

Interfacing with high-level languages: **ASM + C** multi-module programming

Calling a procedure (subprogram) defined in C based on calling conventions.

A call convention gives answers to the following questions:

- How do we pass parameters to subroutines?
 - Which types of parameters can we pass? and in what order?
 - How many parameters? Any number of parameters?
- What resources are volatile (may be altered by the **callee = programul apelat**)?
- Where is the result stored?
- What clean-up actions are required after the call?
- Who is responsible to make them?
- Conventions
 - Decided (and documented) by callee, not by caller = programul apelant!
 - Commonly used: **CDECL**, **STDCALL**
 - Less commonly used or obsolete: PASCAL, FORTRAN, SYSCALL, etc.

1. CDECL convention

- Specific to the C programming language
- How do we pass parameters to subroutines? By pushing them on the stack
 - Which types of parameters can we pass? Any type, but extended at least to **DWORD**
 - In what order? From right to the left, that is in the reverse order of declaration
 - How many parameters? Any? Yes, in C is allowed functions with any parameters (ex: printf)
 - What resources are volatile? EAX, ECX, EDX, Eflags (*just flags)
 - **Where is the result stored? EAX, EDX:EAX**
 - What clean-up actions are required? Freeing up the arguments
 - Who is responsible? The caller! (Programul Apelant)

Parameters			Volatile resources	Results	Cleanup
Storage	Order	Number			
Stack	Reverse	<u>Any</u>	EAX, ECX, EDX, Flags*	EAX / EDX:EAX	<u>Caller</u>

2. STDCALL convention

- Specific to Windows operating system
 - Also called **WINAPI**
 - Used by Windows system libraries
- Very similar to the CDECL convention
- Differences:
 - A fixed number of parameters
 - The cleanup is performed by callee (Programul Apelant)

Parameters		Results	Cleanup
------------	--	---------	---------

Storage	Order	Number	Volatile resources		
Stack	Reverse	<u>Fixed</u>	EAX, ECX, EDX, Flags*	EAX / EDX:EAX	<u>Callee</u>

For calling a procedure defined in C it is MANDATORY to follow these stages:

1. **Entry code:** entering the procedure and preparing its execution
2. **Call code:** preparing and performing the call
3. **Return/exit code:** return from the procedure and free resources that are not necessary anymore

The modern compilers generate AUTOMATICALLY the corresponding code for each stage, but when we call a procedure defined in the C language from a procedure defined in assembly, it is MANDATORY that we write the corresponding code for each stage.

1. Entry code

Purpose: entering the procedure and preparing for execution

Tasks:

- **Create a stack frame**
- Reserve on the stack memory required for storing local variables
- Save on the stack a copy of the modified non-volatile resources

A **stack frame** is a data structure stored on the stack and it may contain:

- The parameters pushed by the calling program
- The return address (to the instruction that is after the call instruction)
- Copies of the non-volatile resources used by the procedure
- Local variables

Example:

```
; Entry code:
; ...
; - create the stack frame
push ebp
mov ebp, esp
```

2. Call code

Purpose: prepare and perform the call

Tasks:

- Save volatile resources that are used (push EAX, push ECX, pushad, ...)
- Ensure/enforce constraints (ESP is aligned, DF = 0 etc.)

- Prepare arguments sent to the procedure (place on the stack/in registers based on the call convention)
- Perform the call:
 - **call _procedure** - if the procedure will be linked statically (from C imported)
 - **call [procedure]** - if the procedure will be linked dynamically (from msvcrt.dll imported)

Example:

```
; Calling code
; -----
; Storing the volatile resources being in use;
; push eax, push ecx, pushad, ...

; - compliance with established constraints
; this is not the case here

; - prepare the arguments for the called procedure (pushing them on the stack)
```

3. Exit/return code

Purpose: return from the procedure and free the unnecessary resources.

Tasks:

- Restoring nonvolatile altered resources;
- Removing local variables of the function;
- Destroying the stack frame;
- Returning to the calling code and removing the parameters.

Example:

```
; restoration of the stack frame of the calling program
mov esp, ebp
pop ebp

; ' return from the procedure without freeing the space for the parameters
; (it is the caller's responsibility to do so)
ret
```

Example:

Read 2 numbers a and b (in .c file). Compute the sum of the numbers (in .asm file) and print the result in the .c file.

```

/*++
Se cere un program C care apeleaza functia sumaNumere scrisa in limbaj de asamblare.
Acesta functie primeste
ca parametri doua numere naturale citite in programul C, calculeaza suma lor si
transmite aceasta valoare ca rezultat.
Programul C va afisa suma calculata de functia sumaNumere
--*/

#include <stdio.h>

// functia declarata in fisierul modulSumaNumere.asm
int sumaNumere(int a, int b);

int main()
{
    // declaram variabilele
    int a = 0;
    int b = 0;
    int sum = 0;

    // citim de la tastatura cele doua numere
    printf("a=");
    scanf("%d", &a);

    printf("b=");
    scanf("%d", &b);

    // apelam functia scrisa in limbaj de asamblare
    sum = sumaNumere(a, b);

    // afisam valoarea calculata de functie
    printf("Suma numerelor este %d", sum);
    return 0;
}

```

```

1 bits 32
2
3 ; informam asamblorul ca dorim ca functia _sumaNumere sa fie disponibila altor unitati
4 de compilare
5 global _sumaNumere
6
7 ; linkerul poate folosi segmentul public de date si pentru date din afara
8 segment data public data use32
9
10 ; codul scris in asamblare este dispus intr-un segment public, posibil a fi partajat
11 cu alt cod extern
12 segment code public code use32
13
14 ; int sumaNumere(int, int)
15 ; conventie cdecl
16 _sumaNumere:
17     ; creare cadrul de stiva pentru programul apelat
18     push ebp
19     mov ebp, esp
20
21     ; obtinem argumentele transmise pe stiva functiei sumaNumere
22     ; la locatia [ebp+4] se afla adresa de return (valoarea din EIP la momentul
23     ; apelului)
24     ; la locatia [ebp] se afla valoarea ebp pentru apelant
25     mov eax, [ebp + 8] ; eax <- a
26     mov ebx, [ebp + 12] ; ebx <- b
27     add eax, ebx ; calculam suma
28     ; valoarea de rezultat a functiei este in eax
29     ; refacem cadrul de stiva pentru programul apelant
30     mov esp, ebp
31     pop ebp
32     ret
33 ; conventie cdecl - este responsabilitatea programului apelant sa elibereze
34 parametrii transmissi
35

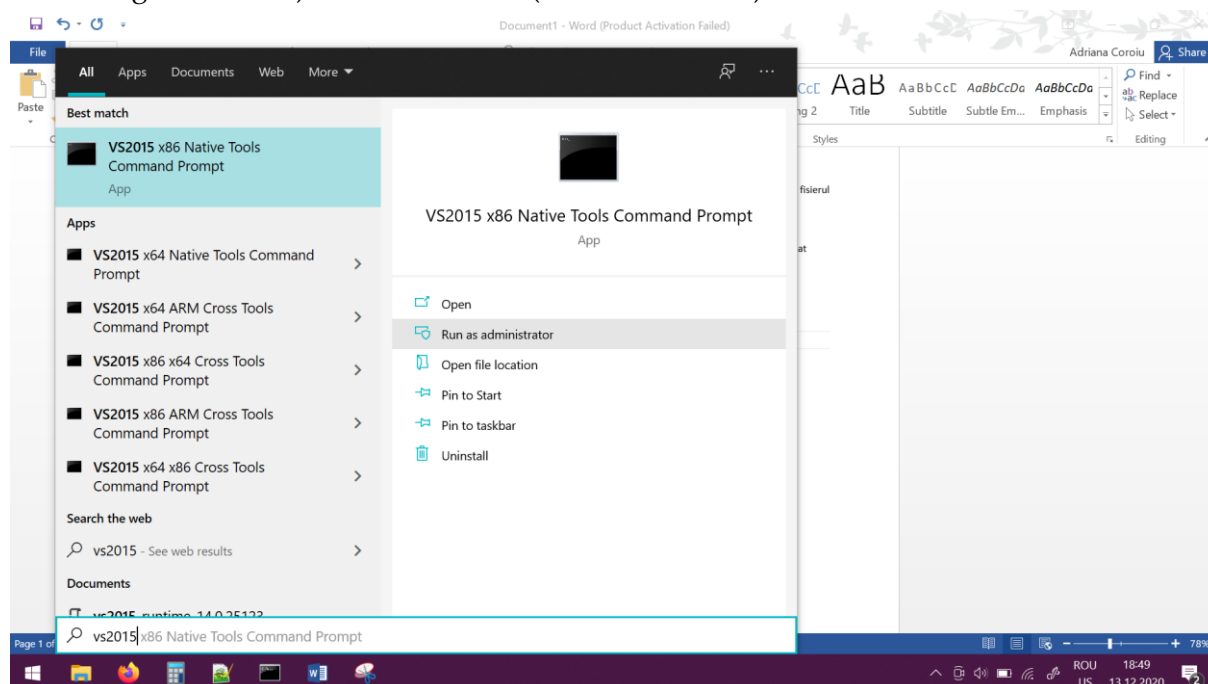
```

In order to execute the examples, we save the both files .asm and .c into the same folder with the nasm.exe:

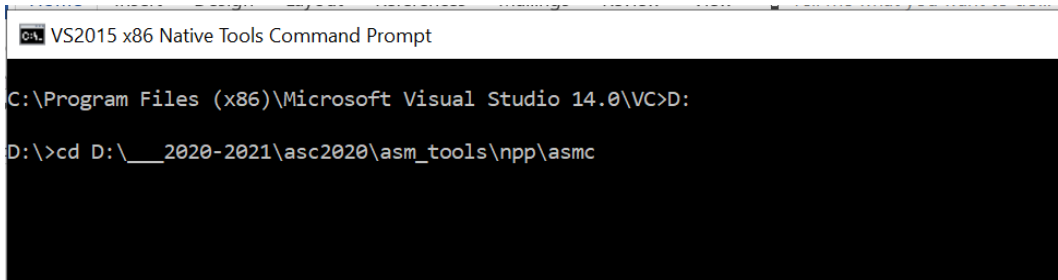
➤ This PC ➤ Local Disk (D:) ➤ __2020-2021 ➤ asc2020 ➤ asm_tools ➤ npp ➤ asmc

Name	Date modified	Type	Size
mainconcatenare	13.12.2020 18:32	C Source	2 KB
mainsuma	13.12.2020 18:40	C Source	1 KB
modulconcatenare	13.12.2020 18:33	Assembler Source	3 KB
modulsuma	13.12.2020 18:40	Assembler Source	2 KB
nasm	02.05.2017 08:19	Application	1.099 KB

Then open the Command Line (for instance cmd from Visual Studio 2015 – or other variants containing C modules): Start -> vs2015 (atentie sa fie x86)



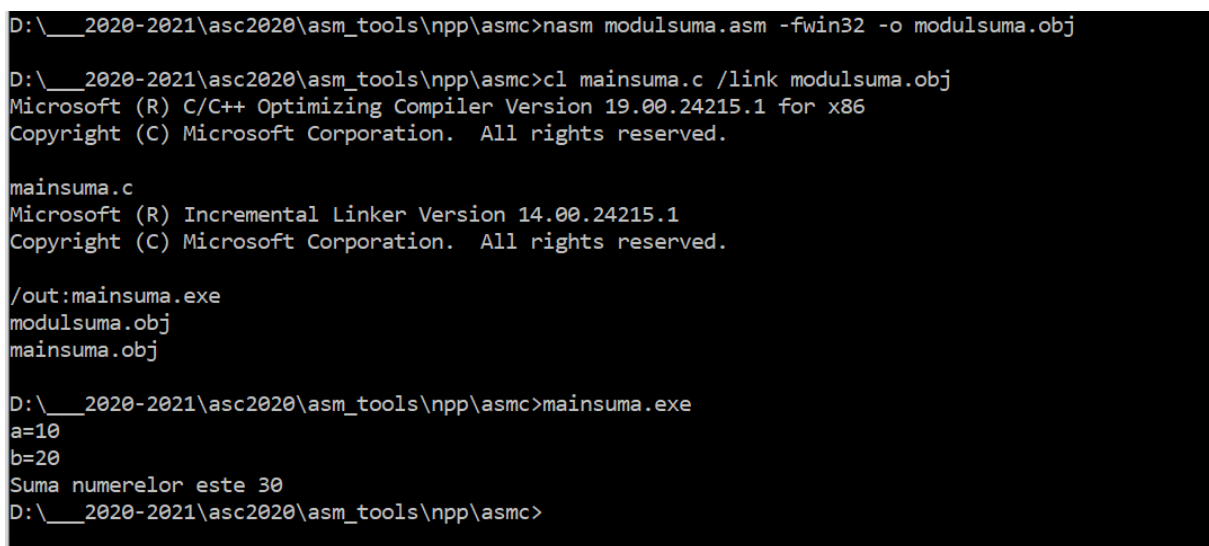
Now, in this cmd, we will open the folder with the codes (we can use command Change Directory CD and the path to the folder):



```
VS2015 x86 Native Tools Command Prompt
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC>D:
D:\>cd D:\__2020-2021\asc2020\asm_tools\npp\asmc
```

Now the commands for converting the asm into obj and to link the modules:

1. `nasm modulsuma.asm -fwin32 -o modulsuma.obj`
2. `cl mainsuma.c /link modulsuma.obj`
3. `mainsuma.exe`



```
D:\__2020-2021\asc2020\asm_tools\npp\asmc>nasm modulsuma.asm -fwin32 -o modulsuma.obj
D:\__2020-2021\asc2020\asm_tools\npp\asmc>cl mainsuma.c /link modulsuma.obj
Microsoft (R) C/C++ Optimizing Compiler Version 19.00.24215.1 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

mainsuma.c
Microsoft (R) Incremental Linker Version 14.00.24215.1
Copyright (C) Microsoft Corporation. All rights reserved.

/out:mainsuma.exe
modulsuma.obj
mainsuma.obj

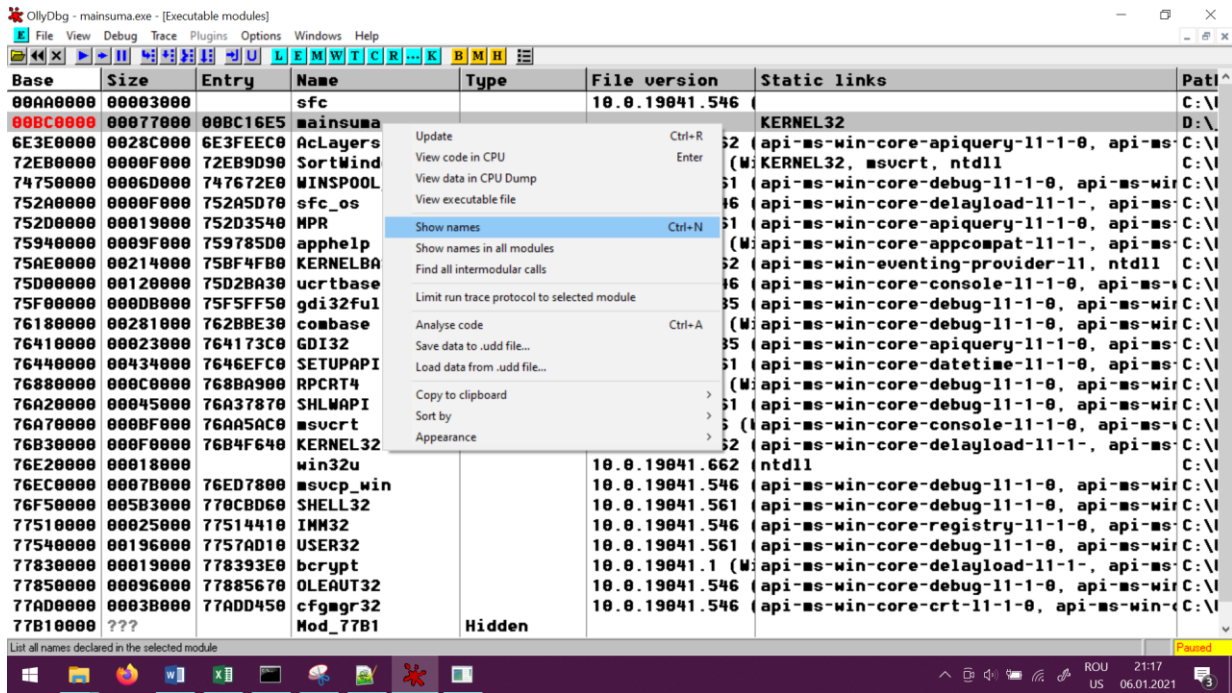
D:\__2020-2021\asc2020\asm_tools\npp\asmc>mainsuma.exe
a=10
b=20
Suma numerelor este 30
D:\__2020-2021\asc2020\asm_tools\npp\asmc>
```

If we want to check the code into the debugger, the commands are:

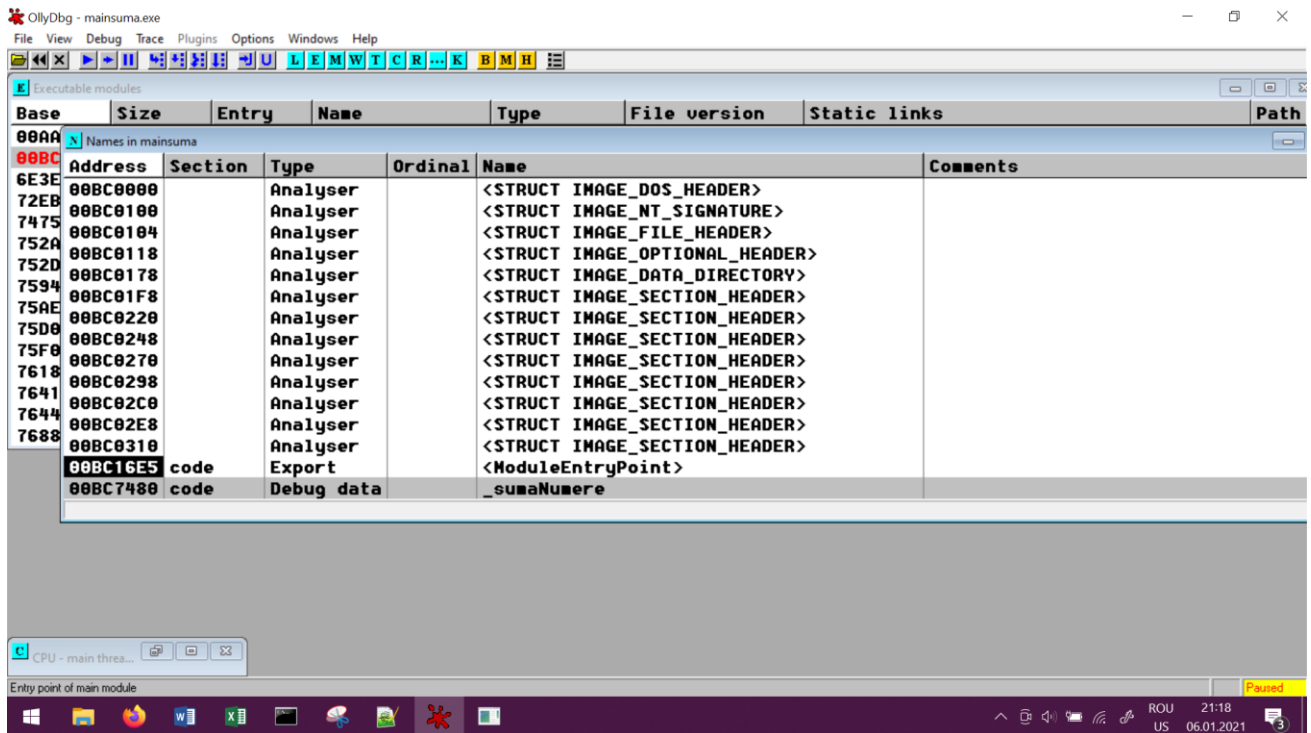
- `nasm modulsuma.asm -fwin32 -g -o modulsuma.obj`
- `cl /Z7 mainsuma.c /link modulsuma.obj`

Then we open the Olydbg from file

File -> Open



Then double click on the name of the function in asm:



Now we can check the code step-by-step:

OllyDbg - mainsuma.exe - [CPU - main thread, module mainsuma]

File View Debug Trace Plugins Options Windows Help

00BC747C CC INT3
00BC747D CC INT3
00BC747E CC INT3
00BC747F CC INT3
00BC7480 55 PUSH EBP mainsuma._sumaNu
00BC7481 89E5 MOV EBP,ESP
00BC7483 8B45 08 MOV EAX,DWORD PTR SS:[ARG.1]
00BC7486 8B5D 0C MOV EBX,DWORD PTR SS:[ARG.2]
00BC7489 01D8 ADD EAX,EBX
00BC748B 89EC MOV ESP,EBP
00BC748D 5D POP EBP
00BC748E C3 RETN
00BC748F CC INT3
00BC7490 CC INT3

Registers (3DNow!)
EAX 0099F98C
ECX 00BC16E5 mainsuma.<ModuleE
EDX 00BC16E5 mainsuma.<ModuleE
EBX 00E7B000
ESP 0099F934
EBP 0099F940
ESI 00BC16E5 mainsuma.<ModuleE
EDI 00BC16E5 mainsuma.<ModuleE
EIP 00BC16E5 mainsuma.<ModuleE
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 1 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit E7E000(FFF)

Stack [0099F930]=0 (current registers)
EBP=0099F940 (current registers)
Local call from main+5F

Address Hex dump AS 0099F934 76B4FA29) u
00C2E000 61 3D 00 00 25 64 00 00 62 3D 00 00 25 64 00 00 0099F938 00E7B000 aSa
00C2E010 53 75 6D 61 20 6E 75 6D 65 72 65 6C 6F 72 20 65 0099F93C 76B4FA10 u
00C2E020 73 74 65 20 25 64 00 00 00 00 00 00 00 00 00 00 0099F940 0099F99C aa0a
00C2E030 FF FF FF FF 01 00 00 00 00 00 00 00 00 00 00 00 00 00 0099F944 77B875F4 uSw
00C2E040 01 00 00 00 B1 19 BF 44 14 CA 09 E4 00 00 00 00 00 00 0099F948 00E7B000 aSa
00C2E050 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0099F94C 0C166AC8 ajQ
00C2E060 20 05 93 19 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0099F950 00000000 aaaa
00C2E070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0099F954 00000000 aaaa
00C2E080 00 00 00 00 01 20 00 00 00 00 00 00 00 00 00 00 00 00 0099F958 00E7B000 aSa
00C2E090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0099F95C 00000000 aaaa

Entry point of main module

Paused

21:20
US 06.01.2021