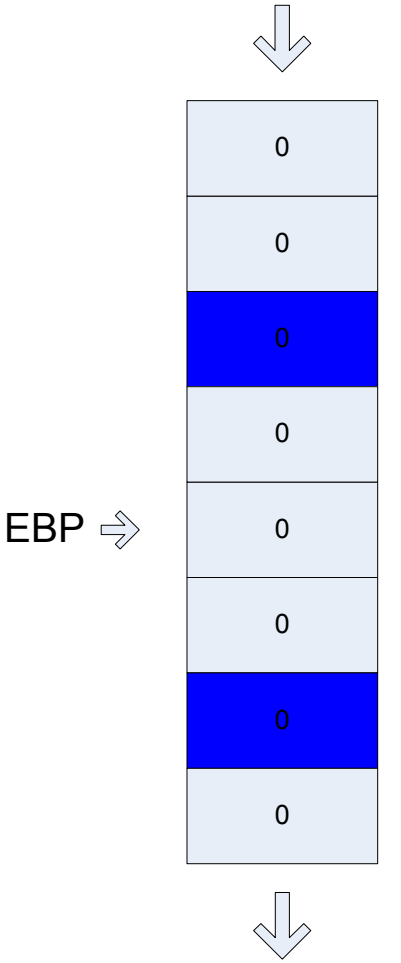
## Function Parameters

Designed, written and illustrated by Dmitry Vostokov

# “Arithmetical” project

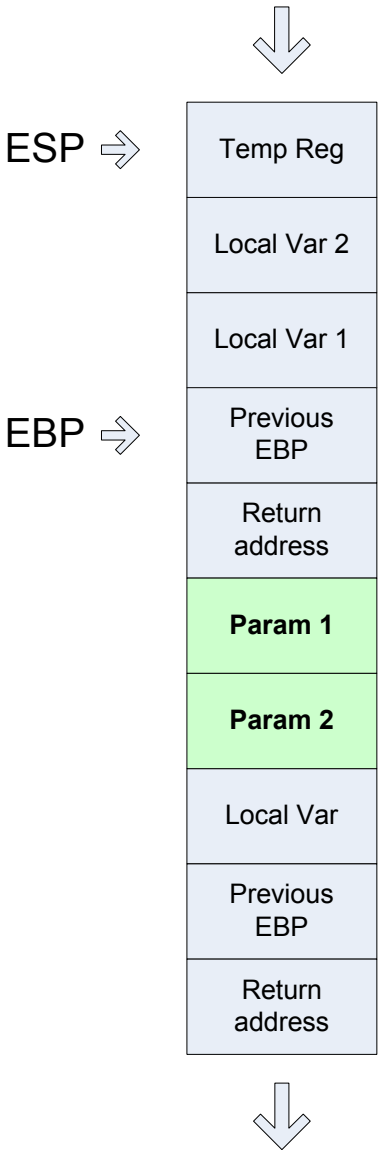
```
// FunctionParameters.cpp
int arithmetic (int a, int b);

int main(int argc, char* argv[])
{
   int result = arithmetic (1, 1);
   return 0;
}

// Arithmetic.cpp
int arithmetic (int a, int b)
{
   b = b + a;
   ++a;
   b = a * b;

   return b;
}
```

# Addressing array elements




| | Address of element | Value of element |
|---|---|---|
| 0x10001000 | EBP-10 | [EBP-10] |
| 0x10001004 | EBP-C | [EBP-C] |
| 0x10001008 | EBP-8 | [EBP-8] |
| 0x1000100C | EBP-4 | [EBP-4] |
| 0x10001010 | EBP | [EBP] |
| 0x10001014 | EBP+4 | [EBP+4] |
| 0x10001018 | EBP+8 | [EBP+8] |
| 0x1000101C | EBP+C | [EBP+C] |

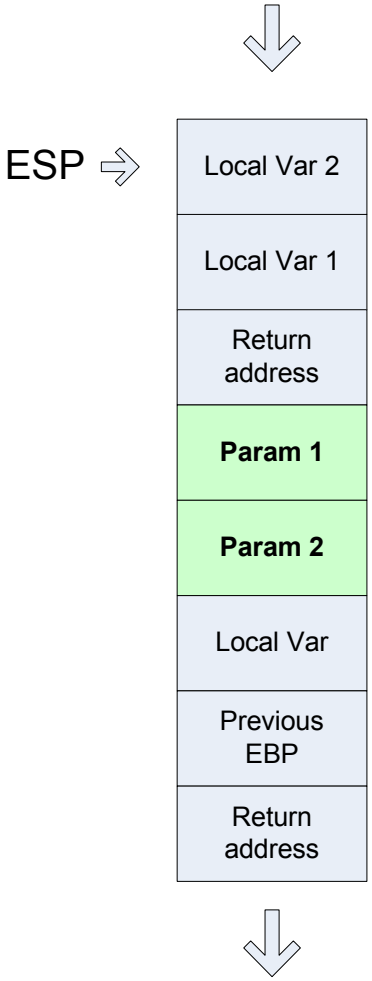
# Stack structure




```
...
[EBP-8] Local Var 2 (DWORD)
[EBP-4] Local Var 1 (DWORD)
[EBP]   previous EBP
[EBP+4] return address
[EBP+8] Param 1 (DWORD)
[EBP+C] Param 2 (DWORD)
...
```

```
void func(int Param1, int Param2)
{
   int var1, var2;
}
```

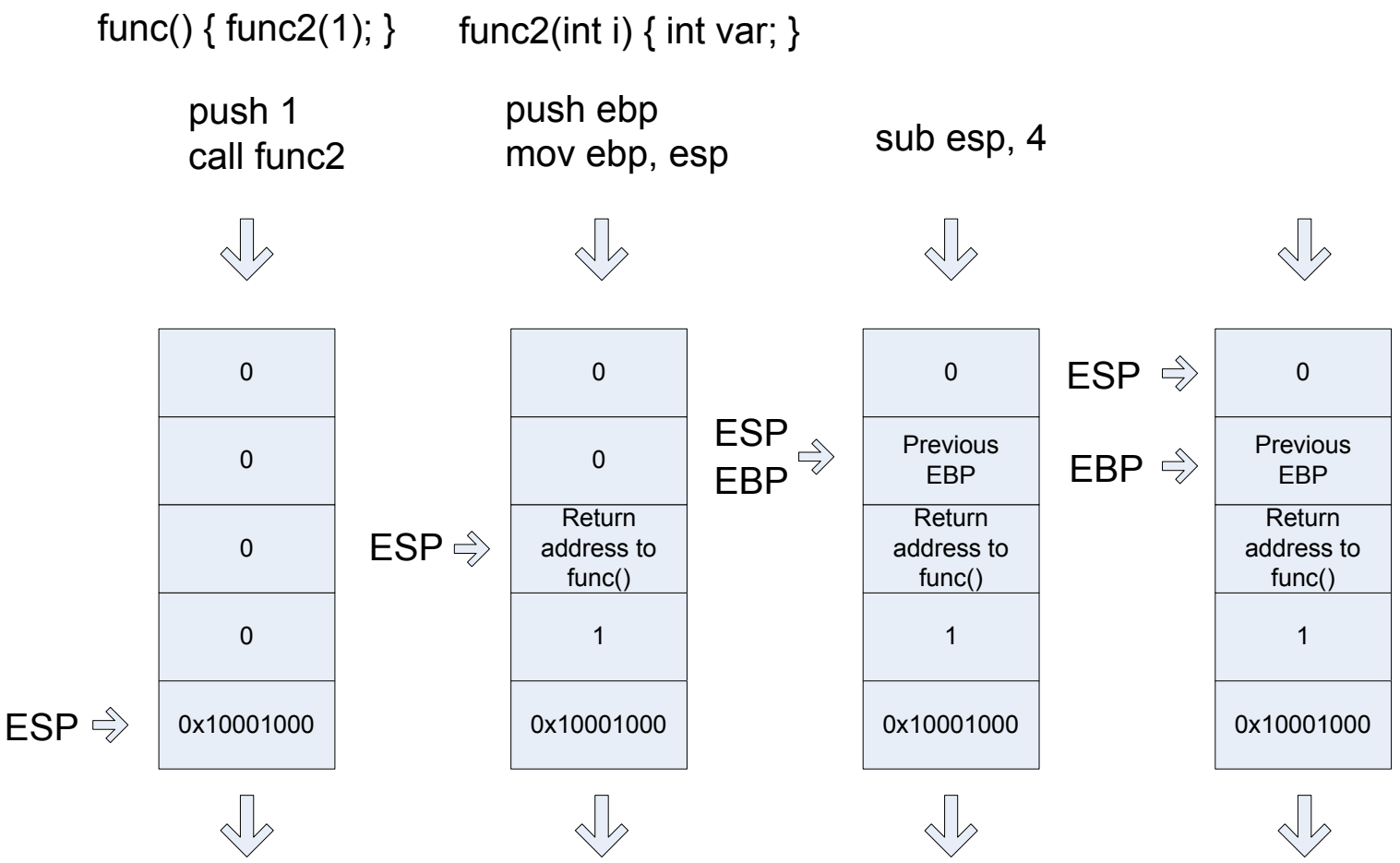
# Stack structure (with FPO)



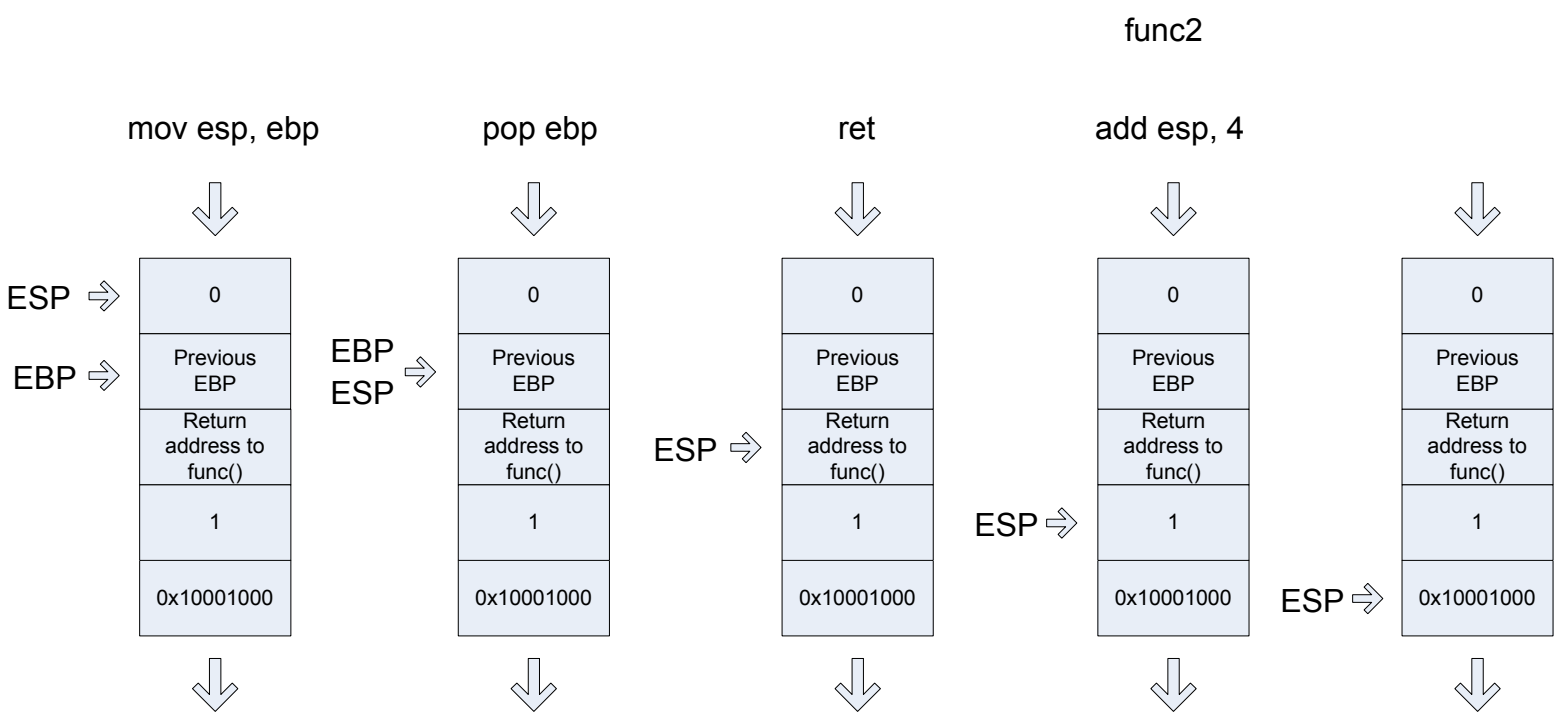

```
[ESP]    Local Var 2 (DWORD)
[ESP+4]  Local Var 1 (DWORD)
[ESP+8]  return address
[ESP+C]  Param 1 (DWORD)
[ESP+10] Param 2 (DWORD)
...
```

```
void func(int Param1, int Param2)
{
   int var1, var2;
}
```

# Function prolog



# Function epilog



```
FunctionParameters!main:
push    ebp                             ; establishing stack frame
mov     ebp,esp                         ;
sub     esp,0xcc                        ; creating stack frame for locals
push    ebx                             ; saving registers that might be used
push    esi                             ;   outside
push    edi                             ;
lea     edi,[ebp-0xcc]                  ; getting lowest address of stack frame
mov     ecx,0x33                        ; filling stack frame with 0xCC
mov     eax,0xcccccccc                  ;
rep     stosd                           ;
push    0x1                             ; push rightmost parameter
push    0x1                             ; push next to right parameter
call    FunctionParameters!ILT+845(?arithmeticYAHHHZ) (00411352)
add     esp,0x8                         ; adjust stack (2 int parameters)
mov     [ebp-0x8],eax                   ; save result to local variable
xor     eax,eax                         ; return value (0)
pop     edi                             ; restoring registers
pop     esi                             ; (in reverse order)
pop     ebx                             ;
add     esp,0xcc                        ; clearing stack
cmp     ebp,esp                         ; ESP == EBP ?
call    FunctionParameters!ILT+935(__RTC_CheckEsp) (004113ac)
mov     esp,ebp                         ; restoring previous stack pointer
pop     ebp                             ; restoring previous stack frame
ret                                     ; return 0
```

```
FunctionParameters!arithmetic:
push    ebp                     ; establishing stack frame
mov     ebp,esp                 ;
sub     esp,0xc0                ; creating stack frame for locals
push    ebx                     ; saving registers that might be used
push    esi                     ;   outside
push    edi                     ;
lea     edi,[ebp-0xc0]          ; getting lowest address of stack frame
mov     ecx,0x30                ; filling stack frame with 0xCC
mov     eax,0xcccccccc          ;
rep     stosd                   ;
mov     eax,[ebp+0xc]           ; eax := [b]
add     eax,[ebp+0x8]           ; eax += [a]
mov     [ebp+0xc],eax           ; [b] := eax      (b = b + a)
mov     eax,[ebp+0x8]           ; eax := [a]
add     eax,0x1                 ; eax := eax + 1
mov     [ebp+0x8],eax           ; [a] := eax      (++a)
mov     eax,[ebp+0x8]           ; eax := [a]
imul    eax,[ebp+0xc]           ; eax := eax * [b]
mov     [ebp+0xc],eax           ; [b] := eax      (b = a * b)
mov     eax,[ebp+0xc]           ; eax := [b]      (return value, b)
pop     edi                     ; restoring registers
pop     esi                     ; (in reverse order)
pop     ebx                     ;
mov     esp,ebp                 ; restoring previous stack pointer
pop     ebp                     ; restoring previous stack frame
Ret                             ; return 0
```

```
0:000> dds esp l200
0012fd28  0012fedc  ; edi
0012fd2c  00000000  ; esi
0012fd30  7ffdf000  ; ebx
0012fd34  cccccccc
0012fd38  cccccccc
0012fd3c  cccccccc
0012fd40  cccccccc
0012fd44  cccccccc
0012fd48  cccccccc
...
...
...
0012fda8  cccccccc
0012fdac  cccccccc
0012fdb0  cccccccc
0012fdb4  cccccccc
0012fdb8  cccccccc
0012fdbc  cccccccc
0012fdc0  cccccccc
0012fdc4  cccccccc
0012fdc8  cccccccc
0012fdcc  cccccccc
0012fdd0  cccccccc
0012fdd4  cccccccc
0012fdd8  cccccccc
0012fddc  cccccccc
0012fde0  cccccccc
0012fde4  cccccccc
0012fde8  cccccccc
0012fdec  cccccccc
0012fdf0  cccccccc
0012fdf4  0012fedc ; prev ebp
0012fdf8  00411a47 FunctionParameters!main+0x27
0012fdfc  00000002
0012fe00  00000004
0012fe04  00000000
...
...
```

# Release (FPO)

```
FunctionParameters!main:
push    0x1
push    0x1
call    FunctionParameters!arithmetic (00401010)
add     esp,0x8
xor     eax,eax
ret

FunctionParameters!arithmetic:
mov     eax,[esp+0x4]       ; eax := [b]
mov     ecx,[esp+0x8]       ; ecx := [a]
add     ecx,eax             ; ecx := ecx + eax
inc     eax                 ; eax := eax + 1
imul    eax,ecx             ; eax := eax * ecx
ret                         ; return result in eax
```

# **cdecl** calling convention (C, C++)

- Parameters are passed from right to left
- Caller is responsible for adjusting the stack after calling the function (callee)
- Allows to call functions with variable number of parameters (printf)

```
printf("result = %d", value);

EBP->    Previous EBP
         Return address
         "" string
         value

printf("left: %d right: %d top: %d bottom: %d", left, right, top, bottom);

EBP->    Previous EBP
         Return address
         "" string
         left
         right
         top
         bottom
```

# Maintenance problem (in old C)

```
// main.c

int main ()
{
    int locVar;

    func (1, 2);
    return 0;
}
```

```
// func.c
// Next release:
//  add third parameter c

int func (int a, int b)
{
    // use parameters
    return 0;
}
```

# “Solution”: introducing new bug

```
// main.c

int main ()
{
    int locVar;

    func (1, 2);
    return 0;
}

; Expects the callee to see:
EBP -> Previous EBP
       Return address
       1
       2
       locVar
```

```
// func.c

int func (int a, int b, int c)
{
    // use parameters
    return 0;
}

; the callee sees:
EBP -> Previous EBP
       Return address
       a
       b
       c
       ...
```

# Function prototypes in standard C and C++

```
// main.c

int func (int a, int b);

int main ()
{
    int locVar;

    func (1, 2);
    return 0;
}
```

```
// func.c
// Next release:
//  add third parameter c

int func (int a, int b);

int func (int a, int b)
{
    // use parameters
    return 0;
}
```

# “Solution”: introducing new bug

```
// main.c

int func (int a, int b);

int main ()
{
    int locVar;

    func (1, 2);
    return 0;
}
```

```
// func.c

int func (int a, int b, int c);

int func (int a, int b, int c)
{
    // use parameters
    return 0;
}
```

# Maintenance in mind

```
// main.c

#include “func.h”

int main ()
{
    int locVar;

    func (1, 2);
    return 0;
}
```

```
// func.h

int func (int a, int b);


// func.c

#include “func.h”

int func (int a, int b)
{
    // use parameters
    return 0;
}
```

# Solution: no more bugs

```
// main.c

#include “func.h”

int main ()
{
    int locVar;

    func (1, 2); // <- ERROR
    return 0;
}
```

```
// func.h

int func (int a, int b, int c);
```

```
// func.c

#include “func.h”

int func (int a, int b, int c)
{
    // use parameters
    return 0;
}
```

# What's next?

- Passing pointers to functions
- What to do if you can't find exact symbol files (discovering function boundaries)
- Real-life stack examples
- Dumps with stack overflow