

REVERSING WITH IDA PRO FROM SCRATCH

PART 2

As this is a course from scratch, there are things that many people know. If you are one of them, skip them, but most of the readers won't do it. That's why I added that info.

NUMERICAL SYSTEM

The three most used numerical systems are: binary, decimal and hexadecimal.

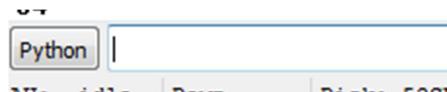
The basic concept of each one is the following:

BINARY: numbers are represented with two characters. 1 and 0 that's why it is called like that.

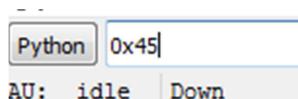
DECIMAL: all numbers are represented with 10 characters. (From 0 to 9)

HEXADECIMAL: all numbers are represented with characters from 0 to F (from 0 to 9, plus A, B, C, D, E y F. (16 in total)

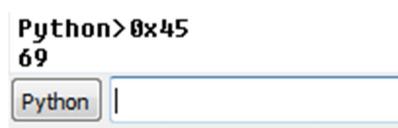
In IDA, in the bottom part, there is a bar to execute PYTHON commands. This will help us shift from one to the other easily.



If I type, for example, 0x45, it interprets it as hexadecimal for the **0x** at the beginning. We can convert from hexadecimal to decimal just pressing ENTER.



After pressing ENTER:



The result is 69 that is 0x45 hex converted to decimal.

If we want to convert the number from decimal to hexadecimal, we use the **hex()** function:

```
Python>hex(69)
0x45
Python hex(69)|
```

To convert it into binary, we use bin()

```
Python>bin(69)
0b1000101
Python |
```

```
Python>bin(0x45)
0b1000101
Python |
```

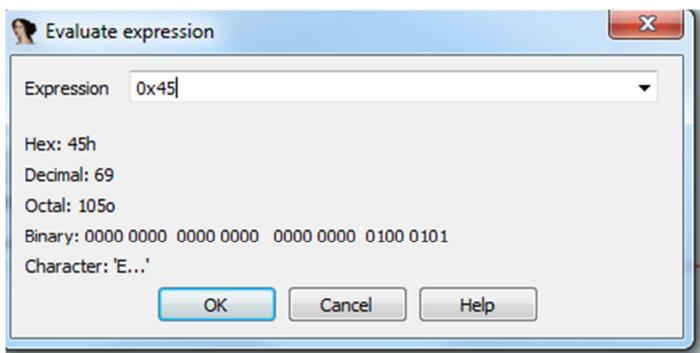
The result is 1000101. **0b**, at the beginning, means binary. We can convert it from binary to decimal or hexadecimal.

```
Python>0b1000101
69
Python 0b1000101|
```

```
Python>hex(0b11)
0x3
Python |
```

Any number typed directly after pressing ENTER will be shown in decimal. To convert it into hexadecimal or binary, we should use the Python functions hex() or bin().

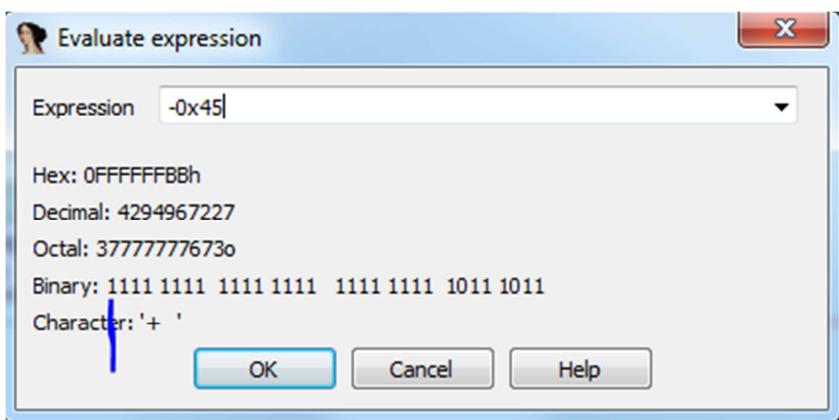
To make conversions, IDA has an integrated calculator in VIEW-CALCULATOR in which we can see any number converted into all numerical systems at the same time. It also shows the character of the value. In this case, 0x45 corresponds to E.



NEGATIVE NUMBERS IN HEXADECIMAL

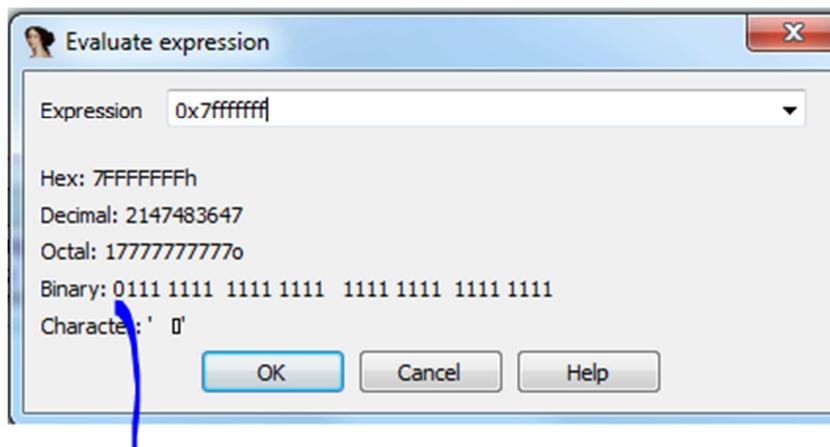
In almost every moment, we will work in hex, but the question is how is a negative hex number represented in 32 bits?

In a 32-bit binary number we use the first bit to mean if it is 0 (positive) or 1 (negative)



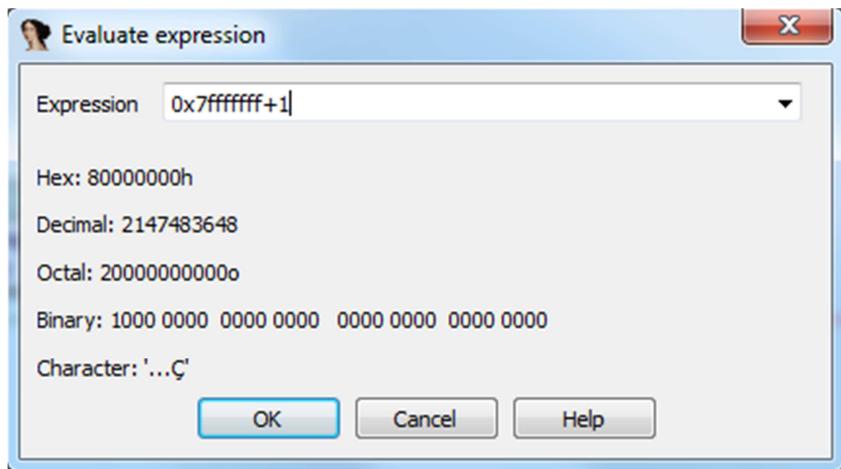
We see that a value, e.g, -0x45 in hex can be represented as 0xFFFFFBB and that its first byte is 1 in binary.

In this way, the minimum positive value is 0 obviously, but what should be the greatest positive value we can represent?



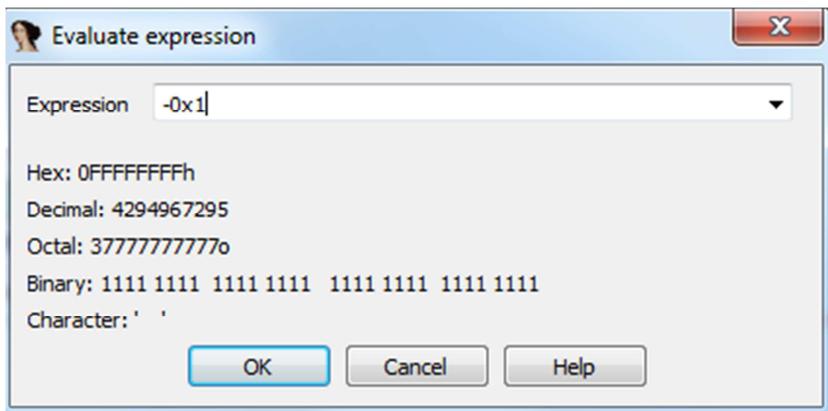
0x7FFFFFFF is the greatest positive value if we consider the sign, then after adding 1, all of the other bits are 1. The greatest value must change to 1 to continue.

If we add it 1.



The first bit changes to 1 and the rest is 0.

This evaluator considers all numbers as positive unless we add “-“ at the beginning. For example:



The minimum negative value -1 corresponds to 0xFFFFFFFF and the greatest negative value will be 0x80000000.

If we don't consider the sign in an operation, then all the values will be positive from 0 to 0xFFFFFFFF.

But considering the sign we will have the positive values from 0x0 to 0x7FFFFFFF and the negative values from 0xFFFFFFFF to 0x80000000.

POSITIVE VALUES

00000000 equals 0 decimal

00000001 equals 1 decimal

.....

.....

7FFFFFFF equals 2147483647 decimal (the greatest positive value)

NEGATIVE VALUES

FFFFFFF equals -1 decimal

FFFFFFE equals -2 decimal

.....

.....

80000000 equals -2147483648 decimal (the greatest negative value)

Hexadecimal	Decimal sin signo	Decimal con signo
0x7FFFFFFF	2.147.483.647	2.147.483.647
0x00000001	1	1
0x00000000	0	0
0xFFFFFFF	4.294.967.295	-1
0x80000000	2.147.483.648	-2.147.483.648

ASCII CHARACTERS

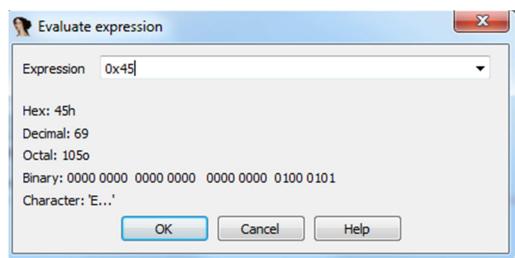
One of the topics we must know too is the way our system writes data on screen. It assigns each character a hex value to interpret them as letters, numbers, symbols, etc.

In the first row, we see the decimal value, in the second one is the hex value and in third one is the character, for example, if I want to write a space, I have to use 0x20 or 32 decimal. Any character we need being a letter or number we can see it in the next chart.

Dec.	Hex.	Cara	Dec.	Hex.	Cara	Dec.	Hex.	Cara
32	20	esp	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	\$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(72	48	H	104	68	h
41	29)	73	49	I	105	69	i
42	2A	*	74	4A	.I	106	6A	i
43	2B	+	75	4B	K	107	6B	k
44	2C		76	4C	I	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2F		78	4F	N	110	6F	n
47	2F	/	79	4F	Ó	111	6F	ó
48	30	0	80	50	P	112	70	ó
49	31	1	81	51	Ó	113	71	á
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u

54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	v
58	3A	:	90	5A	Z	122	7A	z
59	3B	:	91	5B	[123	7B	{
60	3C	<	92	5C	\	124	7C	
61	3D	=	93	5D	1	125	7D	}
62	3E	>	94	5F	^	126	7F	~
63	3F	?	95	5F		127	7F	

IDA calculator evaluates expressions and shows the correspondent characters.
0x45 is the character **E**.



33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	\$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(72	48	H	104	68	h
41	29)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B		75	4B	K	107	6B	k

Another way to see this is using the Python function `chr()`

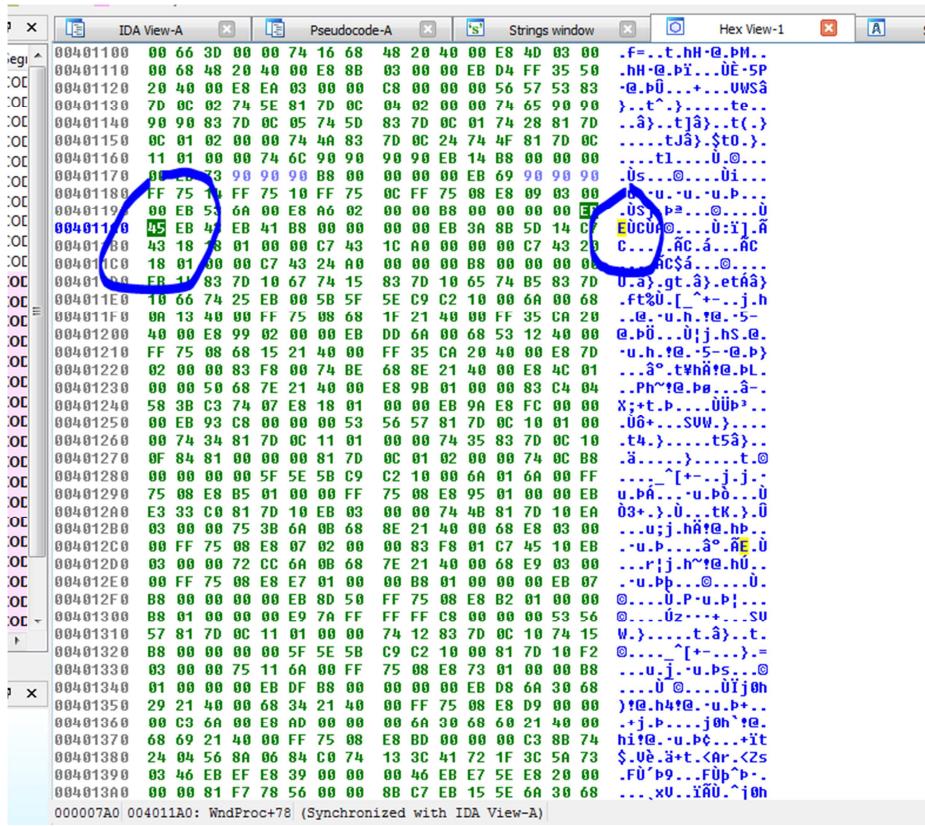
```
Python>chr(0x45)
```

```
E
```

```
Python
```

```
|
```

In the HEX DUMP Windows there is a column that also shows the characters.



```

00401100 00 66 3D 00 00 74 16 68 48 20 40 00 E8 4D 03 00 .F=..t.hN@.pM..
00401110 00 68 48 20 40 00 E8 8B 03 00 00 EB D4 FF 35 50 .hN@.pI...ÜE-5P
00401120 20 40 00 E8 EA 03 00 C8 00 00 56 57 53 83 @.pÜ...UWSä
00401130 7D 0C 02 74 5E 81 7D 0C 04 02 00 00 74 65 90 90 }..t^}....te...
00401140 90 90 83 7D 0C 05 74 5D 83 7D 0C 01 74 28 81 7D ..â}..t]â}..t{()
00401150 0C 01 02 00 00 74 4A 83 7D 0C 24 74 4F 81 7D 0C ....tJâ}.$t{().}
00401160 11 01 00 00 74 6C 90 90 90 90 EB 14 B8 00 00 00 ....t1....U.Ø...
00401170 01 6B 3 90 90 90 B8 00 00 00 EB 69 90 90 90 Üs...Ø...Üi...
00401180 FF 75 10 FF 75 10 FF 75 0C FF 75 08 E8 09 03 00 .ü...u.u.u.u.b...
00401190 00 EB 51 6A 00 E8 46 02 00 00 00 00 EB 3A 8B 5D 14 C4 .Üs...b^...Ø...ü...
004011A0 45 EB 41 B8 00 00 00 00 EB 3A 8B 5D 14 C4 EUÜ...Üi;]ä
004011B0 43 18 18 01 00 00 C7 43 1C A0 00 00 00 C7 43 21 C....üC.ä...üC
004011C0 18 01 00 00 C7 43 24 A0 00 00 00 B8 00 00 00 ..üC.ä...Ø...
004011D0 FR 11 83 7D 10 67 74 15 83 7D 10 65 74 B5 83 7D 0.ä).gt.ä).etÄä)
004011E0 10 66 74 25 EB 00 5B 5F 5E C9 C2 10 00 6A 00 68 .ft%Ü.Ü+--.j.h
004011F0 0A 13 40 00 FF 75 08 68 1F 21 40 00 FF 35 CA 20 ..ü...u.h.Ø@.5-
00401200 40 00 E8 99 02 00 00 EB DD 6A 00 68 53 12 40 00 @.pÜ...Üij.hS.Ø.
00401210 FF 75 08 68 15 21 40 00 FF 35 CA 20 40 00 E8 7D .u.h.Ø@.5-@.p>
00401220 02 00 00 83 F8 00 74 BE 68 8E 21 40 00 EB 4C 01 ...â}..tVññ@.Øl.
00401230 00 00 50 68 7E 21 40 00 E8 9B 01 00 00 83 C4 04 ..Ph^@.Øb...â...
00401240 58 3B C3 74 07 E8 18 01 00 00 EB 9A E8 FC 00 00 X;t.p...ÜÜb...
00401250 00 EB 93 C8 00 00 00 53 56 57 81 7D 0C 10 01 00 ..ü+...SUW.Ø...
00401260 00 74 34 81 7D 0C 11 01 00 00 74 35 83 7D 0C 10 .tñ.}....t5a}...
00401270 0F 84 81 00 00 00 81 7D 0C 01 02 00 00 74 0C B8 ..ä...}....tØ
00401280 00 00 00 00 5F 5E 5B C9 C2 10 00 6A 01 6A 00 FF ...^+[+--.j.j.-
00401290 75 08 E8 B5 01 00 00 FF 75 08 E8 95 01 00 00 EB u.ä...ü.pÜ...ü...
004012A0 E3 33 C0 81 7D 10 00 EB 03 00 00 74 81 7D 10 EA 03+}.Ü...tK.}...ü...
004012B0 03 00 00 75 38 6A 00 68 8E 21 40 00 68 E8 03 00 ...u;j.hä@.hp
004012C0 00 FF 75 08 E8 07 02 00 00 83 F8 01 C7 45 10 EB ..ü.p...ü...ü...
004012D0 03 00 00 72 CC 6A 0B 68 7E 21 40 00 68 E9 03 00 ...r;j.h^@.hÜ...
004012E0 00 FF 75 08 E8 E7 01 00 00 88 01 00 00 00 EB 07 ..ü.pÜ...Ø...
004012F0 B8 00 00 00 00 EB 8D 50 FF 75 08 E8 B2 01 00 00 ...ü.P+ü.P...
00401300 B8 01 00 00 00 E9 74 FF FF FF C8 00 00 00 53 56 ...ü...üz...+...ü...
00401310 57 81 7D 0C 11 01 00 00 74 12 83 7D 0C 10 74 15 W.}....t.ä}.t.
00401320 B8 00 00 00 00 5F 5E 5B C9 C2 10 00 81 7D 10 F2 ...ü...ü...ü...
00401330 03 00 00 75 11 6A 00 FF 75 08 E8 73 01 00 00 B8 ...u.j...ü.pÜ...ü...
00401340 01 00 00 00 EB DF B8 00 00 00 EB D8 6A 30 68 ...ü...ü...ü...
00401350 29 21 40 00 68 34 21 40 00 FF 75 08 E8 D9 00 00 )Ü.hä@.ü.pÜ...
00401360 00 C3 6A 00 E8 AD 00 00 00 6A 30 68 60 21 40 00 ..ü.pÜ...ü...
00401370 68 69 21 40 00 FF 75 08 E8 BD 00 00 00 C3 88 74 hit@.ü.pÜ...+Üt
00401380 24 04 56 8A 06 84 C0 74 13 3C 41 72 1F 3C 5A 73 $.ü.ü.t.<Ar.<Zs
00401390 03 46 EB EF E8 39 00 00 00 46 EB E7 5E E8 20 00 .FÜ.pÜ...FÜ.pÜ...
004013A0 00 00 81 F7 78 56 00 00 88 C7 E5 15 5E 6A 30 68 ...xÜ.iÜ.ü...
000007A0 004011A0: WndProc+78 (Synchronized with IDA View-A)

```

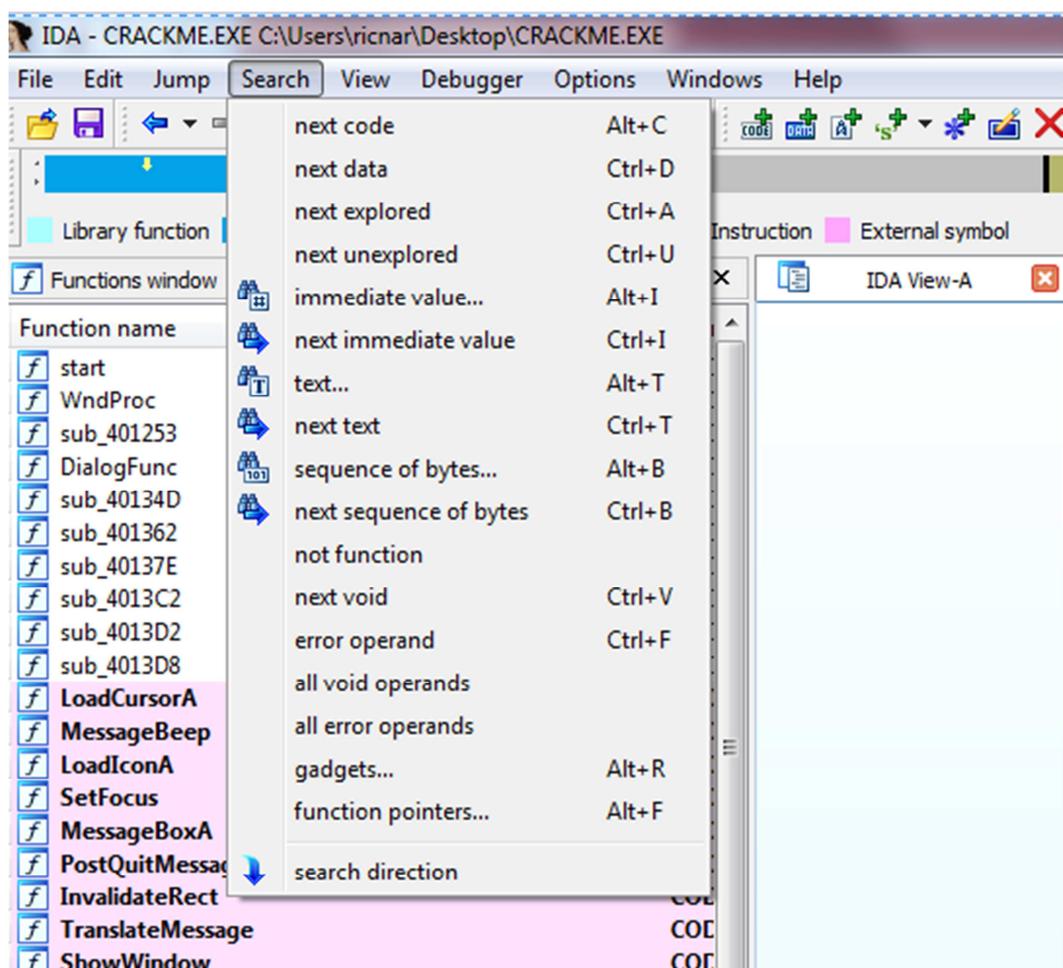
There, we see the **45** in the right column represented by **E**.

SEARCHING POSSIBILITIES

We see the option **SEARCH** in the menu and if we are in the disassembly or **IDA-VIEW** tab we have many searching options that are easy to interpret.

If we shift to the disassembly window and the **SEARCH** submenus are not present, we should click on any instruction to change the focus and make it appear.

Some options in the next image correspond to plugins added to my IDA. So you won't see them in the IDA you downloaded.



NEXT CODE

This will search for the next instruction interpreted as CODE. If there is a part that is not detected as code, it will skip it.

Search completed. Found at 004011A1.
Search completed. Found at 004011A3.
Search completed. Found at 004011A5.
Search completed. Found at 004011AA.
Search completed. Found at 004011AC.
Search completed. Found at 004011AF.
Search completed. Found at 004011B6.

NEXT DATA

This will search for the next instruction interpreted as DATA o data handling in any section.

```

CODE:0040156C          jmp      ds:PrintDlgA
CODE:0040156C ; -----
CODE:00401572          align 100h
CODE:00401600          dd 280h dup(?)
CODE:00401600 CODE      ends
CODE:00401600
DATA:00402000 ; Section 2. (virtual address 00002000)
DATA:00402000 ; Virtual size           : 00001000 ( 4096.)
DATA:00402000 ; Section size in file : 00000200 ( 512.)
DATA:00402000 ; Offset to raw data for section: 00000C00
DATA:00402000 ; Flags C0000040: Data Readable Writable
DATA:00402000 ; Alignment      : default
DATA:00402000 ; =====
DATA:00402000
DATA:00402000 ; Segment type: Pure data
DATA:00402000 ; Segment permissions: Read/Write
DATA:00402000 DATA      segment para public 'DATA' use32
DATA:00402000         assume cs:DATA
DATA:00402000         ;org 402000h
DATA:00402000         db 0
DATA:00402001         db 0
DATA:00402002         db 0
DATA:00402003         db 0
DATA:00402004 ; HWND hWnd
DATA:00402004 hWnd     dd 0          ; DATA XREF: start+C8↑w
DATA:00402004           ; start+CF↑r ...
DATA:00402005         db 0
DATA:00402006         db 0
DATA:00402007         db 0
DATA:00402008         db 0

```

In this case, it detected a dword (dd) in that address that doesn't correspond to any instruction, obviously, if we continue searching, it will search for the next **data**. Here, we see the data section below if I search again.

```

DATA:00402002         db 0
DATA:00402003         db 0
DATA:00402004 ; HWND hWnd
DATA:00402004 hWnd     dd 0          ; DATA XREF: start+C8↑w
DATA:00402004           ; start+CF↑r ...
DATA:00402005         db 0
DATA:00402006         db 0

```

I see that it stops in an address where there is info on the right side meaning that it is a place where it will work with data.

It continues skipping addresses with just zeros and without any reference. It shows just the places with possible data the program uses.

Search completed. Found at 00402004.

Search completed. Found at 00402048.

```

DATA:00402040          uu      u
DATA:00402047          db      0
● DATA:00402048 ; MSG Msg      MSG <0>          ; DATA XREF: start+F7↑o
DATA:00402048          Msg      MSG <0>          ; start+107↑o ...
● DATA:00402064 ; WNDCLASSA WndClass      WNDCLASSA <0>          ; DATA XREF: start:loc_40101D↑w
DATA:00402064          WndClass      WNDCLASSA <0>          ; start+8B↑o ...
● DATA:0040208C          db      0
● DATA:0040208D          db      0
● DATA:0040208E          "      "

```

It will skip everything not detected as data used by the program and it will search for the next one.

SEARCH EXPLORED AND UNEXPLORED

The first one will skip for code or data detected and the second one for zones not detected as valid code or data.

```

DATA:00402000 ; Segment type: Pure data
DATA:00402000 ; Segment permissions: Read/Write
DATA:00402000 DATA          segment para public 'DATA' use32
DATA:00402000          assume cs:DATA
DATA:00402000          ;org 402000h
● DATA:00402000          db      0
● DATA:00402001          db      0
● DATA:00402002          db      0
● DATA:00402003          db      0
● DATA:00402004 ; HWND hWnd      dd      0          ; DATA XREF: st...
DATA:00402004 hWnd          dd      0          ; start+CF↑r ...
● DATA:00402008          db      0

```

The zone with 0's in 0x402000 is found with SEARCH UNEXPLORED.

Search completed. Found at 00402000.

Search completed. Found at 00402000.

Search completed. Found at 00402001.

Search completed. Found at 00402001.

Search completed. Found at 00402002.

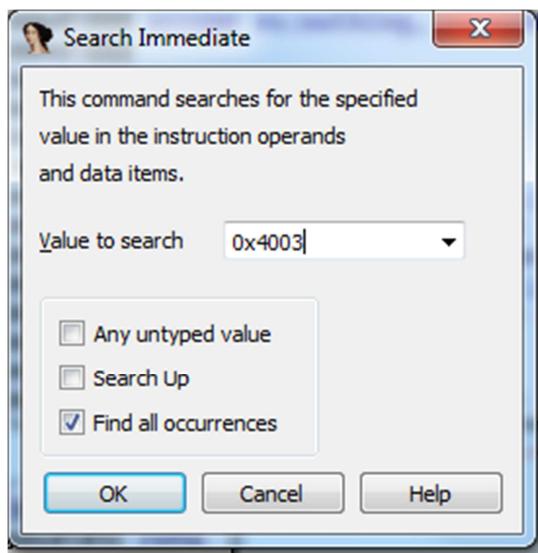
Search completed. Found at 00402003.

Search completed. Found at 00402008

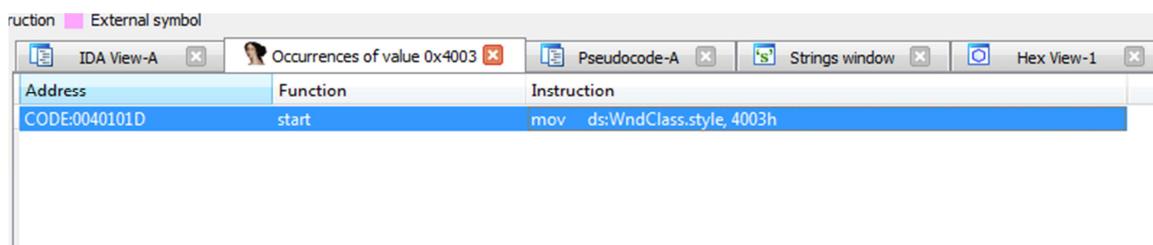
Repeating this search it skips the data in 0x402004 because it detected as EXPLORED.

SEARCH INMEDIATE VALUE - SEARCH NEXT INMEDIATE VALUE

It will search for the constant we write between the instructions and data.

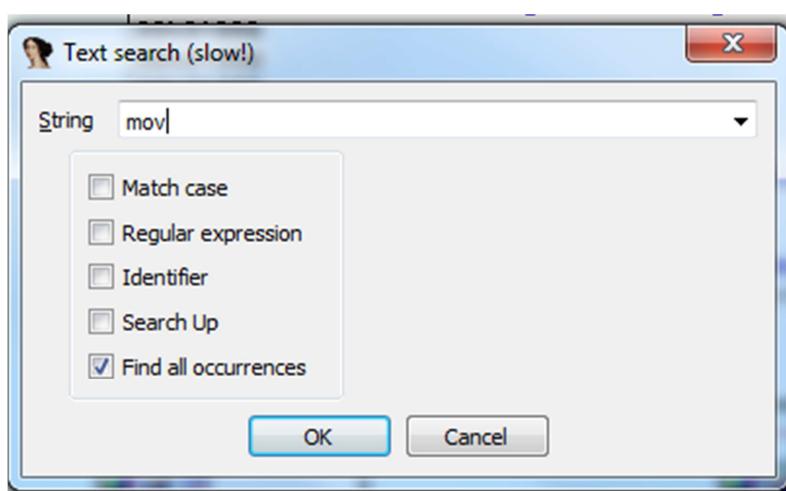


If we check FIND ALL OCURRENCES, it will search for all and if we don't, it will search for one by one. In that case, we should use SEARCH NEXT INMEDIATE VALUE to repeat the search.



SEARCH TEXT –SEARCH NEXT TEXT

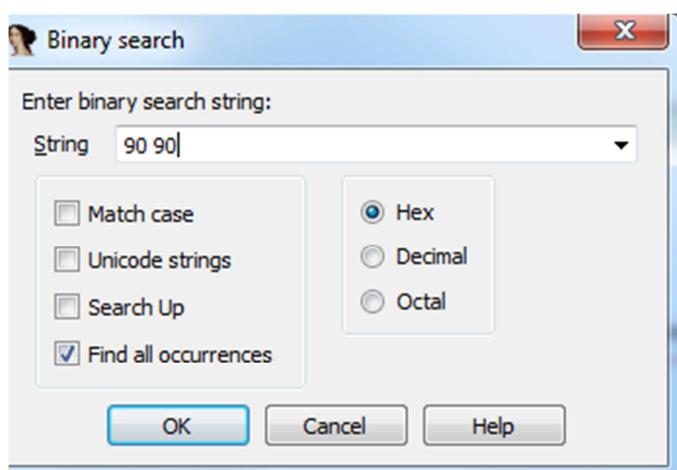
It will search for the text we type or also regular expressions.



Address	Function	Instruction
CODE:00401007	start	mov ds: hInstance, eax
CODE:0040101D	start	mov ds: WndClass.style, 4003h
CODE:00401027	start	mov ds: WndClass.lpfnWndProc, offset WndProc
CODE:00401031	start	mov ds: WndClass.cbClsExtra, 0
CODE:0040103B	start	mov ds: WndClass.cbWndExtra, 0
CODE:00401045	start	mov eax, ds: hInstance
CODE:0040104A	start	mov ds: WndClass.hInstance, eax
CODE:00401057	start	mov ds: WndClass.hIcon, eax
CODE:00401068	start	mov ds: WndClass.hCursor, eax
CODE:0040106D	start	mov ds: WndClass.hbrBackground, 5
CODE:00401077	start	mov ds: WndClass.lpszMenuName, offset aMenu ; "MENU"
CODE:00401081	start	mov ds: WndClass.lpszClassName, offset ClassName ; "No need to disasm..."
CODE:004010C8	start	mov ds: hWnd, eax
CODE:0040116C	WndProc	mov eax, 0
CODE:00401176	WndProc	mov eax, 0
CODE:0040119A	WndProc	mov eax, 0
CODE:004011A5	WndProc	mov eax, 0
CODE:004011AC	WndProc	mov ebx, [ebp+IParam]
CODE:004011AF	WndProc	mov dword ptr [ebx+18h], 118h
CODE:004011B6	WndProc	mov dword ptr [ebx+1Ch], 0A0h
CODE:004011BD	WndProc	mov dword ptr [ebx+20h], 118h
CODE:004011C4	WndProc	mov dword ptr [ebx+24h], 0A0h
CODE:004011CB	WndProc	mov eax, 0
CODE:0040127F	sub_401253	mov eax, 0
CODE:004012CC	sub_401253	mov [ebp+arg_8], 3EBh
CODE:004012E9	sub_401253	mov eax, 1
CODE:004012F0	sub_401253	mov eax, 0
CODE:00401300	sub_401253	mov eax, 1
CODE:00401320	DialogFunc	mov eax, 0
CODE:0040133F	DialogFunc	mov eax, 1
CODE:00401346	DialogFunc	mov eax, 0
CODE:0040137E	sub_40137E	mov esi, [esp+arg_0]
CODE:00401383	sub_40137E	mov al, [esi]
CODE:004013A8	sub_40137E	mov eax, edi
CODE:004013C6	sub_4013C2	mov bl, [esi]
CODE:004013D4	sub_4013D2	mov [esi], al
CODE:004013DE	sub_4013D8	mov esi, [esp+arg_0]

If we check search just one, we will need SEARCH NEXT TEXT to continue searching.

SEARCH SEQUENCE OF BYTES

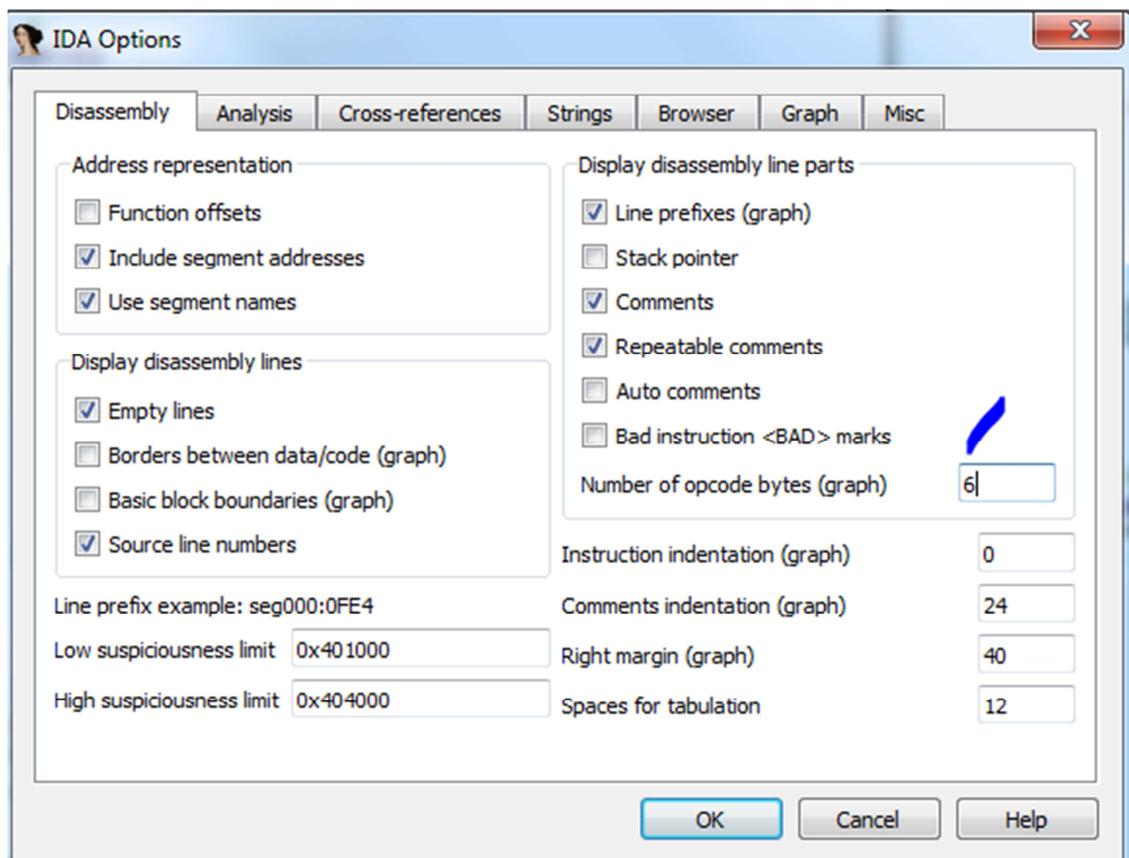


It will search for the sequence of hex bytes we type in the executable.

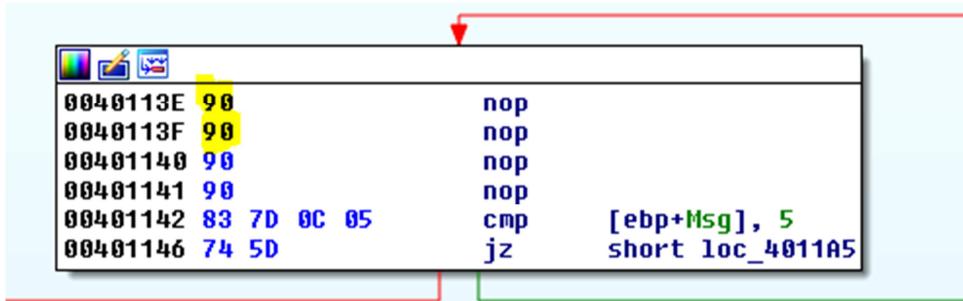
Address	Function	Instruction
CODE:0040113E	WndProc	nop
CODE:0040113F	WndProc	nop
CODE:00401140	WndProc	nop
CODE:00401166	WndProc	nop
CODE:00401167	WndProc	nop
CODE:00401168	WndProc	nop
CODE:00401173	WndProc	db 3 dup(90h)
CODE:0040117D	WndProc	align 10h

If we double click on the first one, for example.

And in IDA options we type 6 to show just 6 bytes maximum for each instruction.



It found the 90 90 we asked it.



SEARCH NOT FUNCTION

Search for the next address where it finds something detected as an incomplete function.

Search completed. Found at 004013D7.

```
CODE:004013D6 sub_4013D2      endp
CODE:004013D6
CODE:004013D7 ; -----
● CODE:004013D7             retn
CODE:004013D8
CODE:004013D8 ; ===== S U B R O U T I N E =====
CODE:004013D8
CODE:004013D8
CODE:004013D8 sub_4013D8      proc near             ; CODE XREF: WndProc+110↑p
CODE:004013D8 arq 0          = dword ptr 4
```

There is a RET alone that is not detected as a function. Sometimes, there are functions that IDA could not detect as functions but they have valid code.

Those are the most important search functions in the IDA menu. Of course, having the opportunity of using Python scripts, we could create better search possibilities with some code lines.

Every search we do is logged in a tab that is always there until we close it.

We go step by step to understand it better.

Until part 3

Ricardo Narvaja

Translated by: @IvinsonCLS