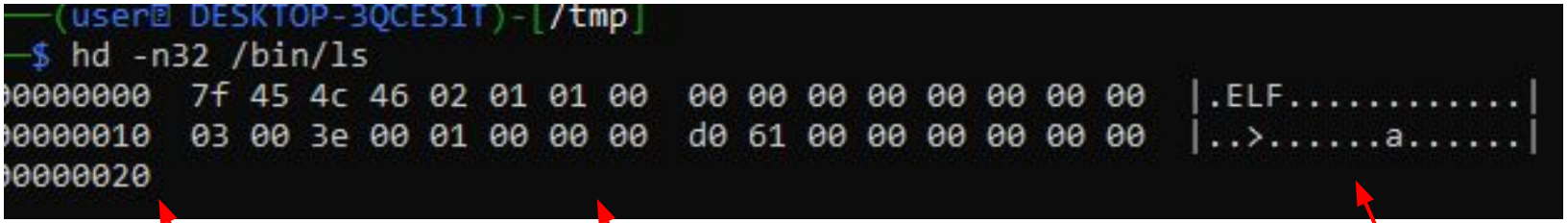
A Arte da Engenharia Reversa

Arquivos

O que é um arquivo?

- Zero ou mais números armazenados em algum lugar.
- Pode-se dizer que contém dados.
- Além disso, há os metadados (mais sobre eles em breve).

Inspecionando o conteúdo de um arquivo



Inspecionando o conteúdo de um arquivo

```
$ hexyl -n32 maf15.png
┌────────┬─────────────────────────┬─────────────────────────┬────────┬────────┐
│00000000│ 89 50 4e 47 0d 0a 1a 0a │ 00 00 00 0d 49 48 44 52 │×PNG__•_│000_IHDR│
│00000010│ 00 00 01 90 00 00 01 1e │ 08 02 00 00 00 2b 67 d4 │00•×00••│••000+g×│
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
```

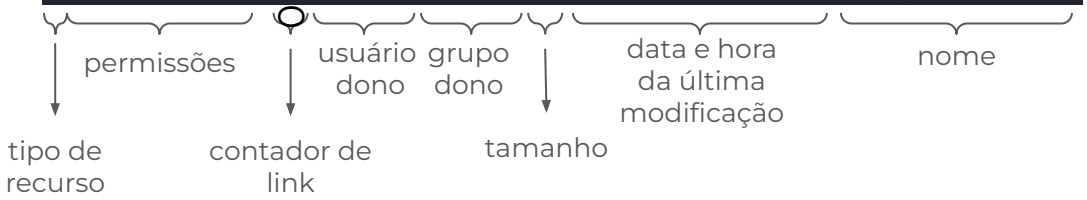
Inspecionando os metadados de um arquivo

Anteriormente:

```
└─$ echo -n 'ab' > arquivo.txt
```

```
└─$ xxd -g1 arquivo.txt
00000000: 61 62
```

```
└─$ ls -l arquivo.txt
-rw-r--r-- 1 kali kali 2 Jun  9 21:01 arquivo.txt
```




```
└─$ stat arquivo.txt
  File: arquivo.txt
  Size: 2               Blocks: 8          IO Block: 4096   regular file
Device: 8,1     Inode: 3636452     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/    kali)   Gid: ( 1000/    kali)
Access: 2024-06-09 21:01:46.140544945 -0300
Modify: 2024-06-09 21:01:34.690822934 -0300
Change: 2024-06-09 21:01:34.690822934 -0300
 Birth: 2024-06-03 21:45:26.549882220 -0300
```

Inspecionando os metadados de um arquivo

```
$ ls -l a.txt
-rw-r--r-- 1 jupyter-professor jupyter-professor 3 Jul 17 23:43 a.txt
$
$ lsattr a.txt
--------------e------- a.txt
$
$ stat a.txt
  File: a.txt
  Size: 3               Blocks: 8          IO Block: 4096   regular file
Device: ca01h/51713d    Inode: 289104      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/jupyter-professor)   Gid: ( 1003/jupyter-professor)
Access: 2022-07-17 23:28:43.964685618 +0000
Modify: 2022-07-17 23:43:22.048733405 +0000
Change: 2022-07-17 23:43:22.048733405 +0000
 Birth: 2022-07-17 23:28:43.964685618 +0000
```

Ordem dos bytes (para números)

- Número 2024
- Em binário

```
obase=2
2024
11111101000
```

- Em hexa

```
obase=16
2024
7E8
```

- Em 4 bytes:
  - 00 00 07 E8
- Big-Endian (MSB):
  - 00 00 07 E8
- Little-Endian (LSB):
  - E8 07 00 00

Ordem dos bytes (para números)

- Endianness / Byte order / Byte sex (IDA)
  - Least Significant Byte (LSB) / Little-Endian (LE)
  - Most Significant Byte (MSB) / Big-Endian (BE)

Exercício

- 435 em decimal
- Escrever em hexa (4 bytes)
  - Big-Endian
  - Little-Endian

Exercício - Respostas

- 435 em decimal
- Escrever em hexa (4 bytes)
  - Big-Endian: 00 00 01 B3
  - Little-Endian: B3 01 00 00

Demonstração

```
┌──(kali㉿kali)-[~]
└─$ python
Python 3.11.7 (main, Dec  8 2023, 14:22:46) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> f = open('a.txt', 'wb')
>>>
```

Lab 03

Visualizando arquivos em hexadecimal

Lab 03 - Visualizando arquivos em hexadecimal

1. Abra o Debian (via WSL) e crie um arquivo contendo seu nome e sobrenome:

```
echo -n "nome sobrenome" > arquivo.txt
```

Agora execute as tarefas abaixo e veja se a saída dos três programas faz **total** sentido para você.

a) Inspecione o conteúdo do arquivo com o hexyl.
b) Inspecione o conteúdo do arquivo com o hd.
c) Inspecione o conteúdo do arquivo com o od.

Se houver alguma dúvida na saída de um dos comandos, pergunte! 🙂

Lab 03 - Visualizando arquivos em hexadecimal

2. Crie um arquivo que contenha o **número** 2024 (perceba que não é o texto “2024”). Para isso, use o seguinte programa em Python:

```
ano = 2024
f = open("arquivo.bin", "wb")
f.write(ano.to_bytes(4, byteorder='little'))
f.close()
```

Agora inspecione-o com qualquer visualizador hexadecimal de linha de comando e explique os bytes encontrados lá.

Lab 03 - Respostas

1.
```
$ echo -n "nome sobrenome" > arquivo.txt
```

a) Inspecione o conteúdo do arquivo com o hexyl.

```
└─$ hexyl arquivo.txt
┌────────┬─────────────────────────┬─────────────────────────┬────────┬────────┐
│00000000│ 6e 6f 6d 65 20 73 6f 62 │ 72 65 6e 6f 6d 65       │nome sob│renome  │
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
```

b) Inspecione o conteúdo do arquivo com o hd.

```
└─$ hd arquivo.txt
00000000  6e 6f 6d 65 20 73 6f 62  72 65 6e 6f 6d 65        |nome sobrenome|
0000000e
```

c) Inspecione o conteúdo do arquivo com o od.

```
└─$ od -t x1 arquivo.txt
0000000 6e 6f 6d 65 20 73 6f 62 72 65 6e 6f 6d 65
0000016
```

-t é para definir formato
x1 é hexa de um em um byte

Lab 03 - Respostas

2. `$ python3 lab3.py`

```
ano = 2024
f = open("arquivo.bin", "wb")
f.write(ano.to_bytes(4, byteorder='little'))
f.close()
```

```
└─$ hexyl arquivo.bin
┌────────┬─────────────────────────┬─────────────────────────┬────────┬────────┐
│00000000│ e8 07 00 00             │                         │×•00    │        │
└────────┴─────────────────────────┴─────────────────────────┴────────┴────────┘
```

4 bytes:
00 00 07 E8 -> big endian
E8 07 00 00 -> little endian

Arquivos Binários

Arquivos Binários

- Depende do contexto:
  - Qualquer arquivo que não seja texto puro, ou seja, que não tenha todos os seus bytes dentro da faixa ASCII/Unicode (que veremos a seguir).
  - Arquivos executáveis (por isso /bin, /sbin, etc).
- No fim, todo arquivo pode, em teoria, ser chamado de binário.

Cadeias de Texto

ASCII 7-bits

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

Exercício

```
└─$ echo -n seu_nome | hd
00000000  73 65 75 5f 6e 6f 6d 65                           |seu_nome|
00000008
```

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

ASCII 7-bits

```
$ sudo apt install ascii
$ ascii -b
   0000000 NUL    0010000 DLE    0100000        0110000 0    1000000 @    1010000 P    1100000 `    1110000 p
   0000001 SOH    0010001 DC1    0100001 !      0110001 1    1000001 A    1010001 Q    1100001 a    1110001 q
   0000010 STX    0010010 DC2    0100010 "      0110010 2    1000010 B    1010010 R    1100010 b    1110010 r
   0000011 ETX    0010011 DC3    0100011 #      0110011 3    1000011 C    1010011 S    1100011 c    1110011 s
   0000100 EOT    0010100 DC4    0100100 $      0110100 4    1000100 D    1010100 T    1100100 d    1110100 t
   0000101 ENQ    0010101 NAK    0100101 %      0110101 5    1000101 E    1010101 U    1100101 e    1110101 u
   0000110 ACK    0010110 SYN    0100110 &      0110110 6    1000110 F    1010110 V    1100110 f    1110110 v
   0000111 BEL    0010111 ETB    0100111 '      0110111 7    1000111 G    1010111 W    1100111 g    1110111 w
   0001000 BS     0011000 CAN    0101000 (      0111000 8    1001000 H    1011000 X    1101000 h    1111000 x
   0001001 HT     0011001 EM     0101001 )      0111001 9    1001001 I    1011001 Y    1101001 i    1111001 y
   0001010 LF     0011010 SUB    0101010 *      0111010 :    1001010 J    1011010 Z    1101010 j    1111010 z
   0001011 VT     0011011 ESC    0101011 +      0111011 ;    1001011 K    1011011 [    1101011 k    1111011 {
   0001100 FF     0011100 FS     0101100 ,      0111100 <    1001100 L    1011100 \    1101100 l    1111100 |
   0001101 CR     0011101 GS     0101101 -      0111101 =    1001101 M    1011101 ]    1101101 m    1111101 }
   0001110 SO     0011110 RS     0101110 .      0111110 >    1001110 N    1011110 ^    1101110 n    1111110 ~
   0001111 SI     0011111 US     0101111 /      0111111 ?    1001111 O    1011111 _    1101111 o    1111111 DEL
$
```

Relações

- Dígitos de 0x30 a 0x39
- Maiúsculas de 0x41 a 0x5a
- Minúsculas de 0x61 a 0x7a
- Imprimíveis de 0x20 a 0x7f
- Maiúscula = minúsculas - 32 (0x20)
- Equivalente numérico = Dígito - 0x30

Upper do Python

No terminal do Python, o que faz a função upper?

```
>>> 'simplesmente'.upper()
'SIMPLESMENTE'
```

O fim de linha

- Windows (herança do MS-DOS, que herdou do CP/M) → \r\n ou CrLf ou 0x0d 0x0a
- Linux (herança do Unix) → \n ou LF ou 0x0a
  - O comando **dos2unix** "converte" ao remover os \r dos pares \r\n encontrados num arquivo.

Lab 04

Convertendo strings para maiúsculo

Lab 04 - Convertendo strings para maiúsculo

1. Escreva uma função chamada *toup()*, na linguagem que quiser, que receba um caractere e, caso este seja minúsculo, converta-o para maiúsculo. Sua função não deve utilizar **nenhuma** função de conversão de caracteres para maiúsculos, mas você pode usar strlen() para obter o tamanho de uma string. Teste sua função.

Se for escrever em C/C++, utilize o DevC++ em **retoolkit → Programming → DevCpp**. Abra o arquivo %userprofile%\desktop\binarios\toup.c e altere nele o que precisar.

Se preferir Python, abra o prompt em **retoolkit → Programming → Python Command Prompt** e depois vá até o diretório onde o esqueleto está:

```
cd %userprofile%/desktop/binarios
python toup.py
```

Os esqueletos não estão alterando a string. Você precisa implementar a lógica que vai realizar essa ação.

Teste sua função com diferentes strings, de diferentes tamanhos e contendo caracteres alfanuméricos e de pontuação.

Lab 04 - Convertendo strings para maiúsculo

C

```c
#include <stdio.h>
#include <string.h>

char* toup(char *s) {
        for (int i=0; i < strlen(s); i++) {
                s[i] = s[i];
        }
        return s;
}

int main(void) {
        char s[] = "menteb.in";
        puts(toup(s));
}
```

Python

```python
def toup(s):
    temp = ""
    for c in s:
        temp = temp + chr(ord(c))
    return temp

s = "menteb.in"
print(toup(s))
```

Lab 04 - Strings para maiúsculo - Respostas

C

```c
#include <stdio.h>
#include <string.h>

char* toup(char *s) {
    for (int i=0; i < strlen(s); i++) {
        if (s[i] >= 0x61 && s[i] <= 0x7a) s[i] = s[i] - 32;
    }
    return s;
}
int main(void) {
    char s[] = "menteb.in";
    puts(toup(s));
}
```

ASCII 7-bits - Curiosidades

- 0x00 é NUL / nullbyte
- 0x08 é o BS / backspace

```
└─$ echo -e "menteb\x08.in"
mente.in
```

- 0x07 é o BEL

```
└─$ echo -e "\x07"
```

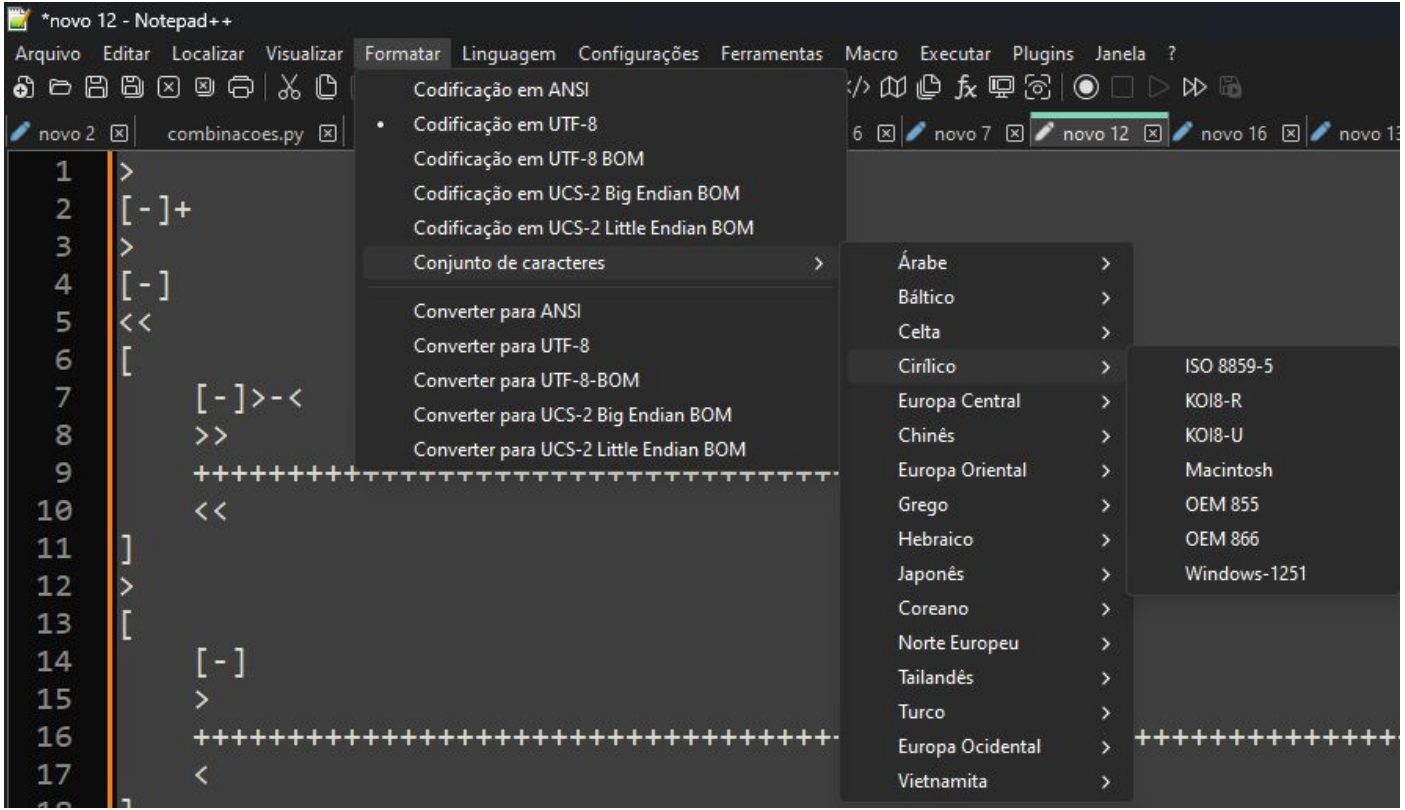
- 0x09 é o HT / horizontal tab

```
└─$ echo -e "menteb\x09.in"
menteb   .in
```

ASCII estendida

- 8 bits
- Não é um padrão
- Centenas de versões diferentes
- A mais famosa é a ISO 8859-1, ou ISO Latin 1, ou o seu superconjunto Windows *(Code Page)* CP-1252
- Altamente dependente do *encoding*

Encoding


*novo 12 - Notepad++
Arquivo Editar Localizar Visualizar Formatar Linguagem Configurações Ferramentas Macro Executar Plugins Janela ?
novo 2
combinacoes.py
novo 7
novo 12
novo 16
Codificação em ANSI
Codificação em UTF-8
Codificação em UTF-8 BOM
Codificação em UCS-2 Big Endian BOM
Codificação em UCS-2 Little Endian BOM
Conjunto de caracteres
Converter para ANSI
Converter para UTF-8
Converter para UTF-8-BOM
Converter para UCS-2 Big Endian BOM
Converter para UCS-2 Little Endian BOM
Árabe
Báltico
Celta
Cirílico
Europa Central
Chinês
Europa Oriental
Grego
Hebraico
Japonês
Coreano
Norte Europeu
Tailandês
Turco
Europa Ocidental
Vietnamita
ISO 8859-5
KOI8-R
KOI8-U
Macintosh
OEM 855
OEM 866
Windows-1251

ASCII estendida (Windows CP-1252 / ISO 8859-1)

Ø	Latin capital letter O with slash	216	D8	&#216;	&Oslash;	┼	Box drawings vertical single and horizontal double
Ù	Latin capital letter U with grave	217	D9	&#217;	&Ugrave;	┘	Box drawings light up and left
Ú	Latin capital letter U with acute	218	DA	&#218;	&Uacute;	┌	Box drawings light down and right
Û	Latin capital letter U with circumflex	219	DB	&#219;	&Ucirc;	■	Full block
Ü	Latin capital letter U with diaeresis	220	DC	&#220;	&Uuml;	▄	Lower half block
Ý	Latin capital letter Y with acute	221	DD	&#221;	&Yacute;	▌	Left half block
Þ	Latin capital letter THORN	222	DE	&#222;	&THORN;	▐	Right half block
ß	Latin small letter sharp s - ess-zed	223	DF	&#223;	&szlig;	▀	Upper half block
à	Latin small letter a with grave	224	E0	&#224;	&agrave;	α	Greek lower case alpha
á	Latin small letter a with acute	225	E1	&#225;	&aacute;	ß	Lower case sharp s
â	Latin small letter a with circumflex	226	E2	&#226;	&acirc;	Γ	Greek upper case letter gamma
ã	Latin small letter a with tilde	227	E3	&#227;	&atilde;	π	Greek lower case pi
ä	Latin small letter a with diaeresis	228	E4	&#228;	&auml;	Σ	Greek upper case letter sigma
å	Latin small letter a with ring above	229	E5	&#229;	&aring;	σ	Greek lower case sigma
æ	Latin small letter ae	230	E6	&#230;	&aelig;	µ	Micro sign
ç	Latin small letter c with cedilla	231	E7	&#231;	&ccedil;	τ	Greek lower case tau
è	Latin small letter e with grave	232	E8	&#232;	&egrave;	Φ	Greek upper case letter phi
é	Latin small letter e with acute	233	E9	&#233;	&eacute;	Θ	Greek upper case letter theta
ê	Latin small letter e with circumflex	234	EA	&#234;	&ecirc;	Ω	Greek upper case letter omega
ë	Latin small letter e with diaeresis	235	EB	&#235;	&euml;	δ	Greek lower case delta
ì	Latin small letter i with grave	236	EC	&#236;	&igrave;	∞	Infinity

(cortada)

Unicode

- Também chamada de *wide* strings
- UTF-8
  - Caracteres de 1 a 4 bytes
  - Ex.: 'á' é a dupla 0xc3 0xa1
- UTF-16 / UCS-2
  - LE ou BE
  - 2 bytes para cada caractere, com zeros se preciso.
  - Windows "encoda" CP-1252 em UTF-16-LE.
  - Usada em .NET no Windows.
  - Termina com **dois** null bytes.

```
└─$ echo -n 'água' | hd
00000000  c3 a1 67 75 61        |..gua|
00000005
```

Unicode

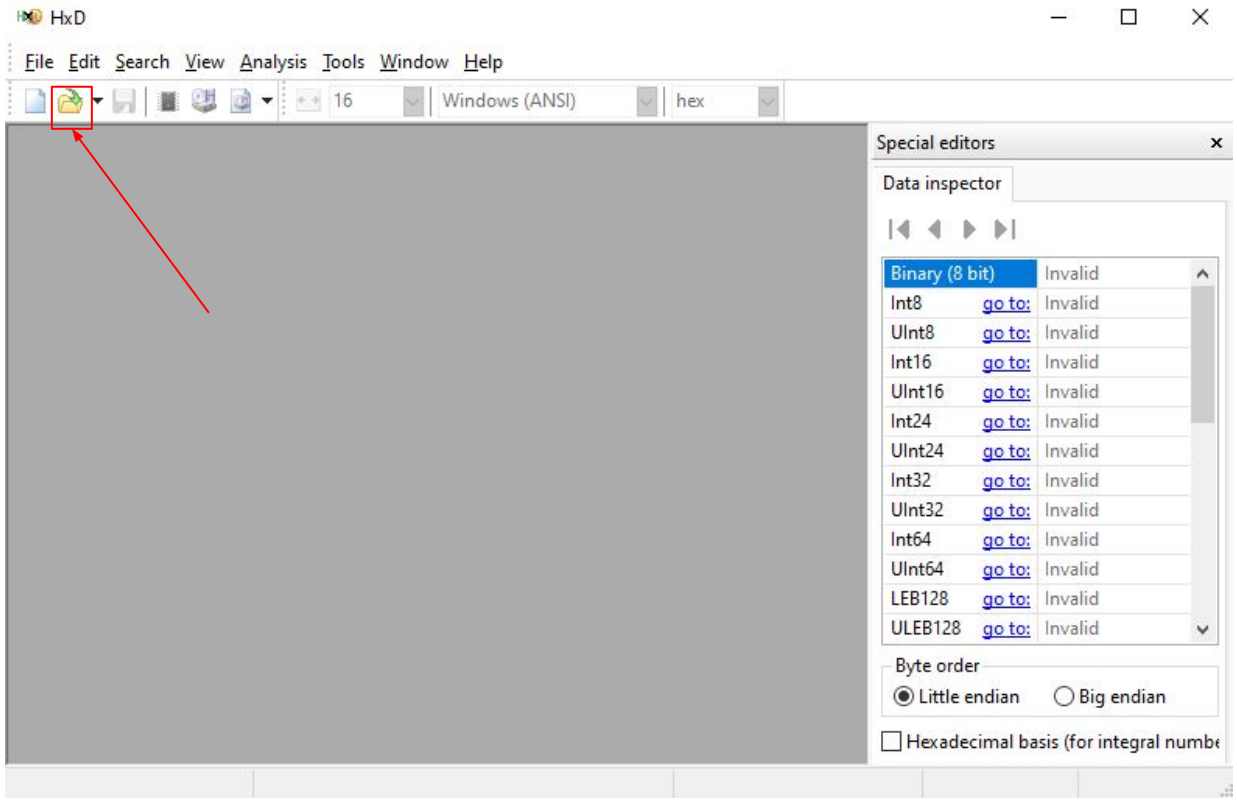
- UTF-32
  - LE ou BE
  - 4 bytes para cada caractere, com zeros se preciso.
  - Usada pela *wchar.h* no Linux.

C Strings

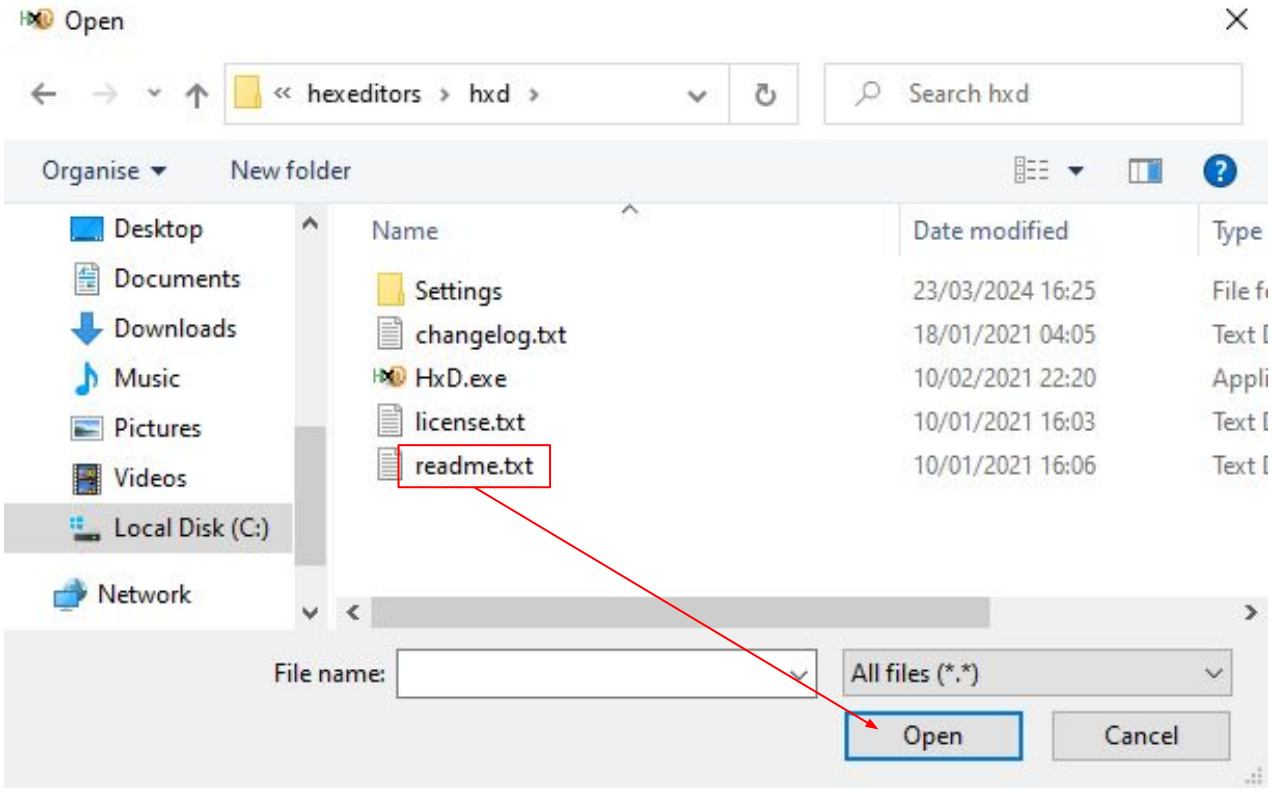
- Programas em C adicionam um caractere nulo no final da string.
- Saber disso é útil, por exemplo, para cortar uma string quando precisamos.

Editores Hexadecimais

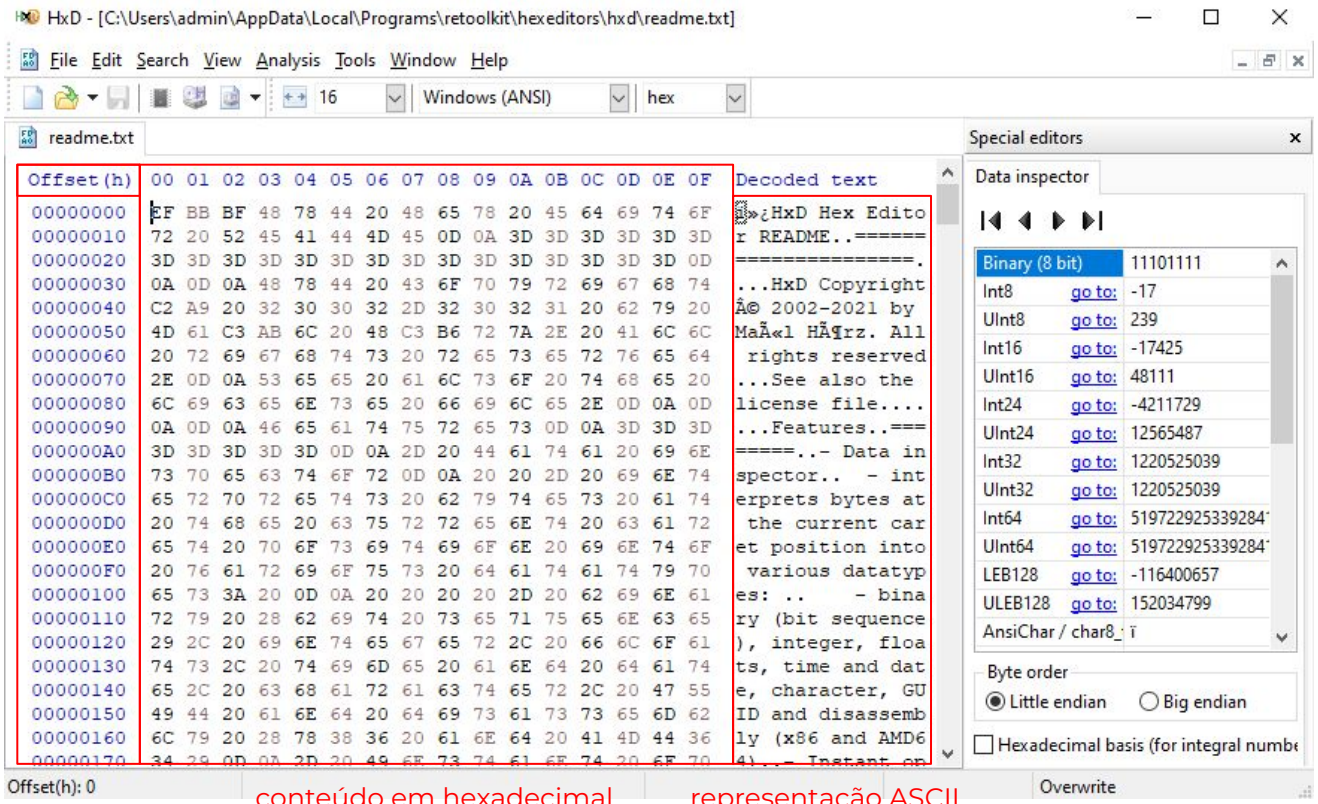
Usando o HxD


HxD
File Edit Search View Analysis Tools Window Help
16
Windows (ANSI)
hex
Special editors
Data inspector
Binary (8 bit) Invalid
Int8 go to: Invalid
UInt8 go to: Invalid
Int16 go to: Invalid
UInt16 go to: Invalid
Int24 go to: Invalid
UInt24 go to: Invalid
Int32 go to: Invalid
UInt32 go to: Invalid
Int64 go to: Invalid
UInt64 go to: Invalid
LEB128 go to: Invalid
ULEB128 go to: Invalid
Byte order
Little endian
Big endian
Hexadecimal basis (for integral numbe

Usando o HxD


Open
hexeditors › hxd ›
Search hxd
Organise
New folder
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Network
Name
Date modified
Type
Settings
23/03/2024 16:25
changelog.txt
18/01/2021 04:05
HxD.exe
10/02/2021 22:20
license.txt
10/01/2021 16:03
readme.txt
10/01/2021 16:06
File name:
All files (*.*)
Open
Cancel

Usando o HxD



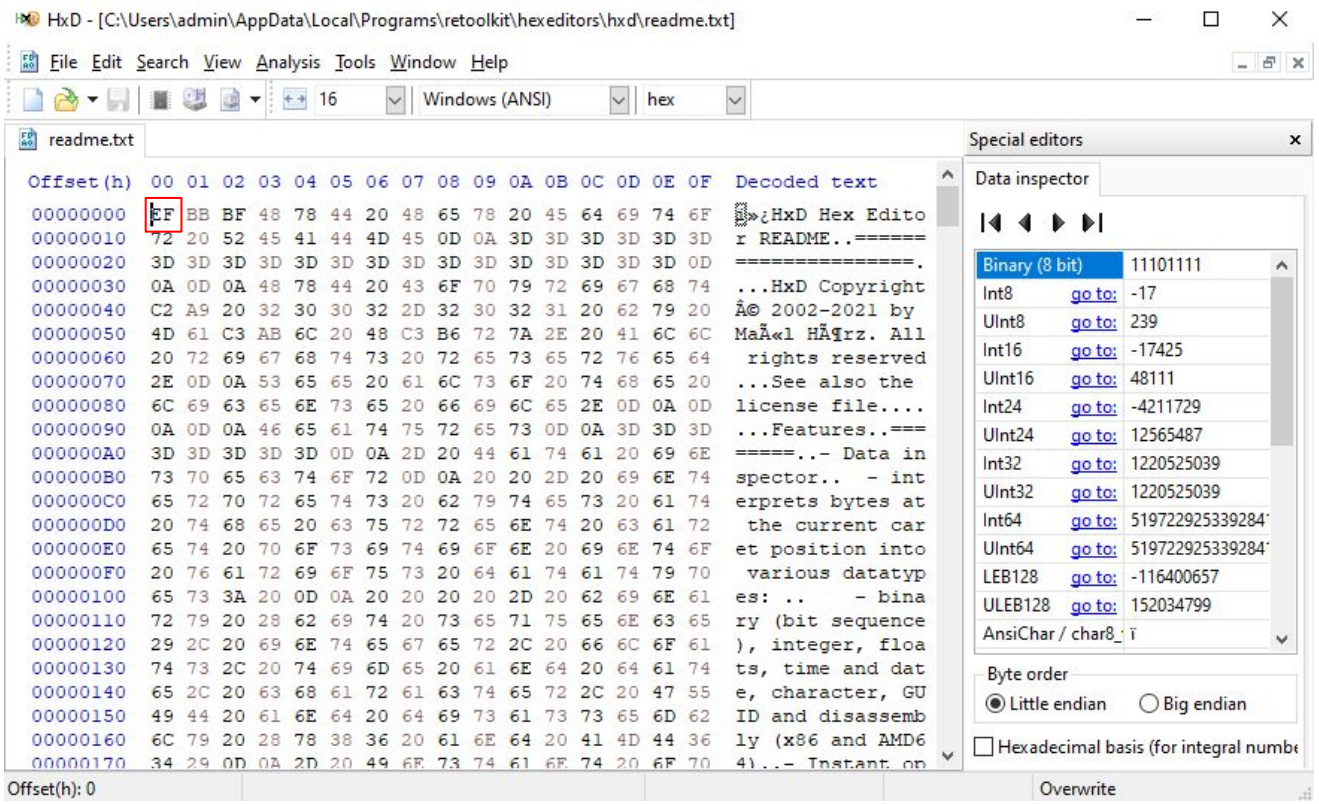
conteúdo em hexadecimal

representação ASCII

Usando o HxD - offset

Qual o byte nas seguintes posições?

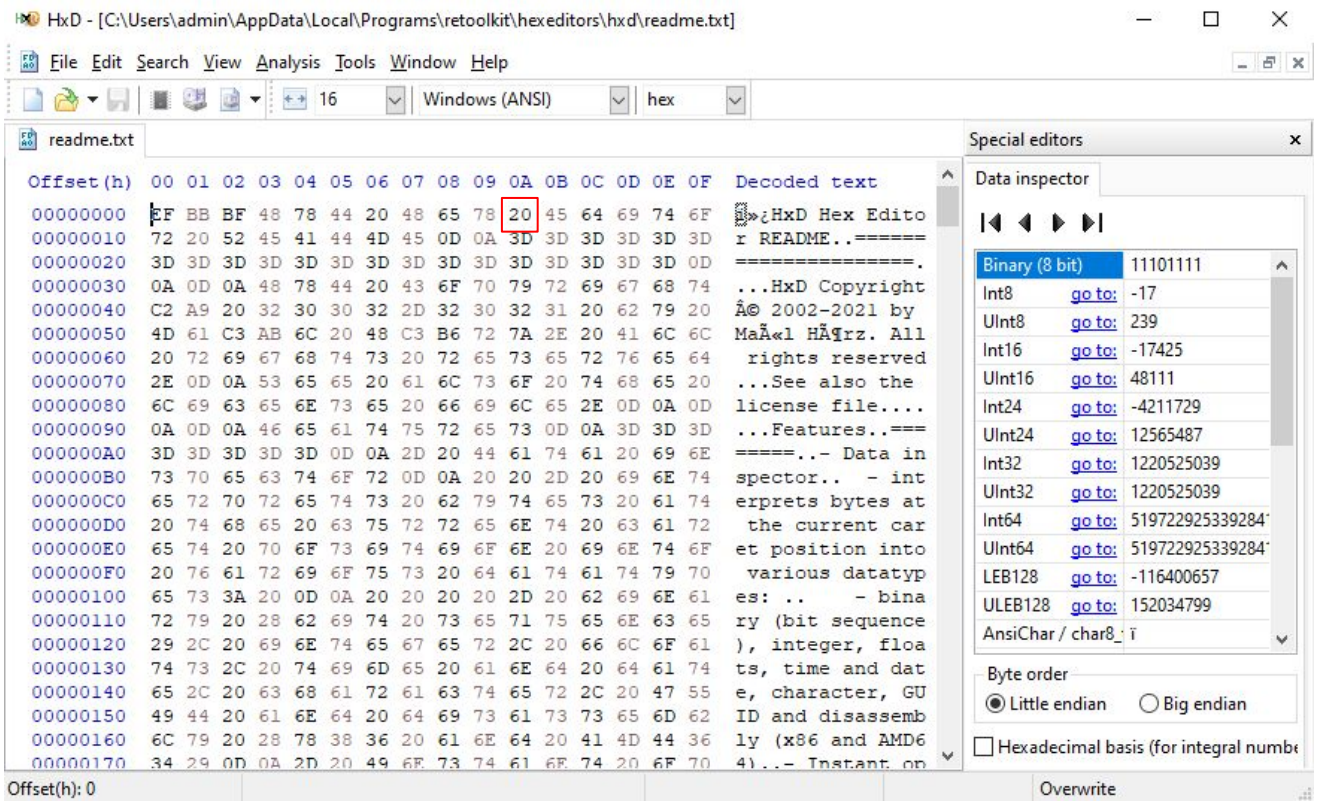
- 0



Usando o HxD - offset

Qual o byte nas seguintes posições?

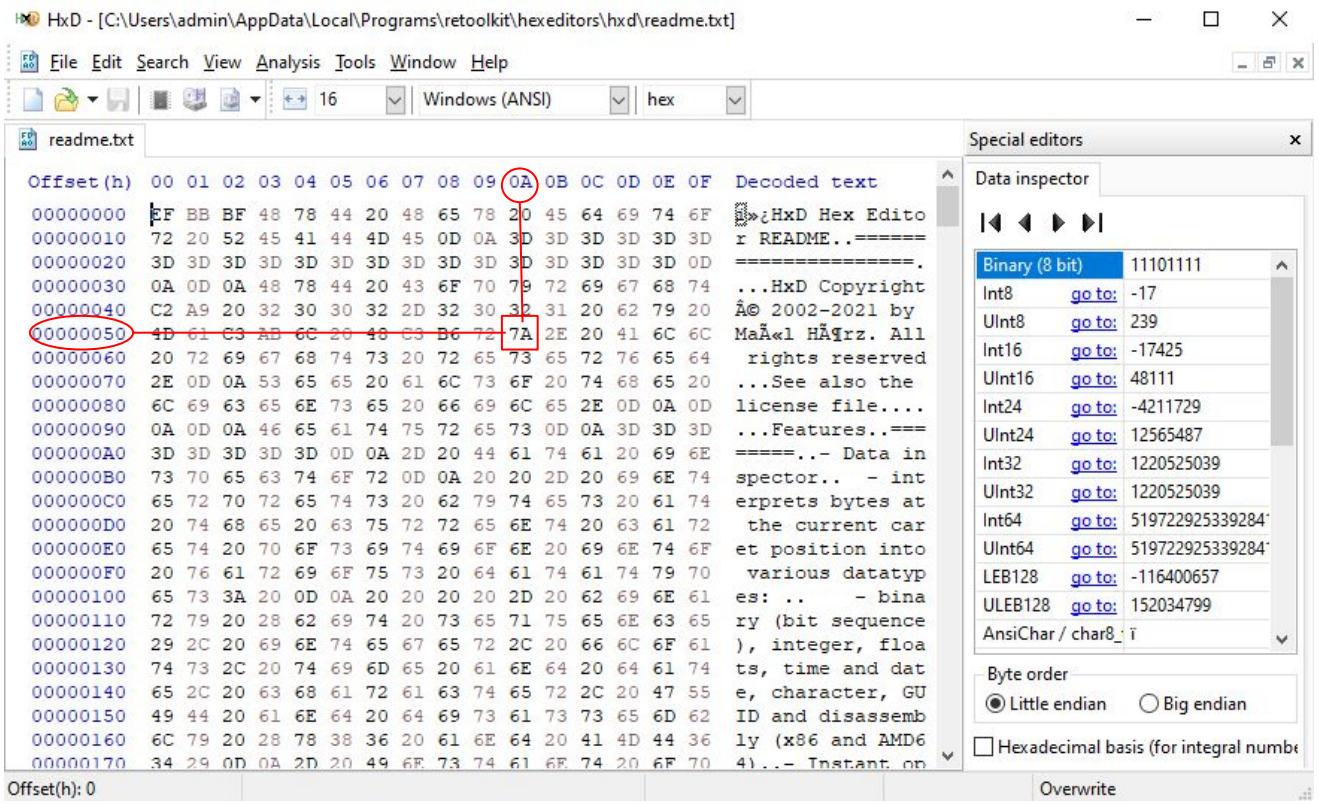
- A



Usando o HxD - offset

Qual o byte nas seguintes posições?

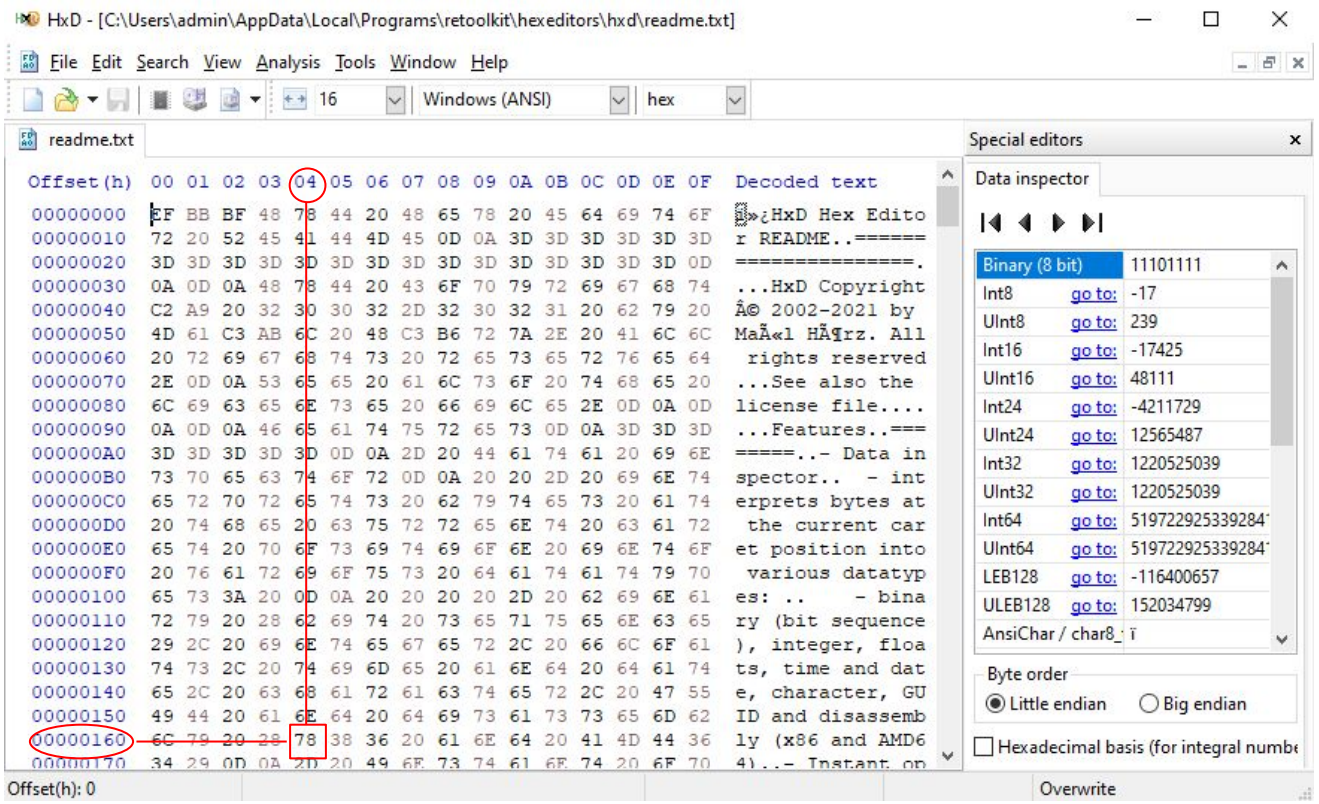
- 5A



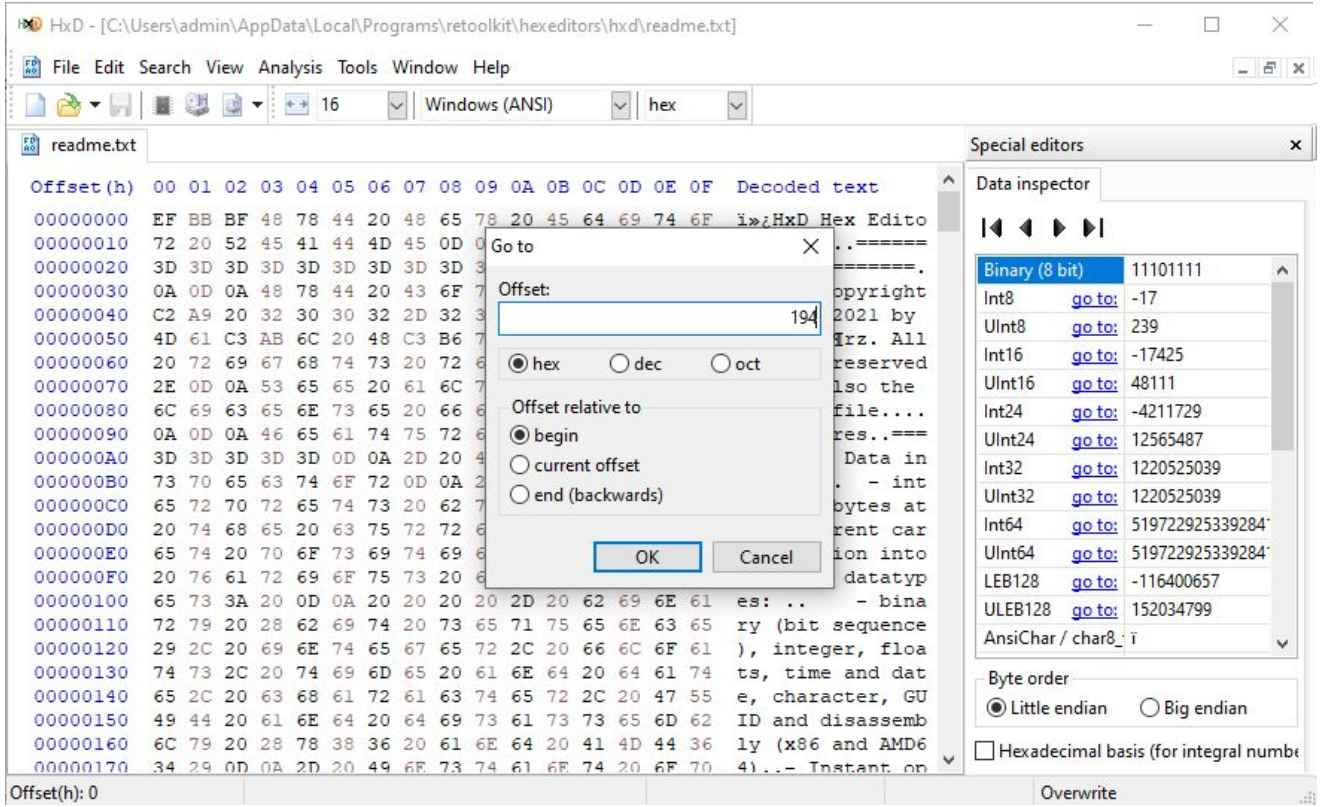
Usando o HxD - offset

Qual o byte nas seguintes posições?

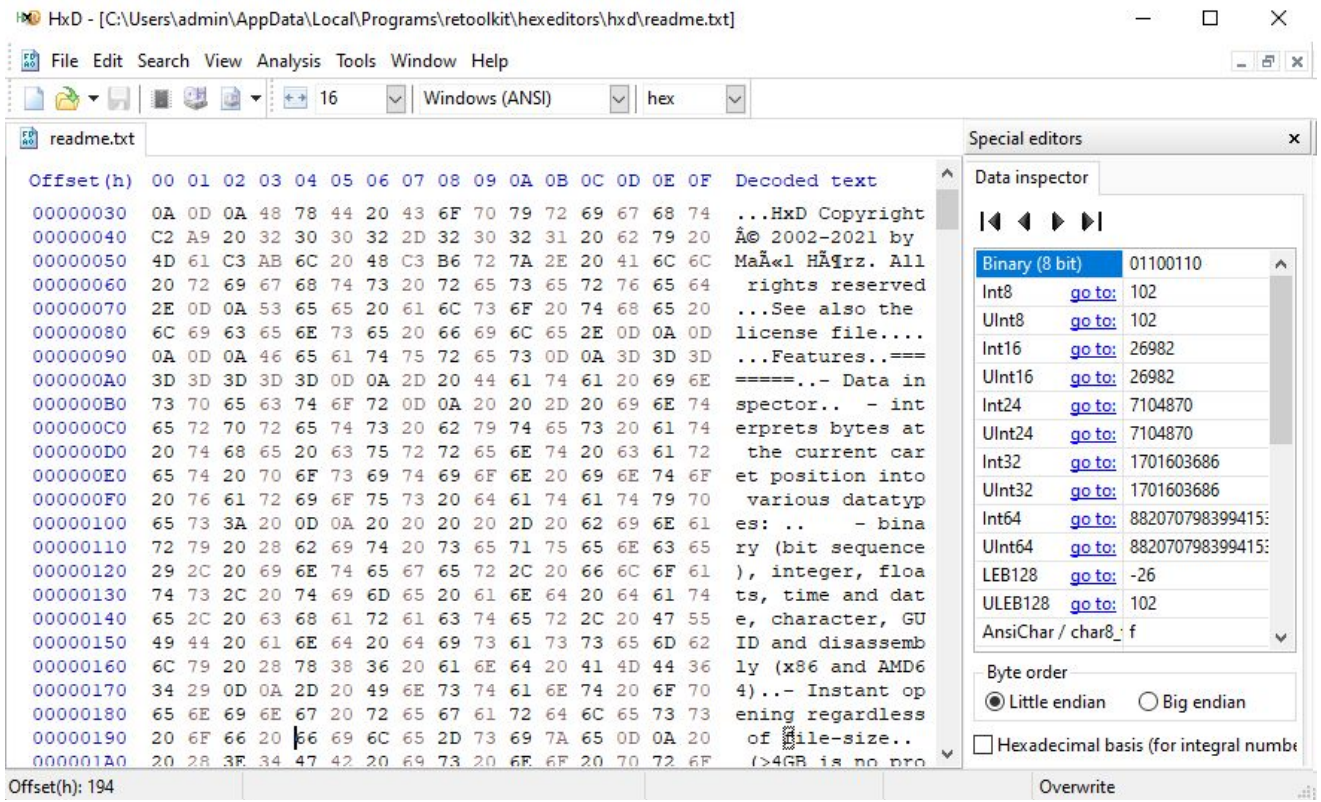
- 164



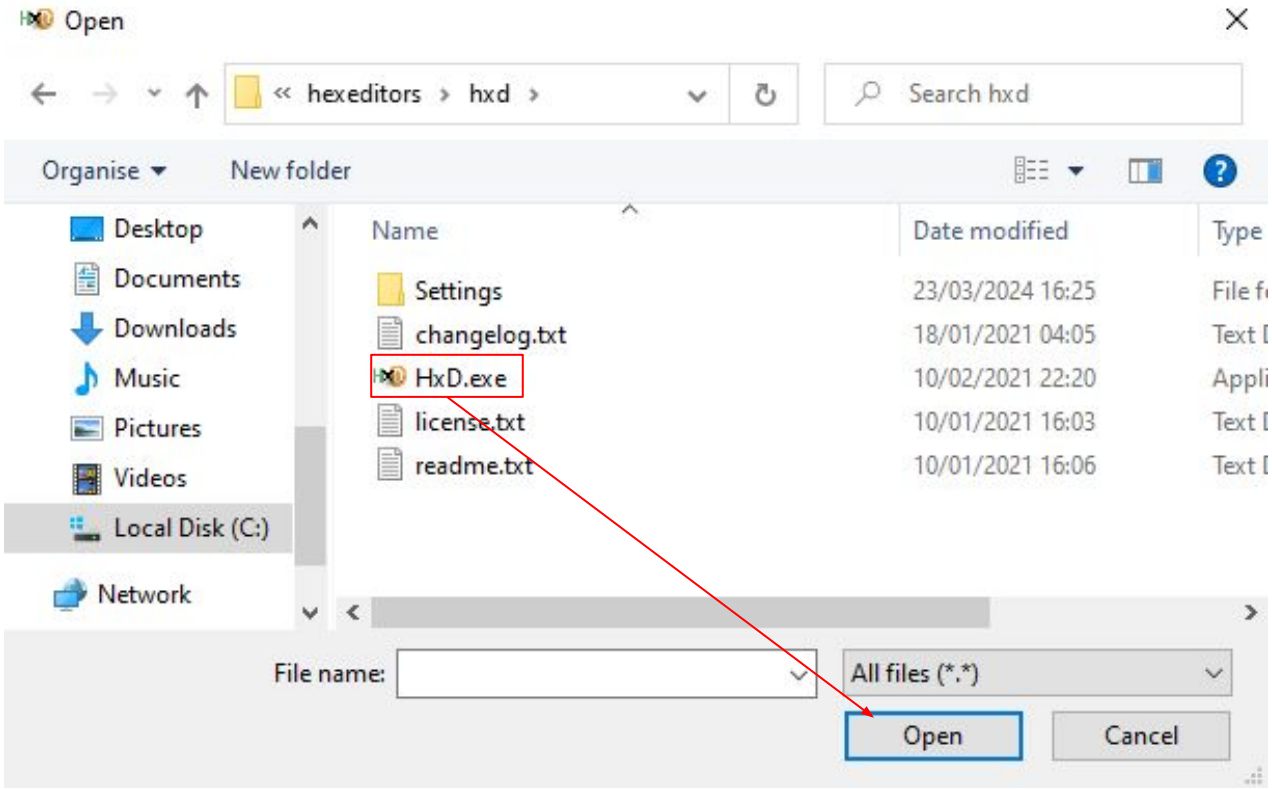
Usando o HxD - Go to



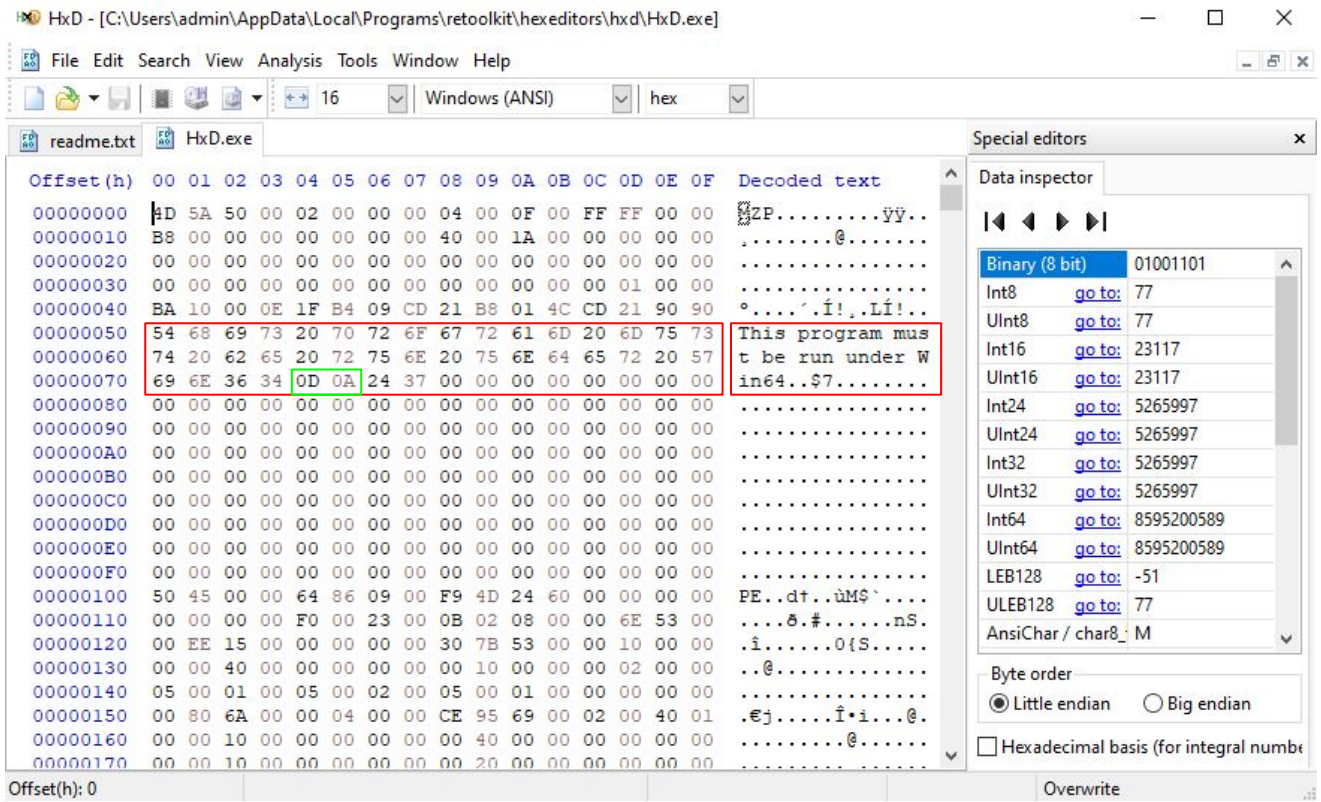
Usando o HxD - Go to



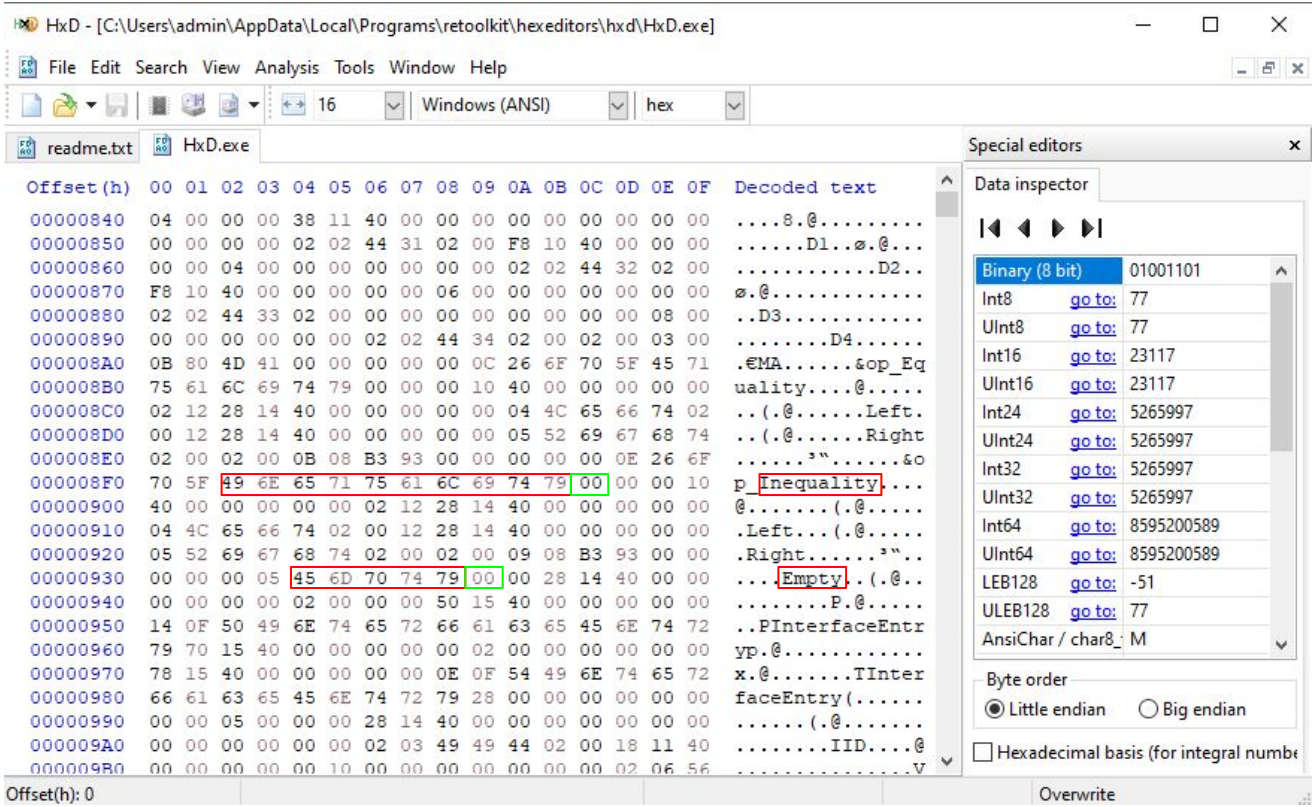
Usando o HxD


Open
hexeditors › hxd ›
Search hxd
Organise
New folder
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Network
Name
Date modified
Type
Settings
23/03/2024 16:25
changelog.txt
18/01/2021 04:05
HxD.exe
10/02/2021 22:20
license.txt
10/01/2021 16:03
readme.txt
10/01/2021 16:06
File name:
All files (*.*)
Open
Cancel

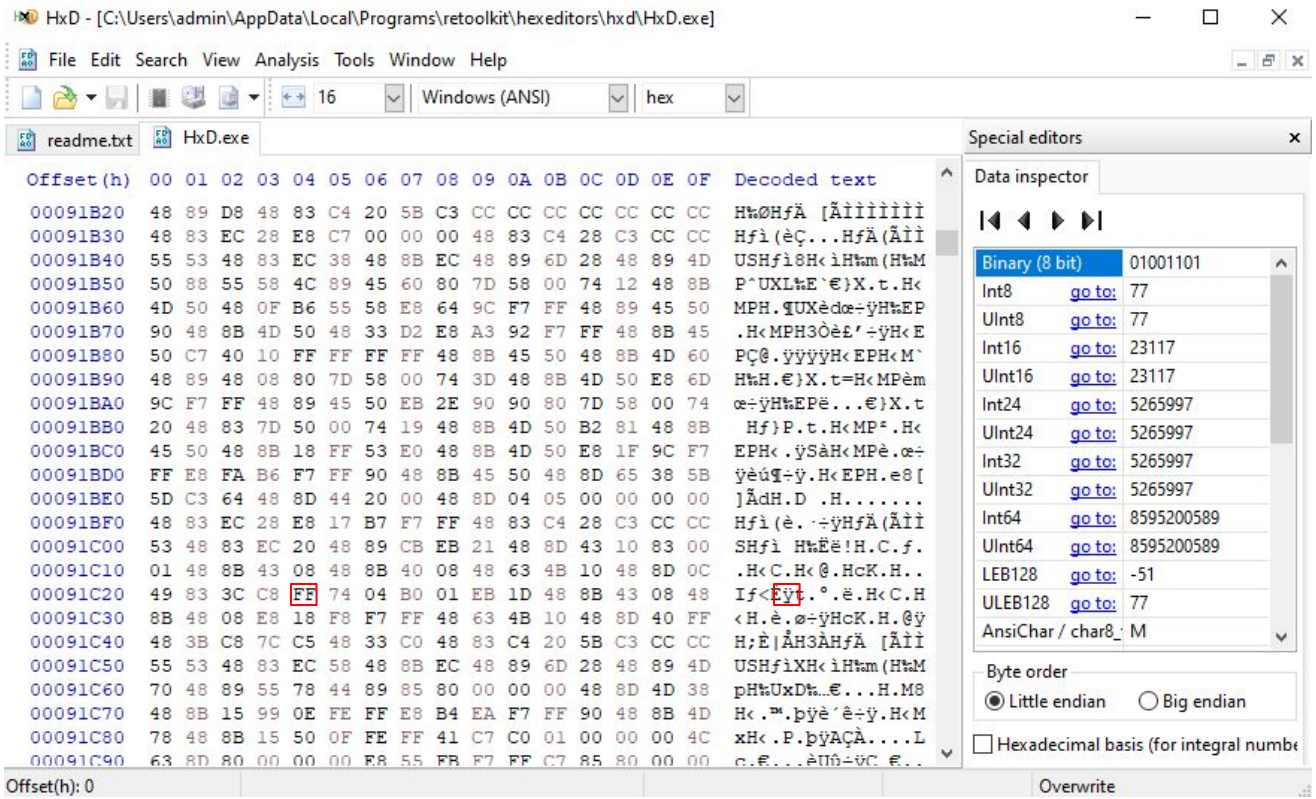
Usando o HxD - fim de linha



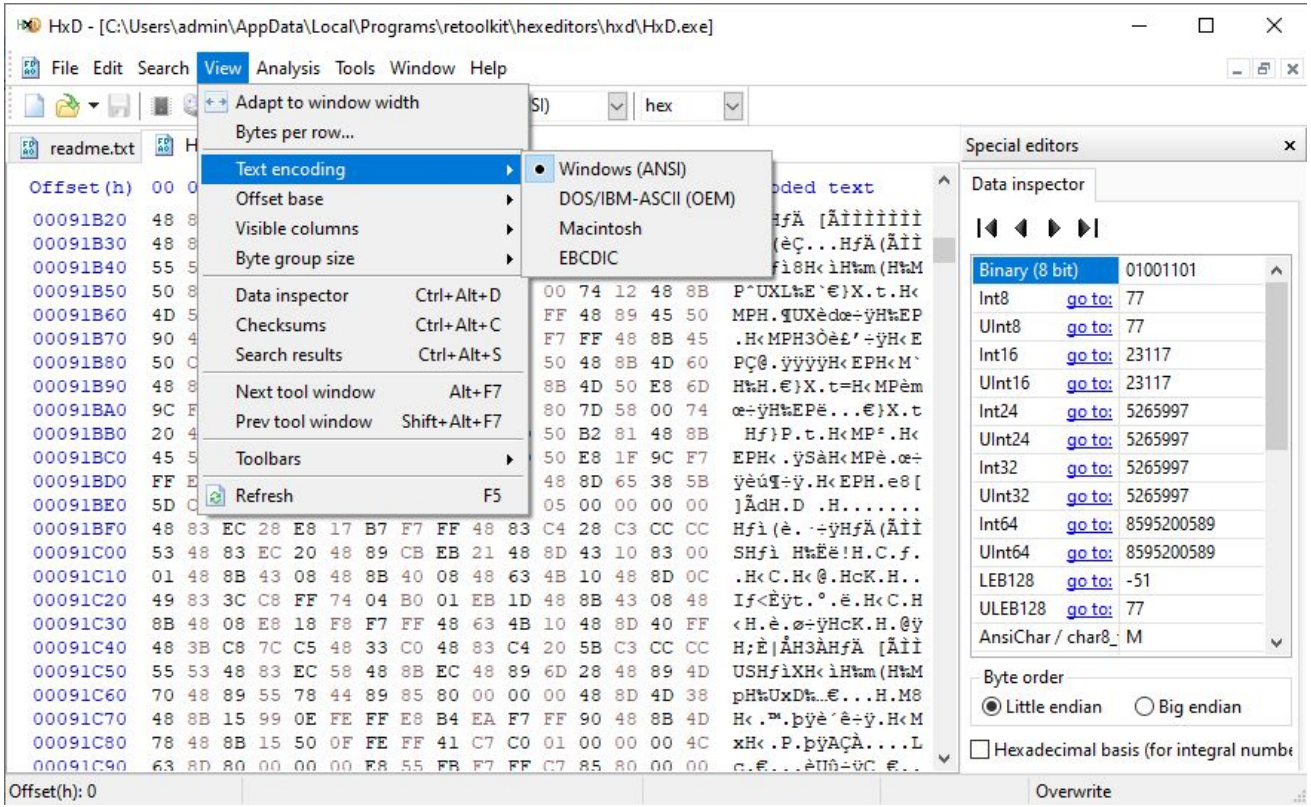
Usando o HxD - encoding



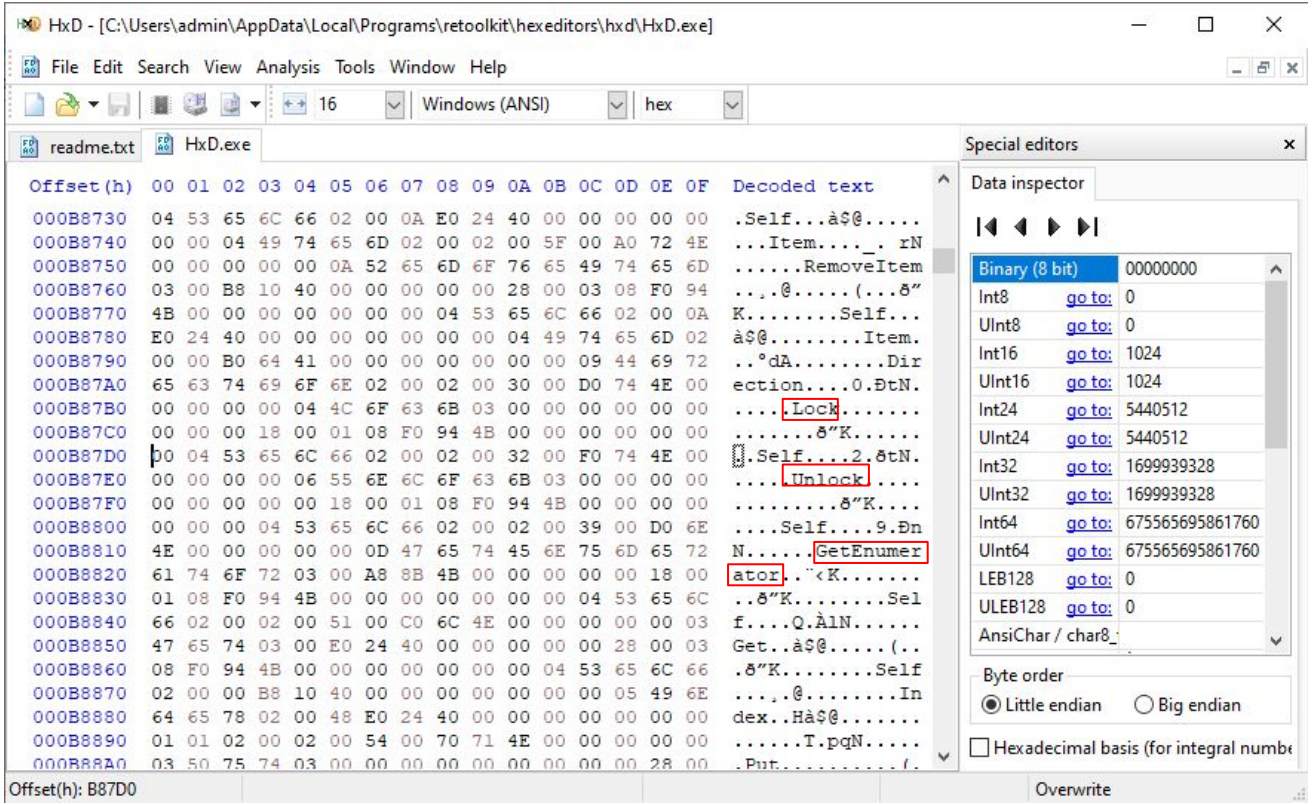
Usando o HxD - encoding



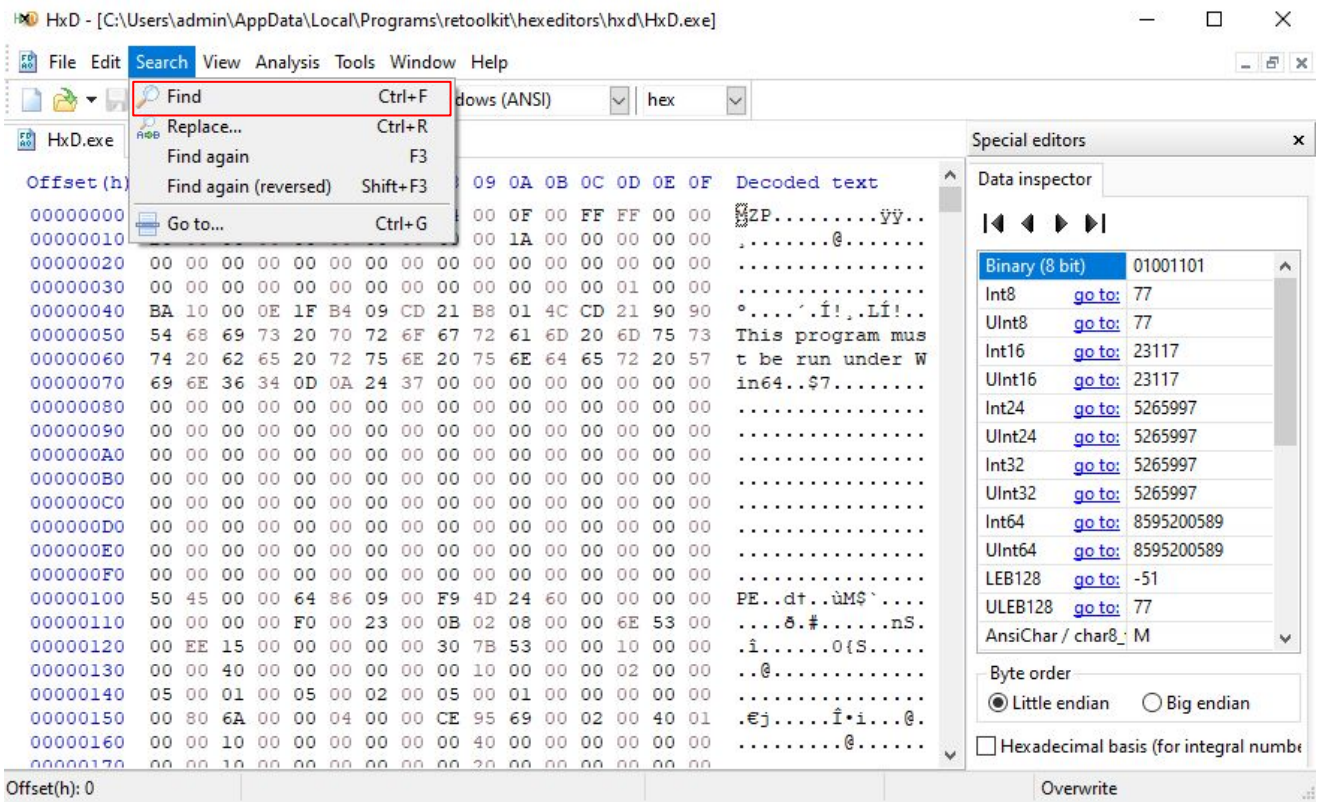
Usando o HxD - encoding



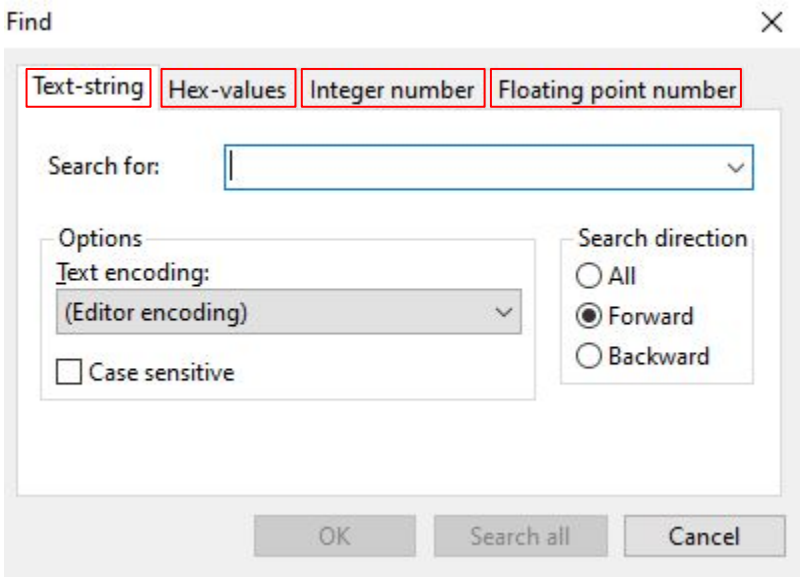
Usando o HxD - encoding



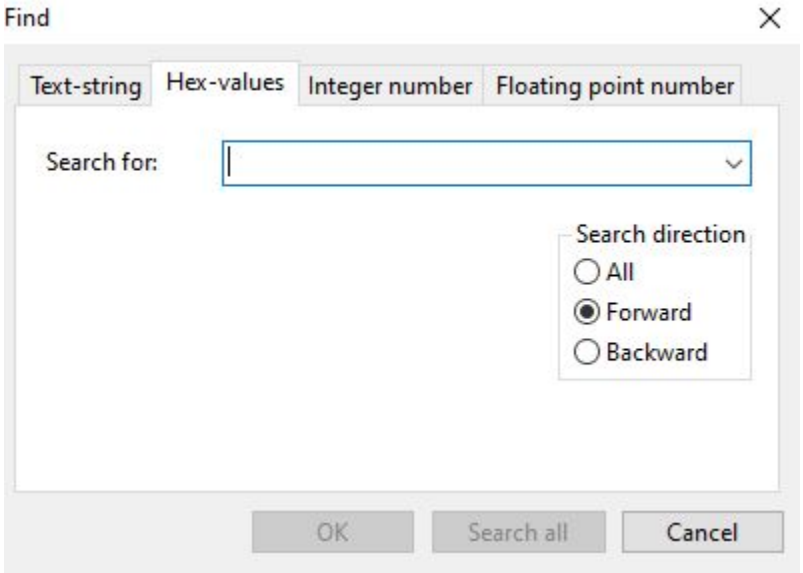
Usando o HxD - Busca



Usando o HxD - Busca



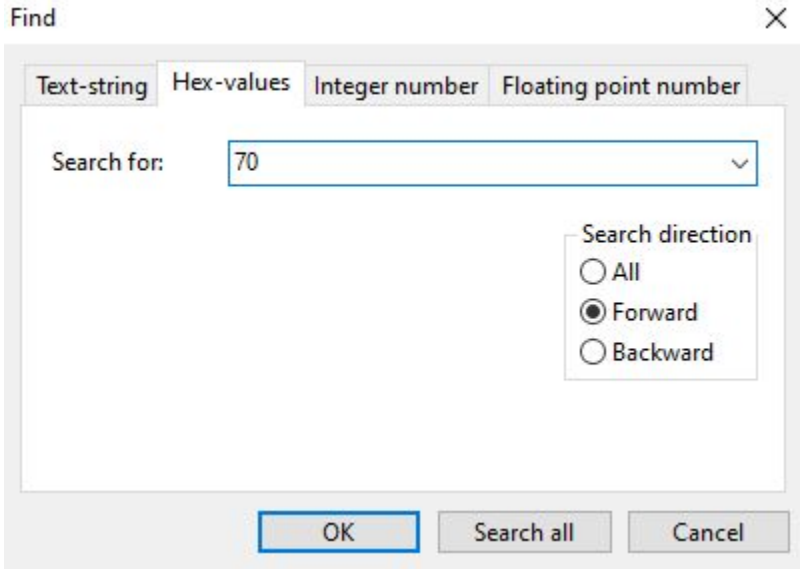
Usando o HxD - Busca hexa por string “pro”



Usando o HxD - Busca hexa por string “pro”

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

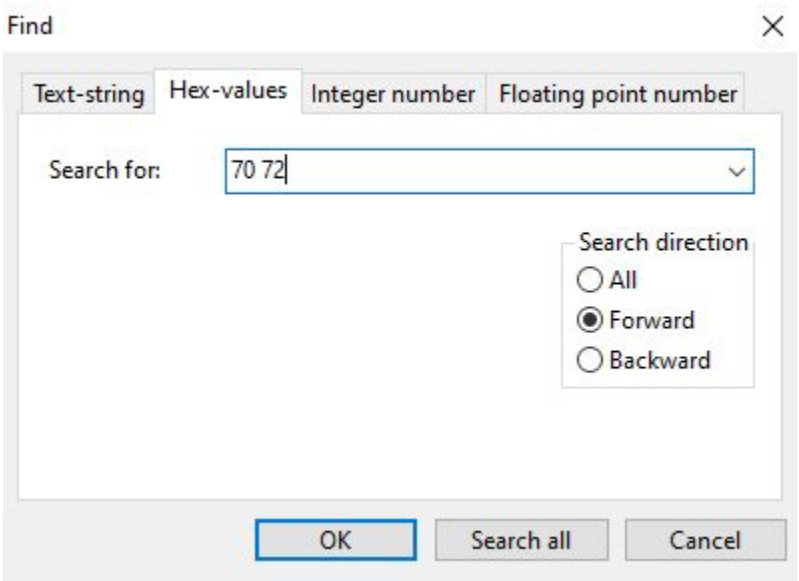
Usando o HxD - Busca hexa por string “pro”



Usando o HxD - Busca hexa por string “pro”

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

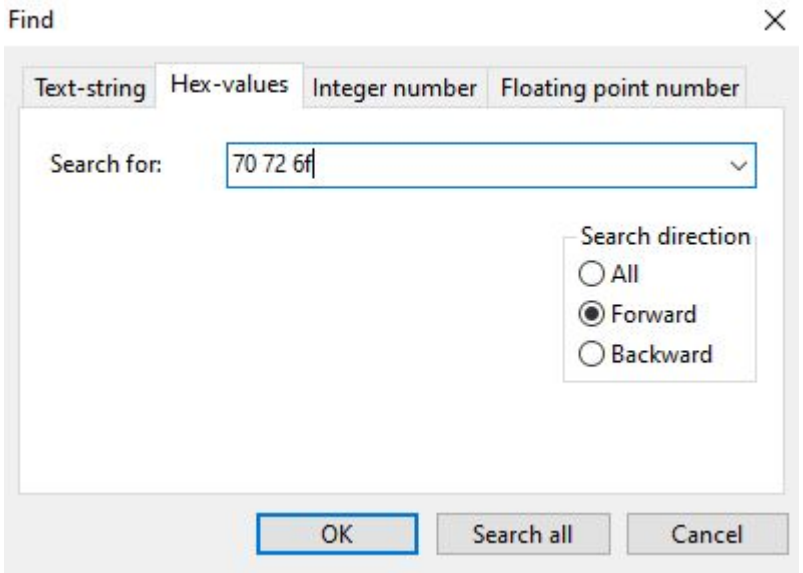
Usando o HxD - Busca hexa por string “pro”



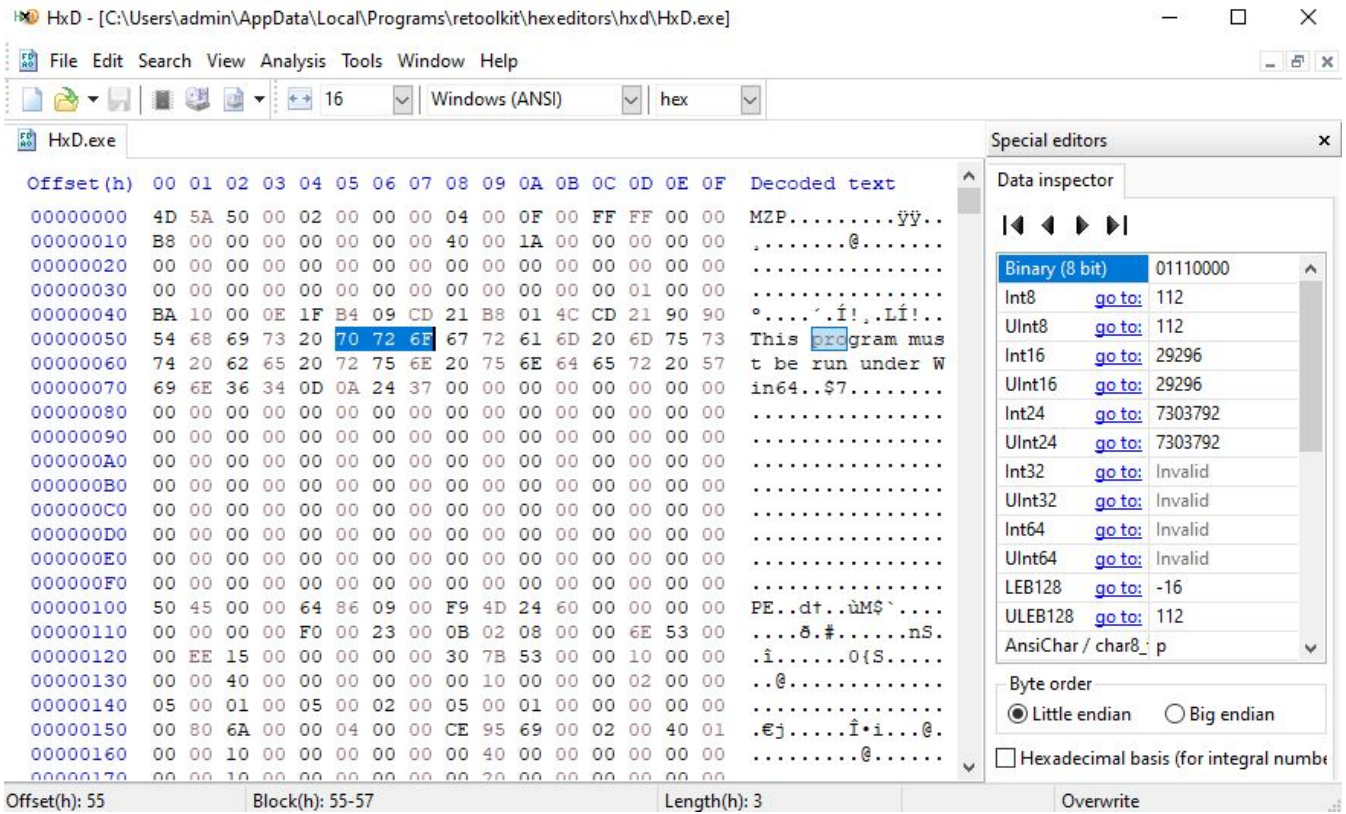
Usando o HxD - Busca hexa por string “pro”

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

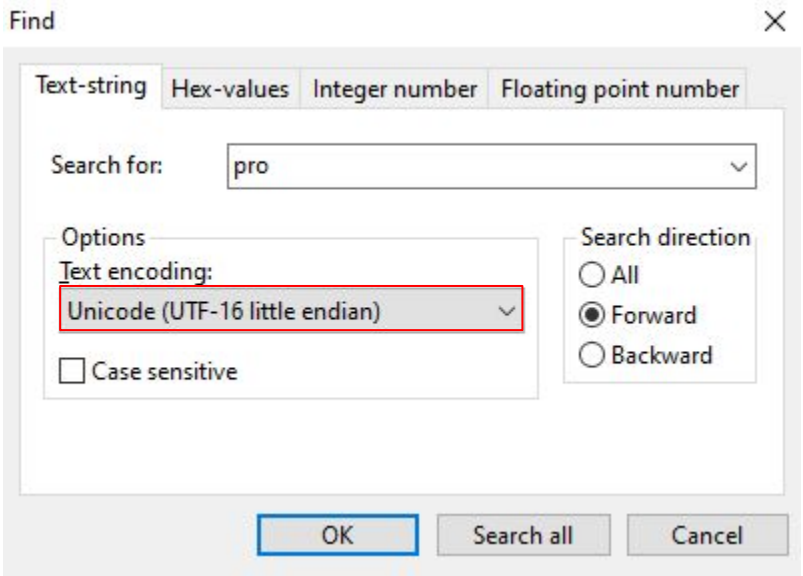
Usando o HxD - Busca hexa por string “pro”



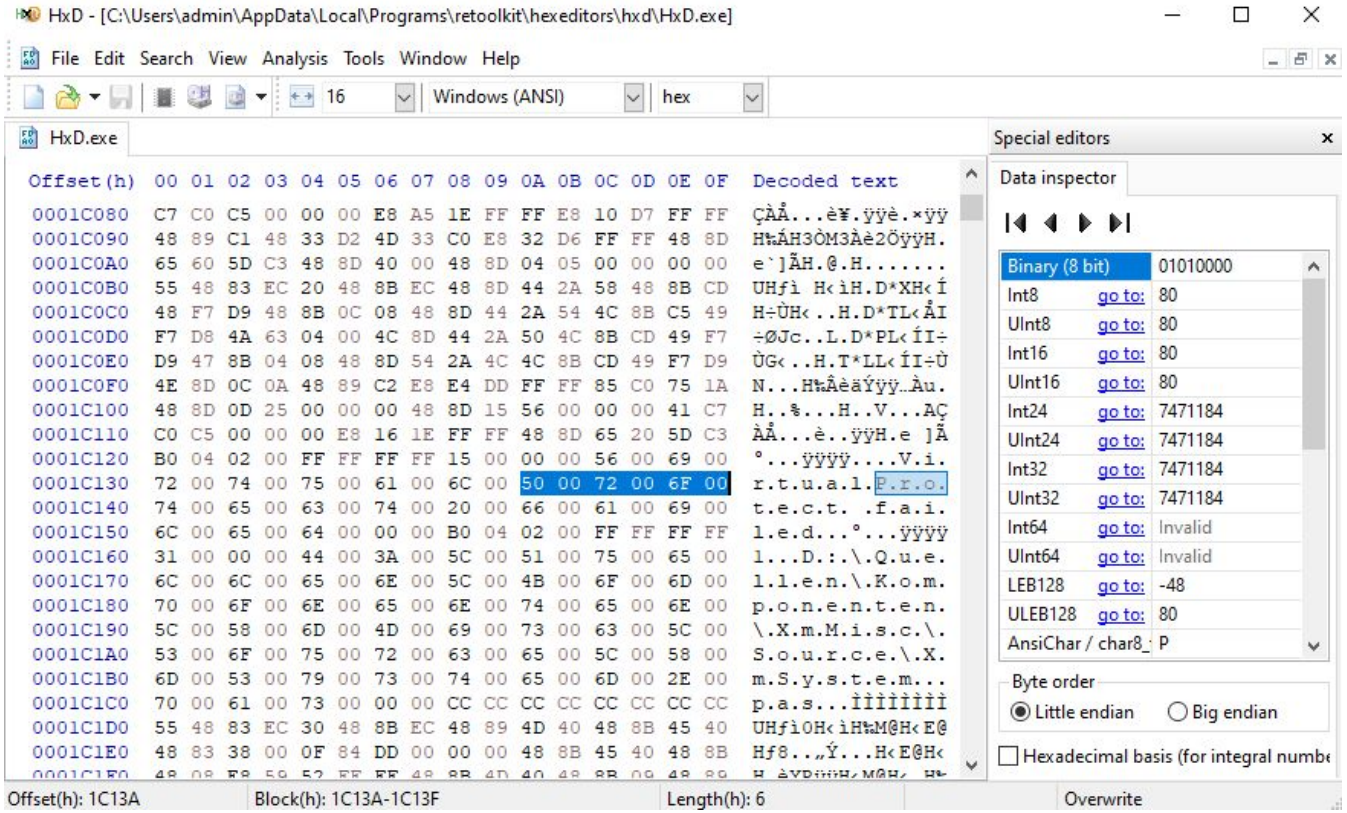
Usando o HxD - Busca hexa por string “pro”



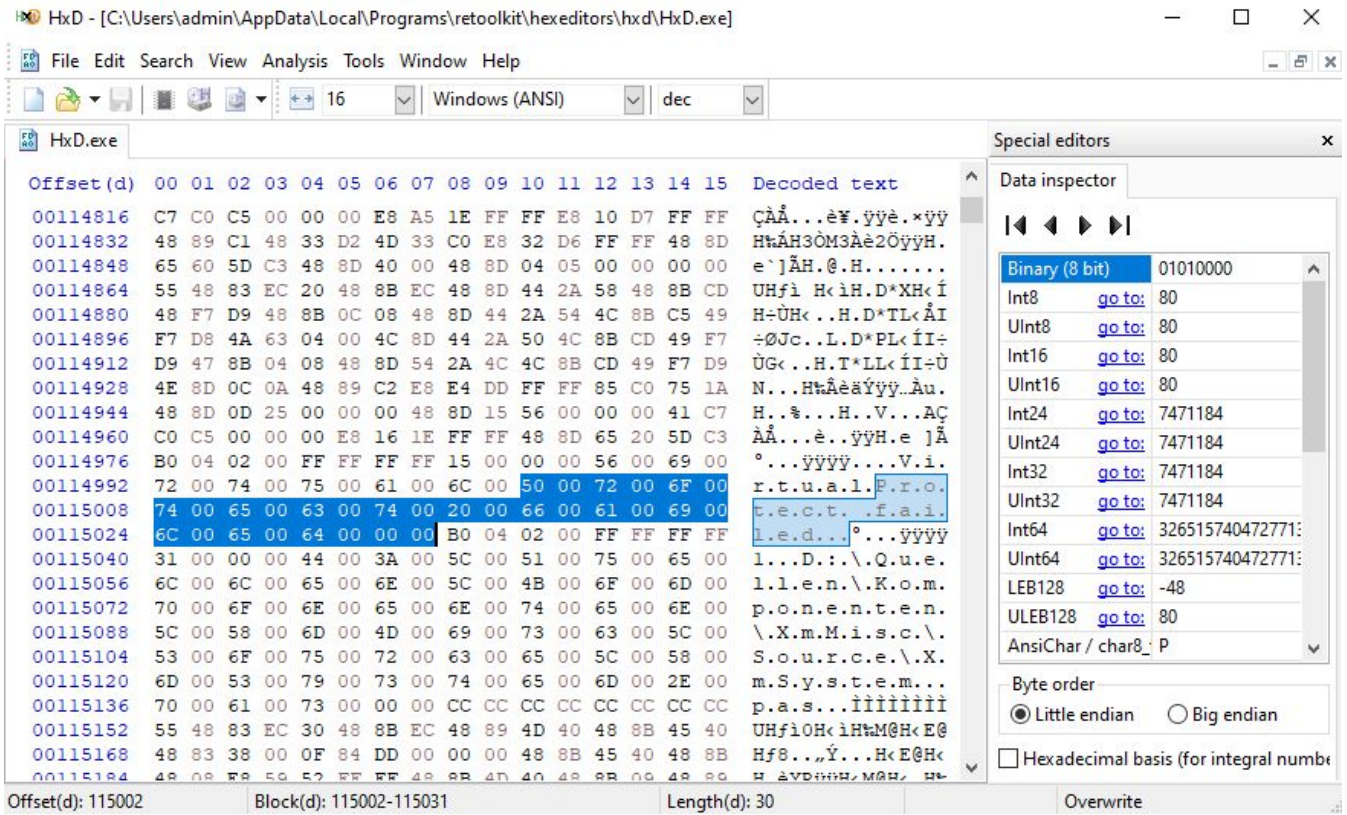
Usando o HxD - Busca por string “pro”



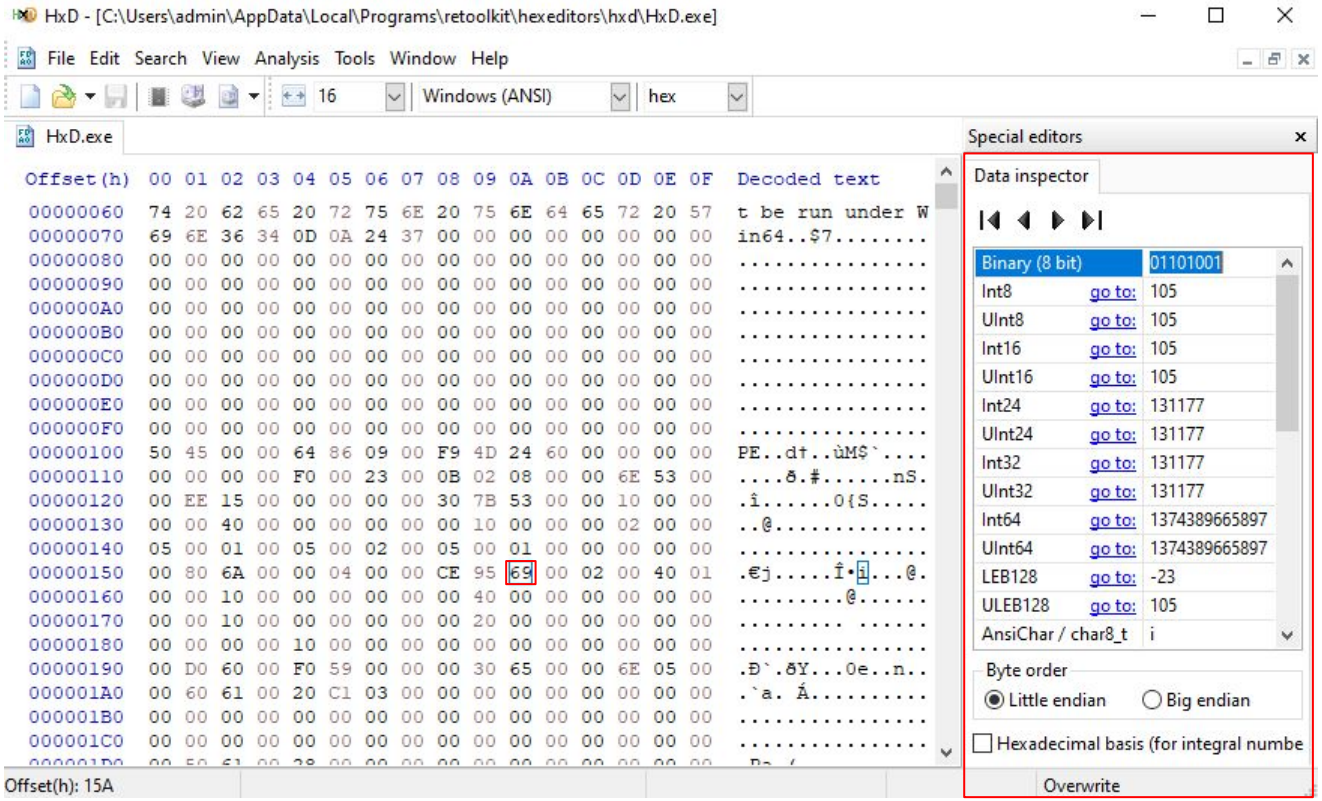
Usando o HxD - Busca por string “pro”



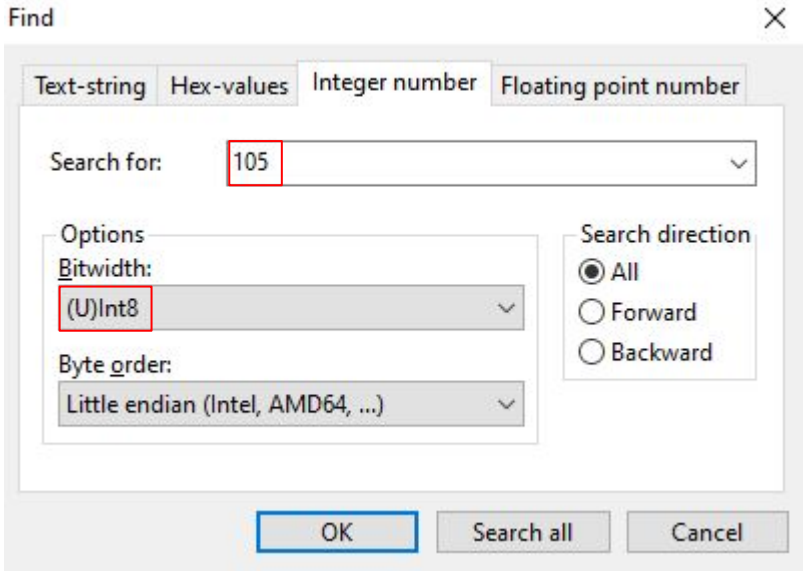
Usando o HxD - Busca por string “pro”



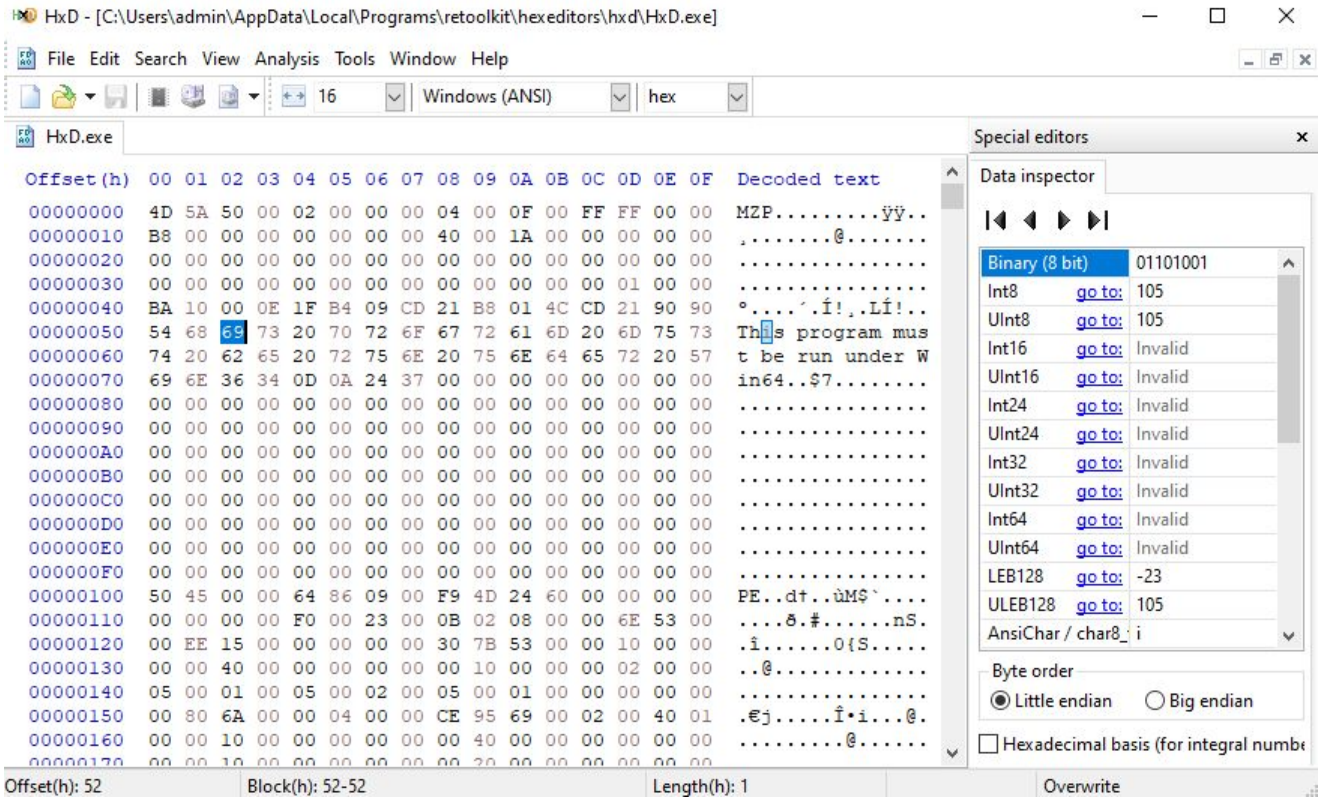
Usando o HxD - Busca por inteiro



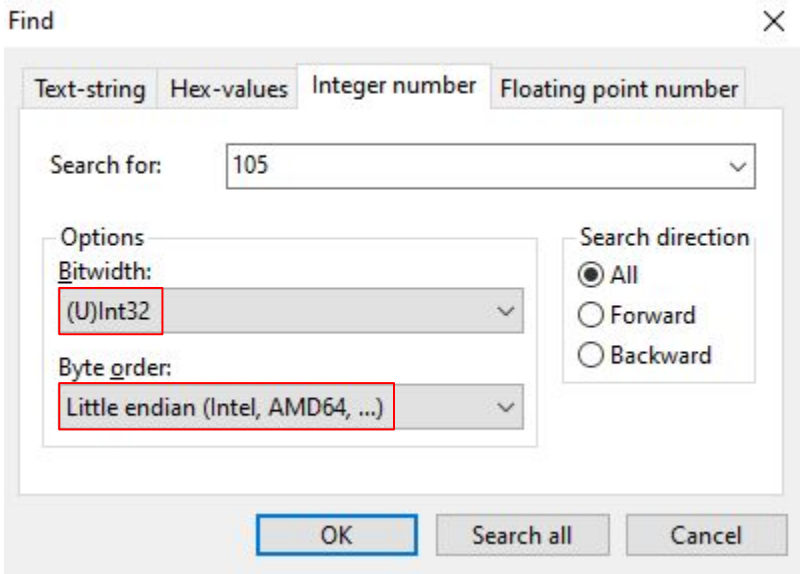
Usando o HxD - Busca por inteiro



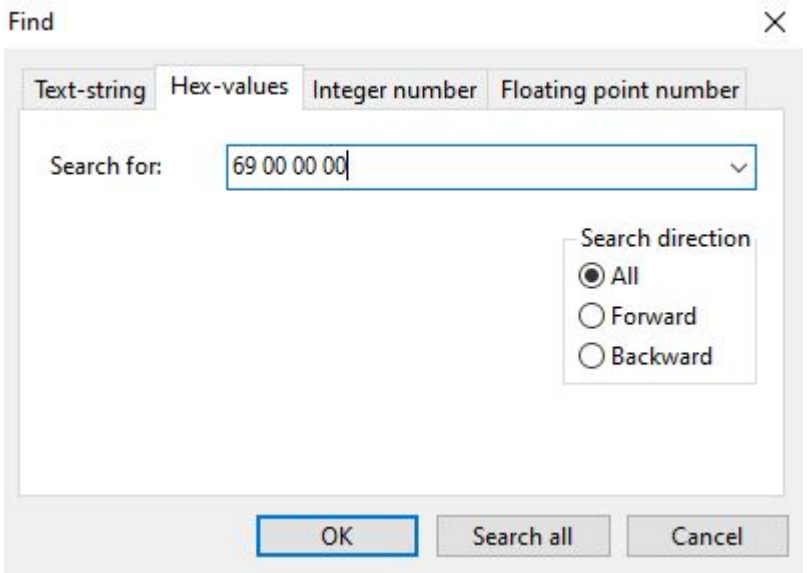
Usando o HxD - Busca por inteiro



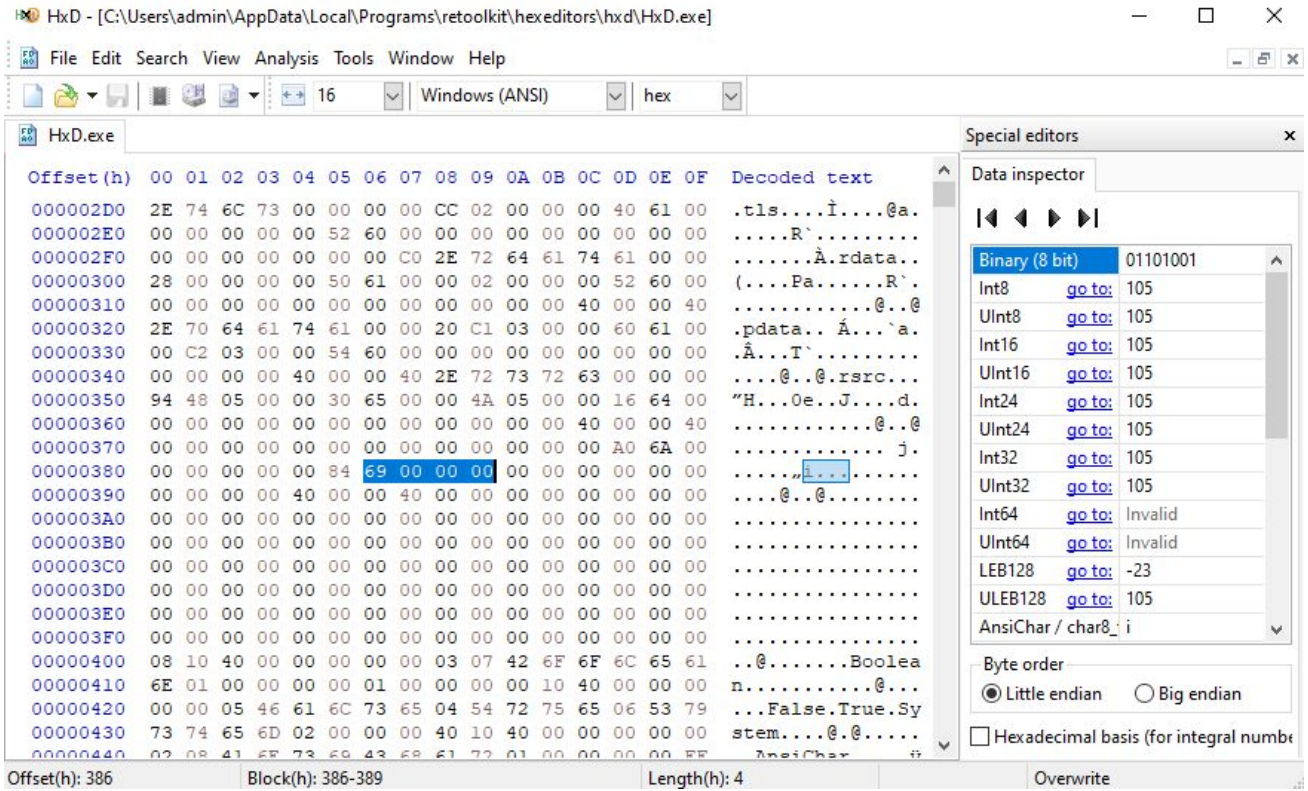
Usando o HxD - Busca por inteiro



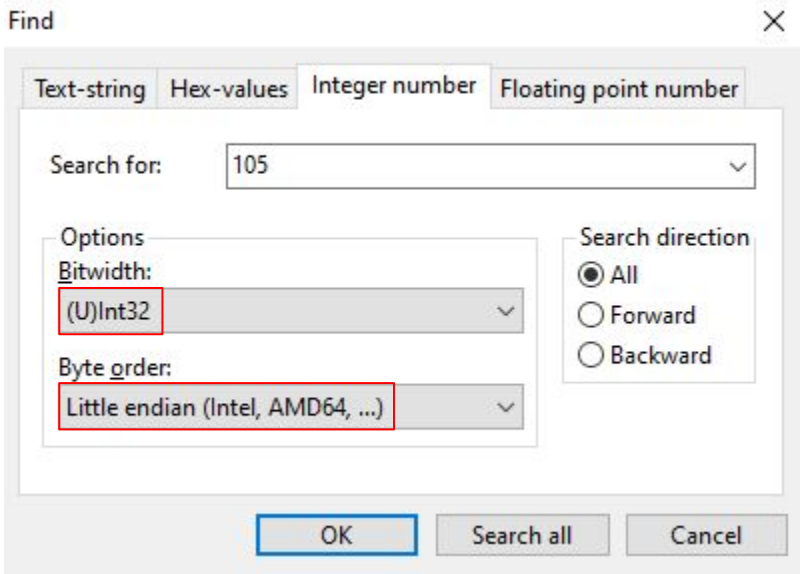
Usando o HxD - Busca por inteiro



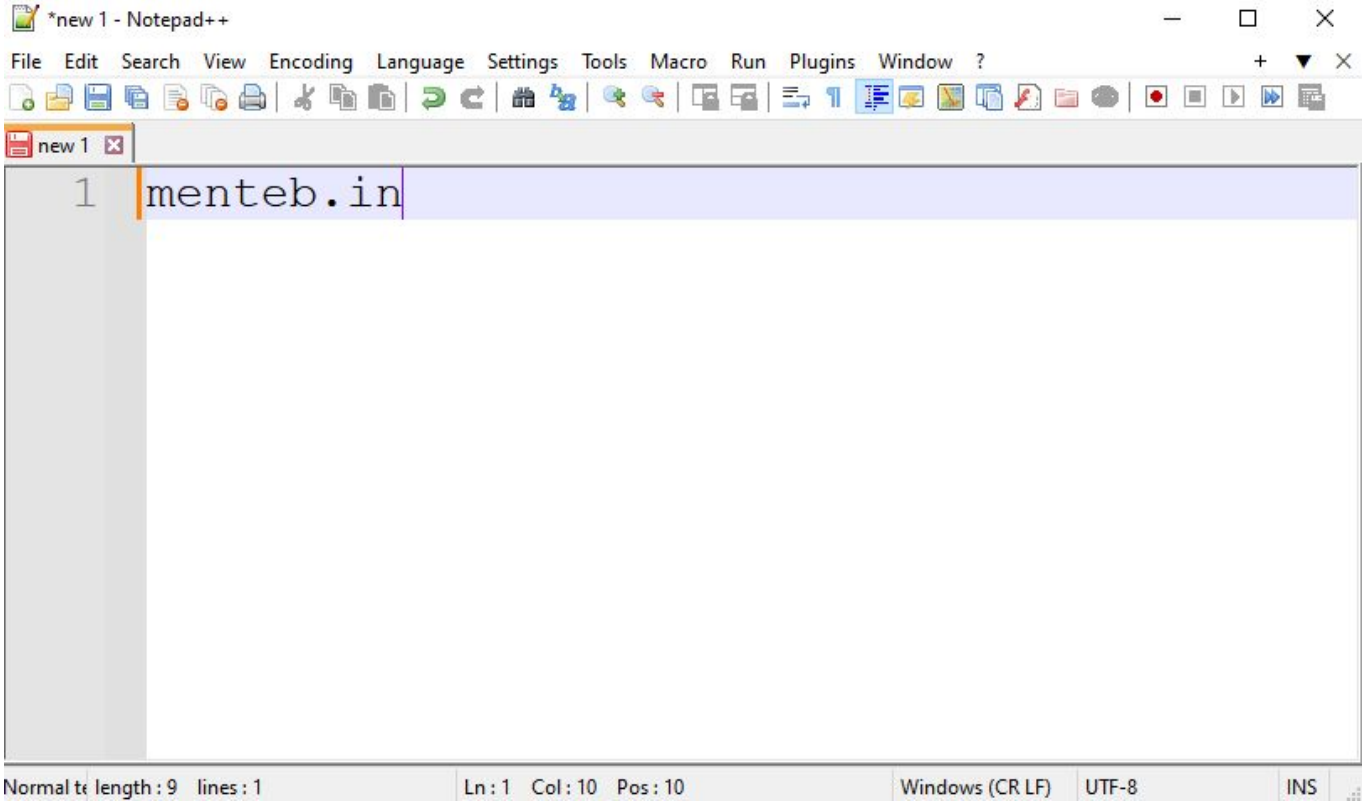
Usando o HxD - Busca por inteiro



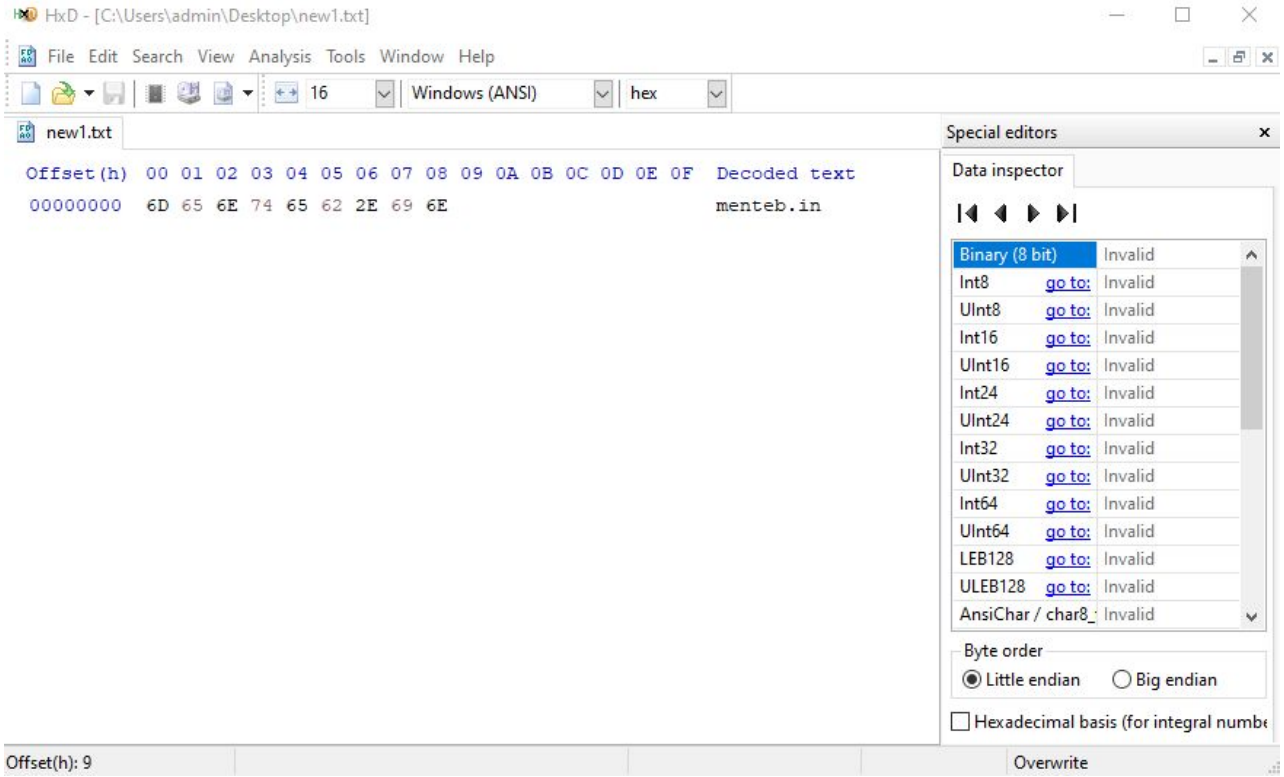
Usando o HxD - Busca por inteiro



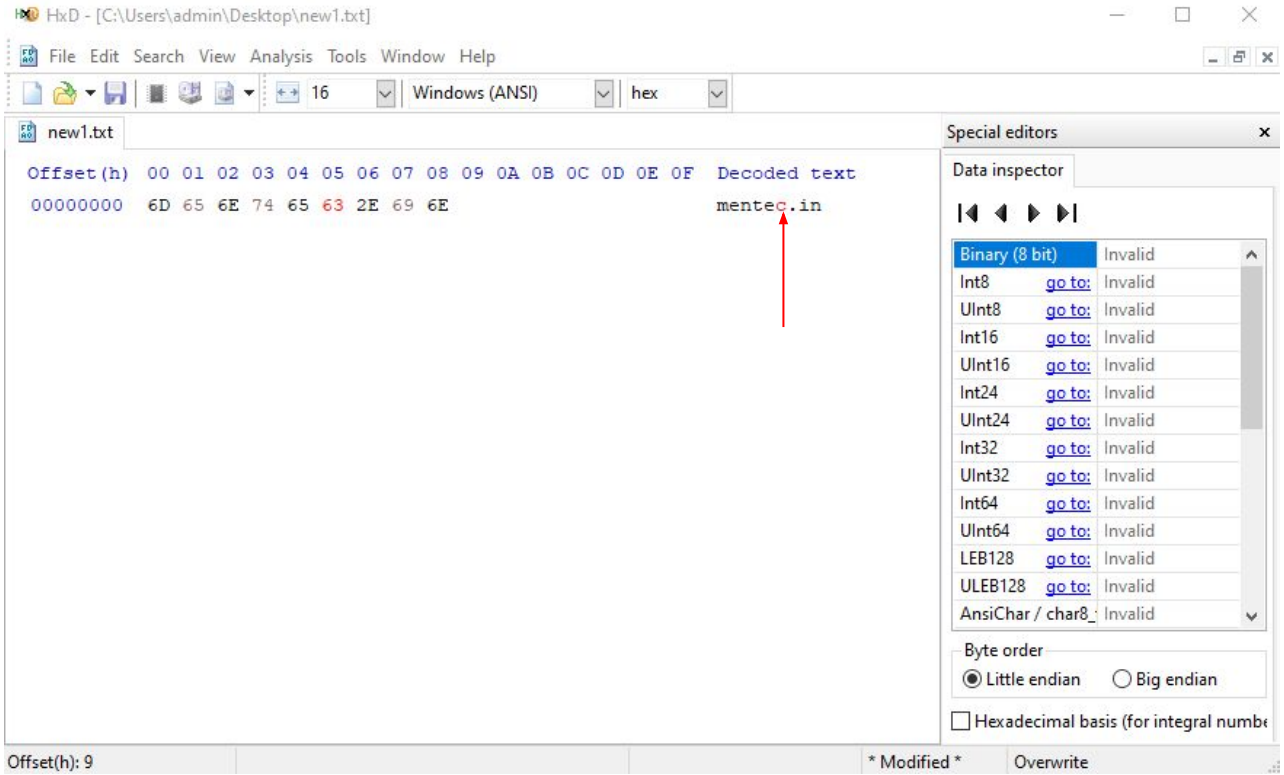
Usando o HxD - Editar e salvar



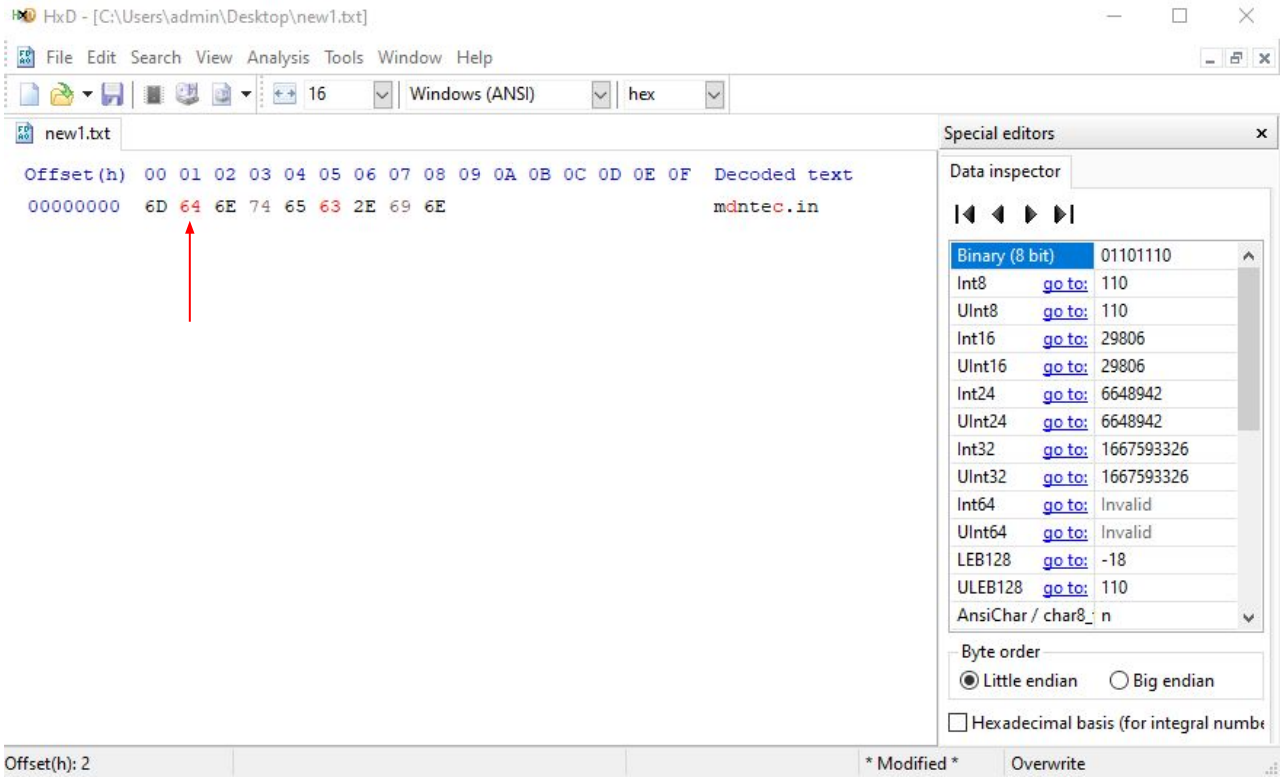
Usando o HxD - Editar e salvar



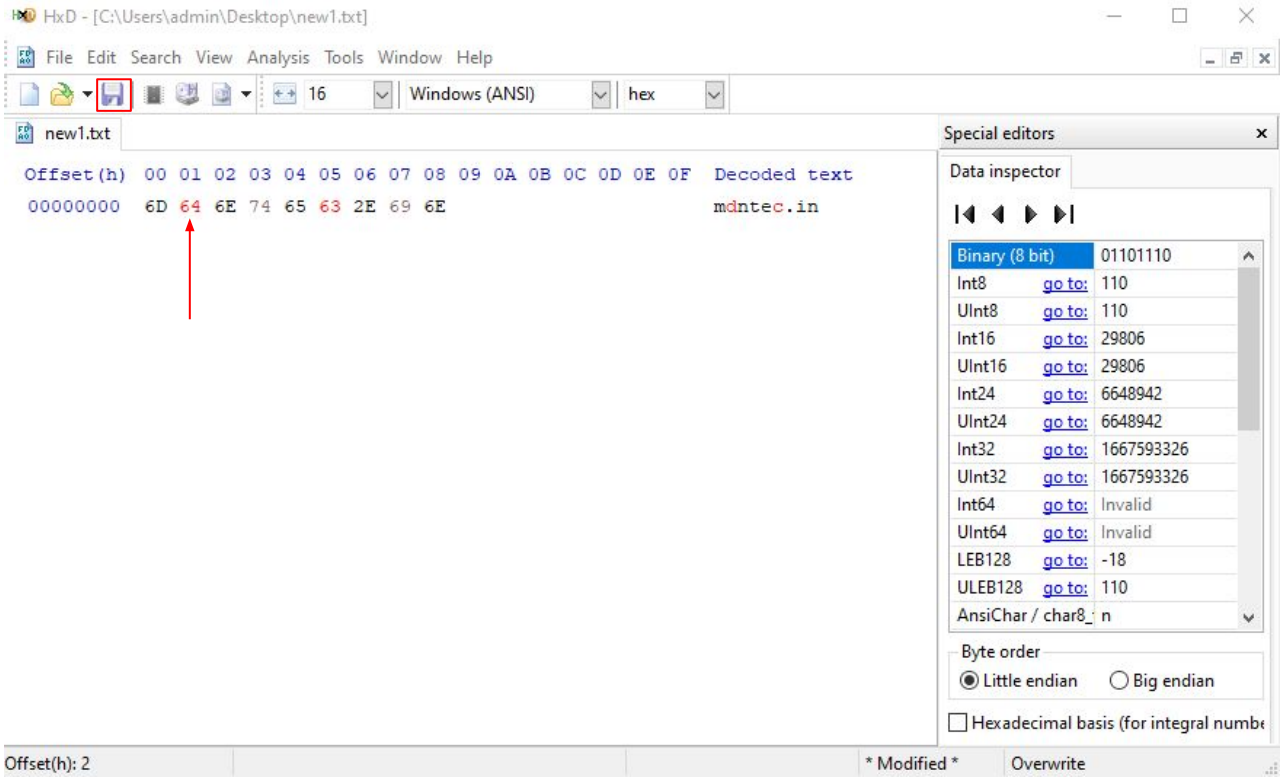
Usando o HxD - Editar e salvar



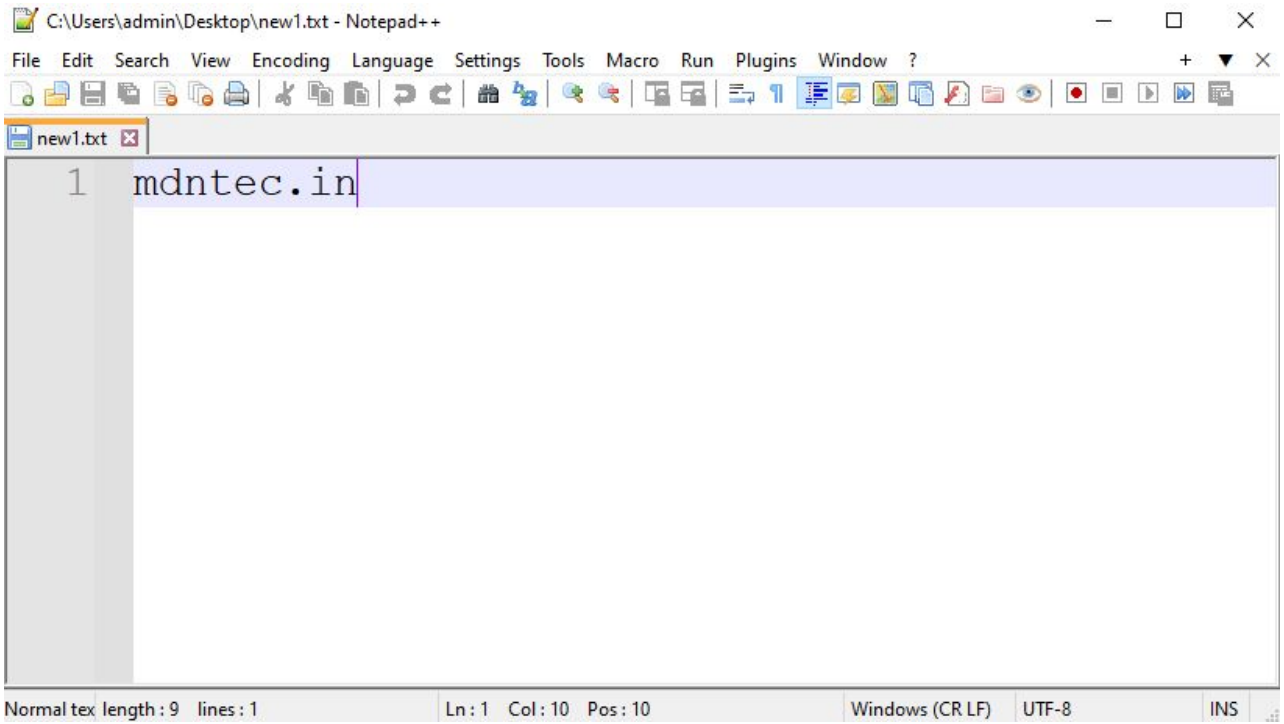
Usando o HxD - Editar e salvar



Usando o HxD - Editar e salvar



Usando o HxD - Editar e salvar



Lab 05

Manipulando strings

Lab 05 - Manipulando strings

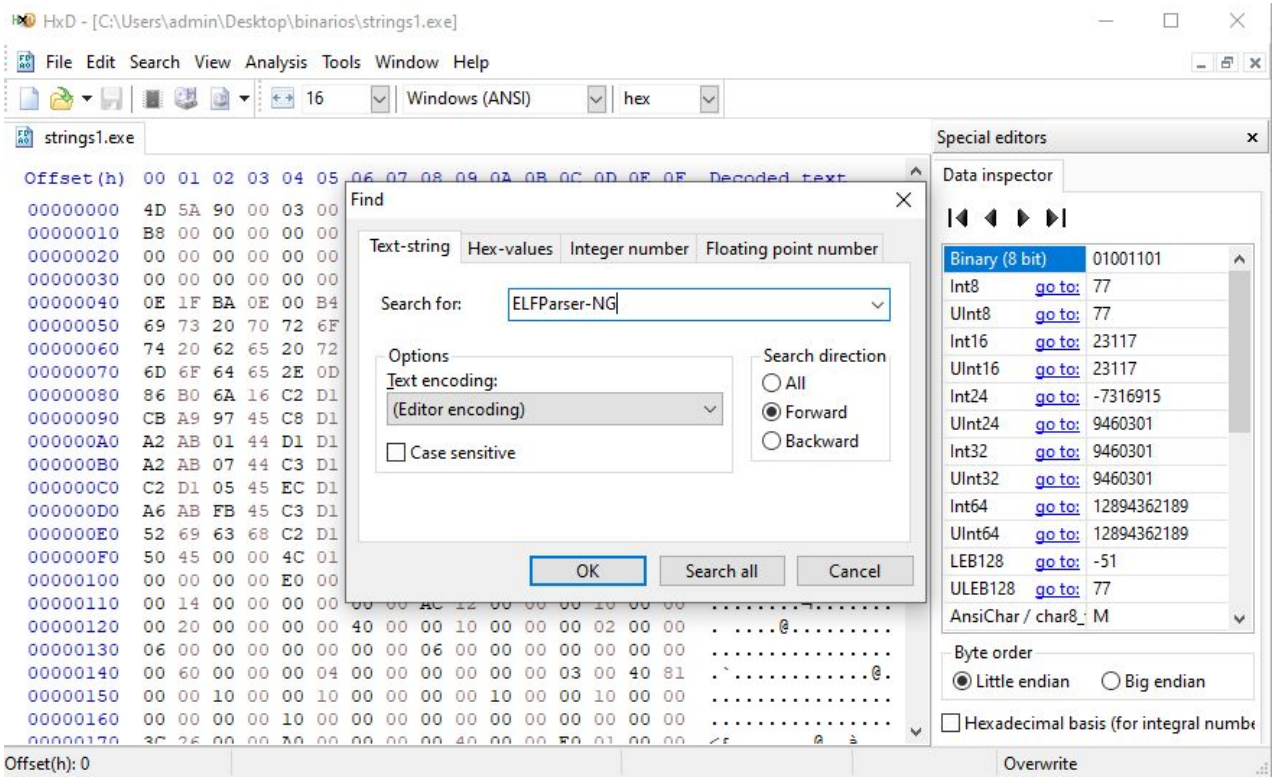
1. Abra o programa ***strings1.exe*** no HxD e responda:

a) Em que offset está a string “ELFParser-NG”?
b) Que tipo de string é esta?

2. Execute o programa ***strings2.exe*** e veja a string que ele exibe. Agora feche-o, abra-o no HxD e altere a string de forma que o programa exiba o seu primeiro nome (respeite o limite de caracteres da string atual).

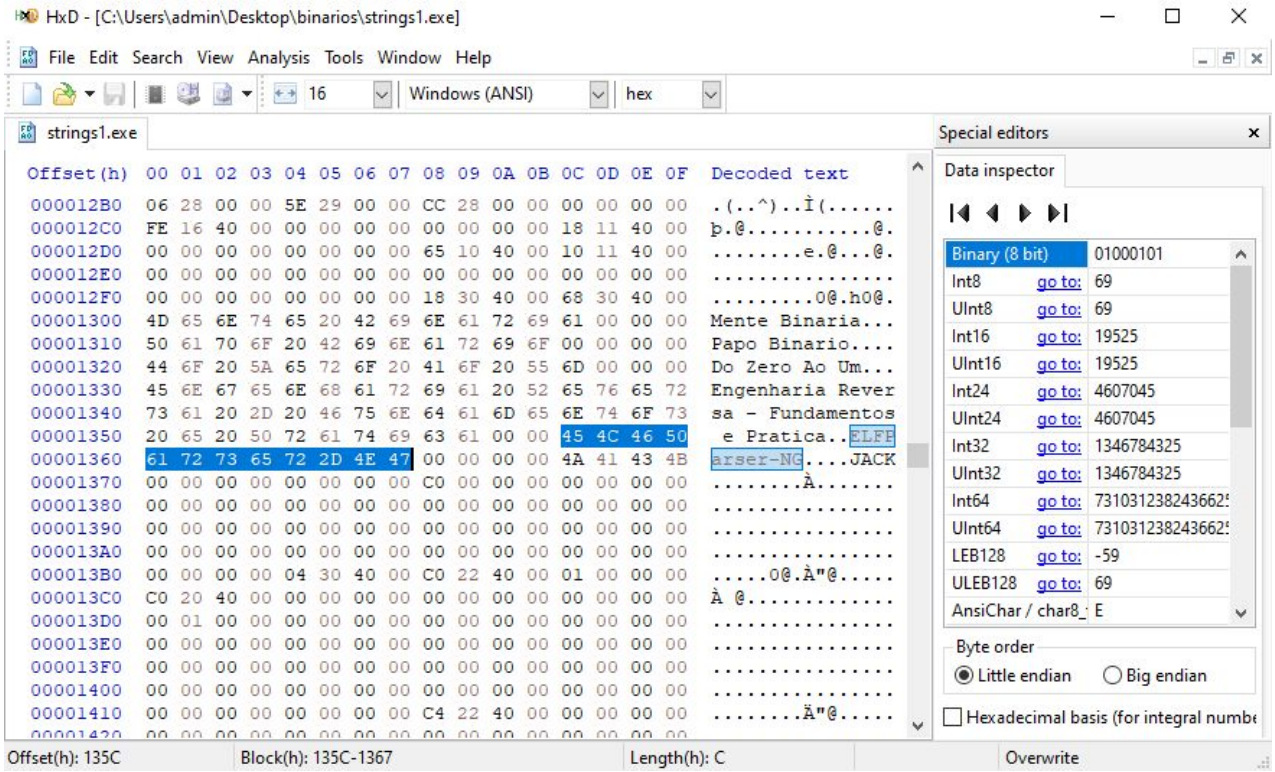
Lab 05 - Manipulando strings - Respostas

1.



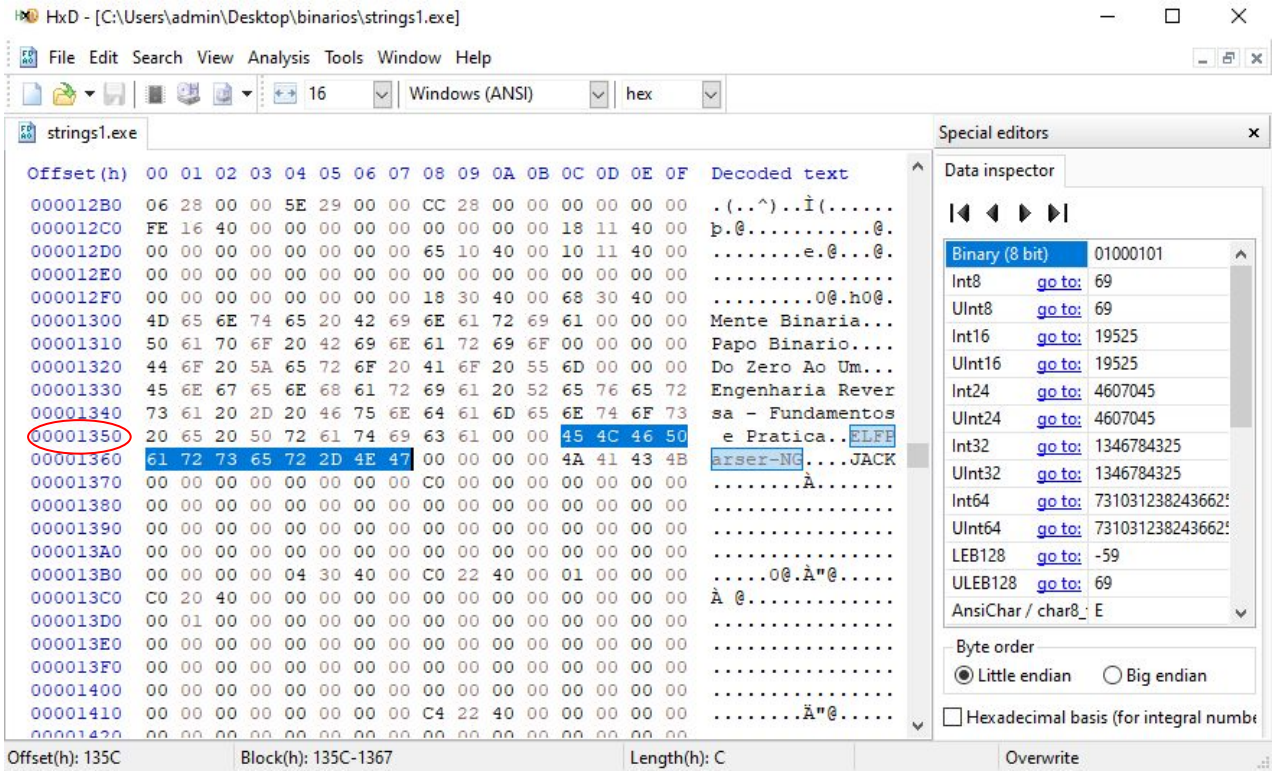
Lab 05 - Manipulando strings - Respostas

1.



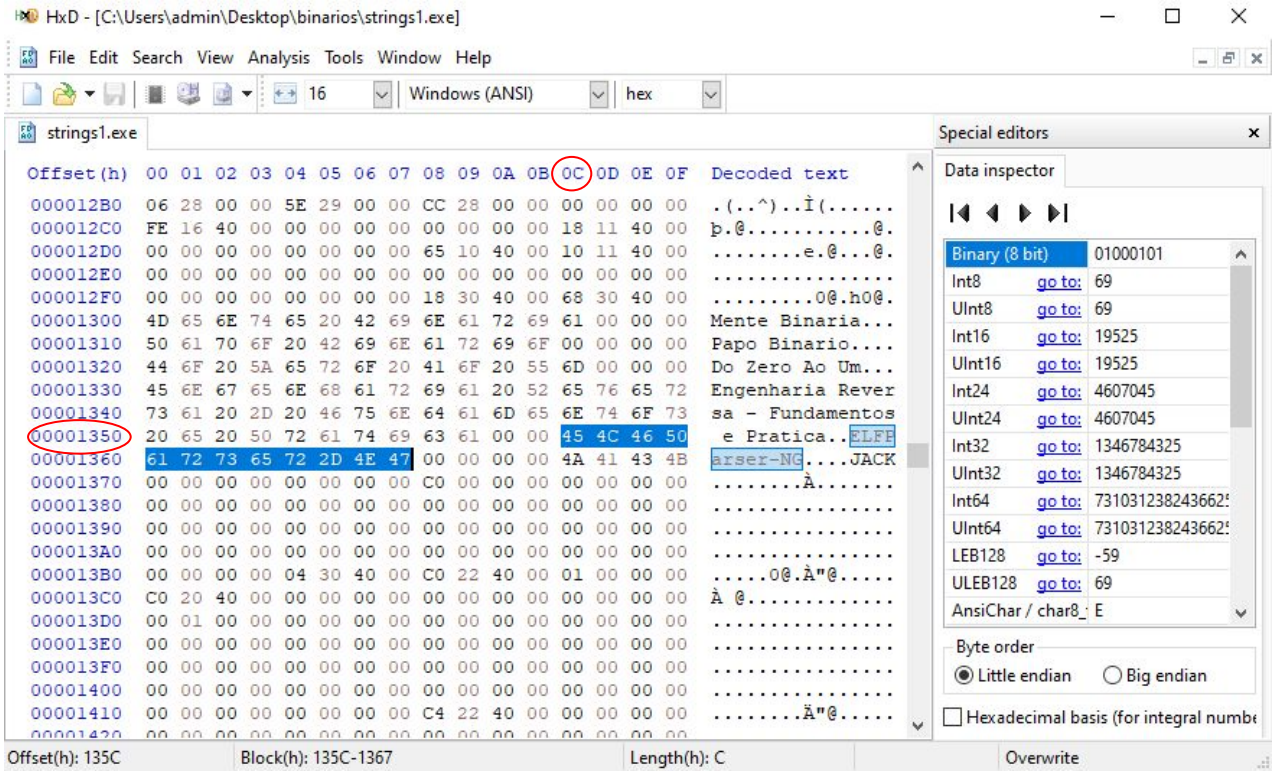
Lab 05 - Manipulando strings - Respostas

1.



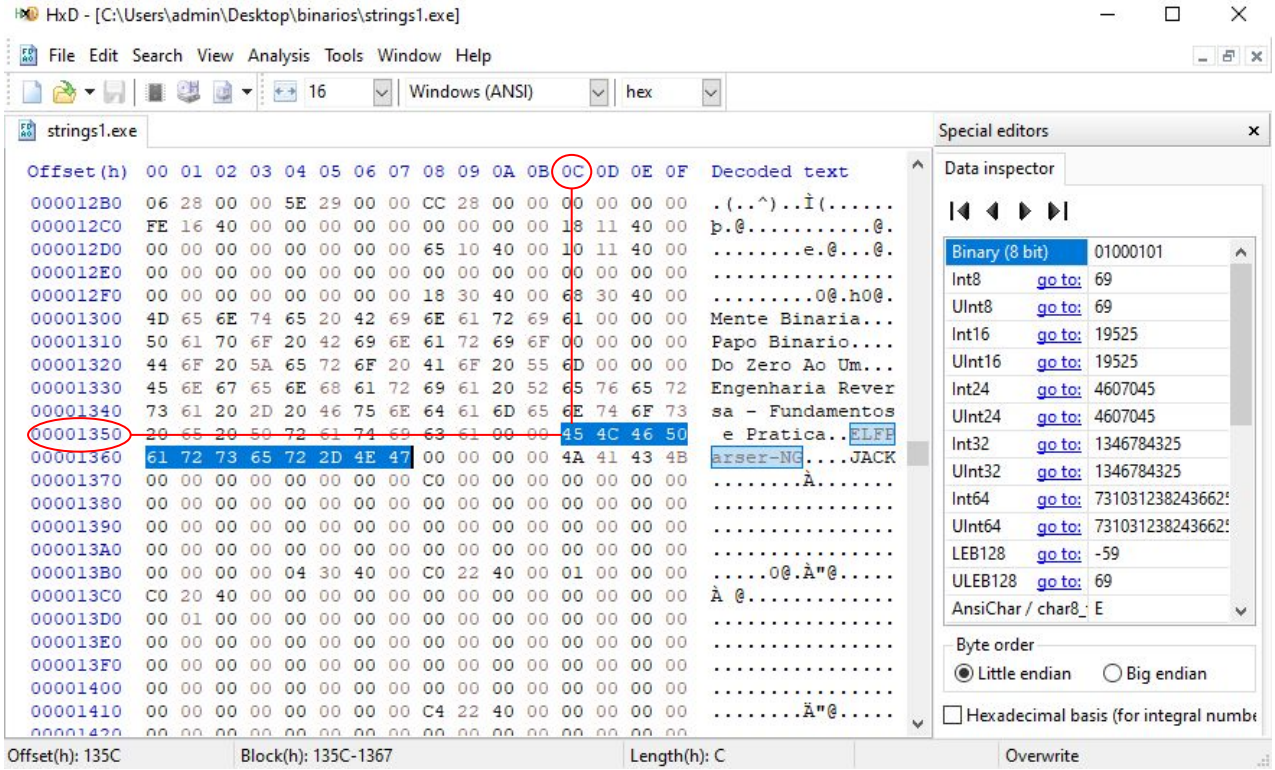
Lab 05 - Manipulando strings - Respostas

1.

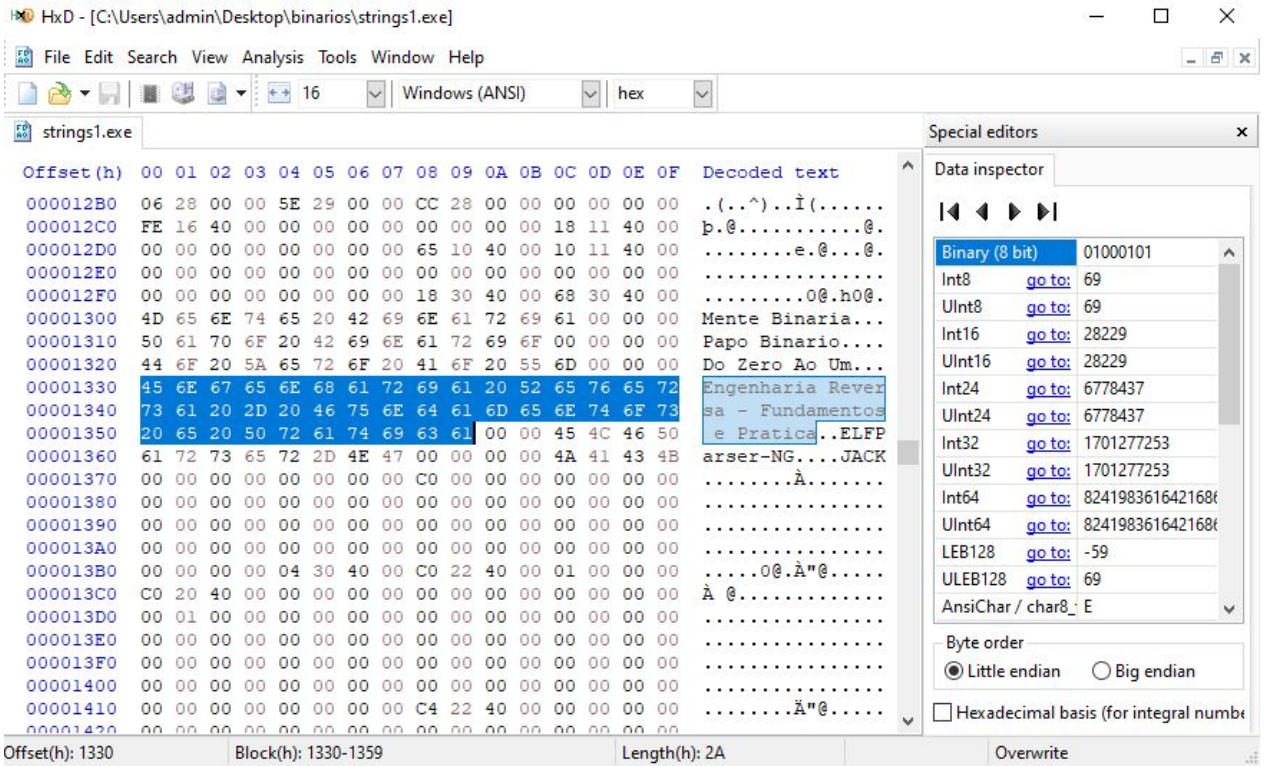


Lab 05 - Manipulando strings - Respostas

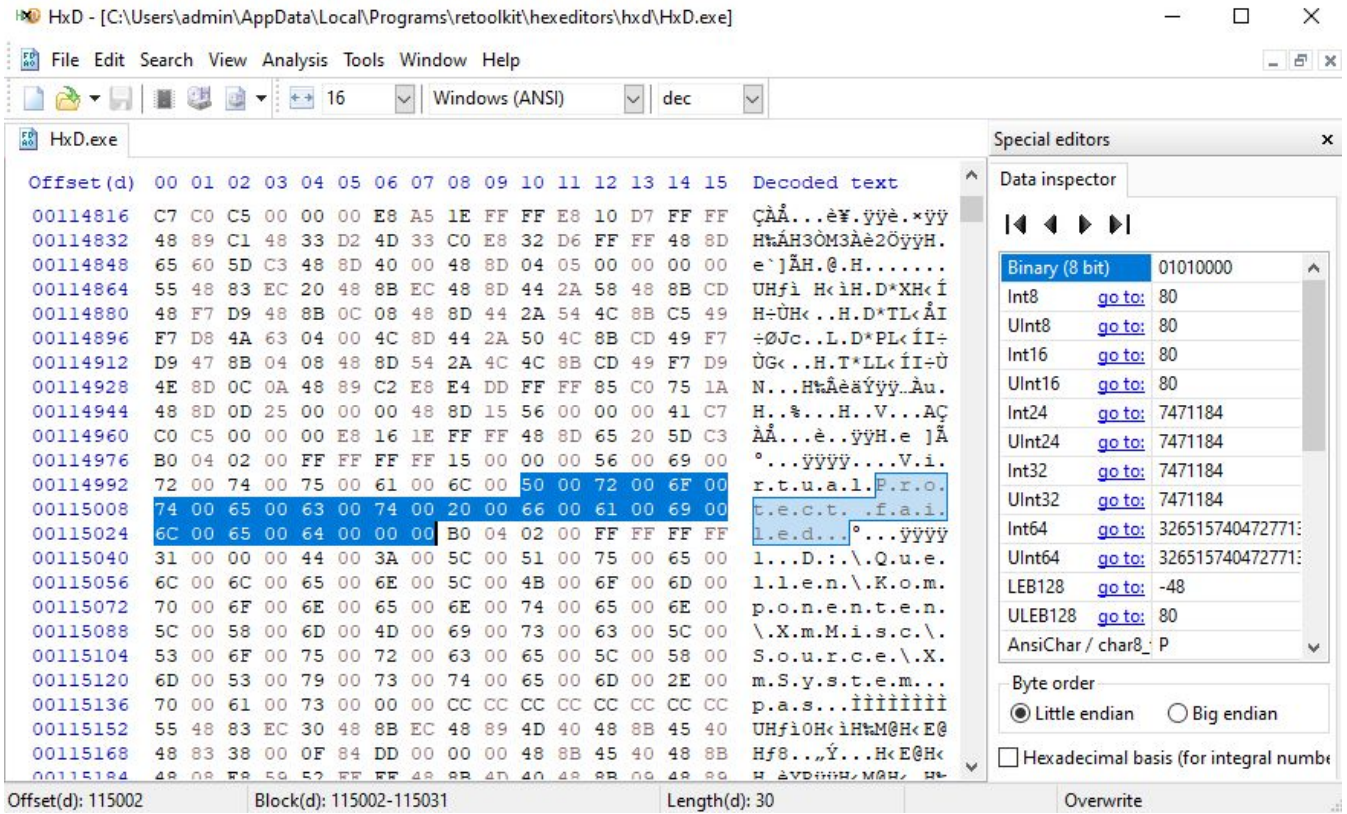
1.



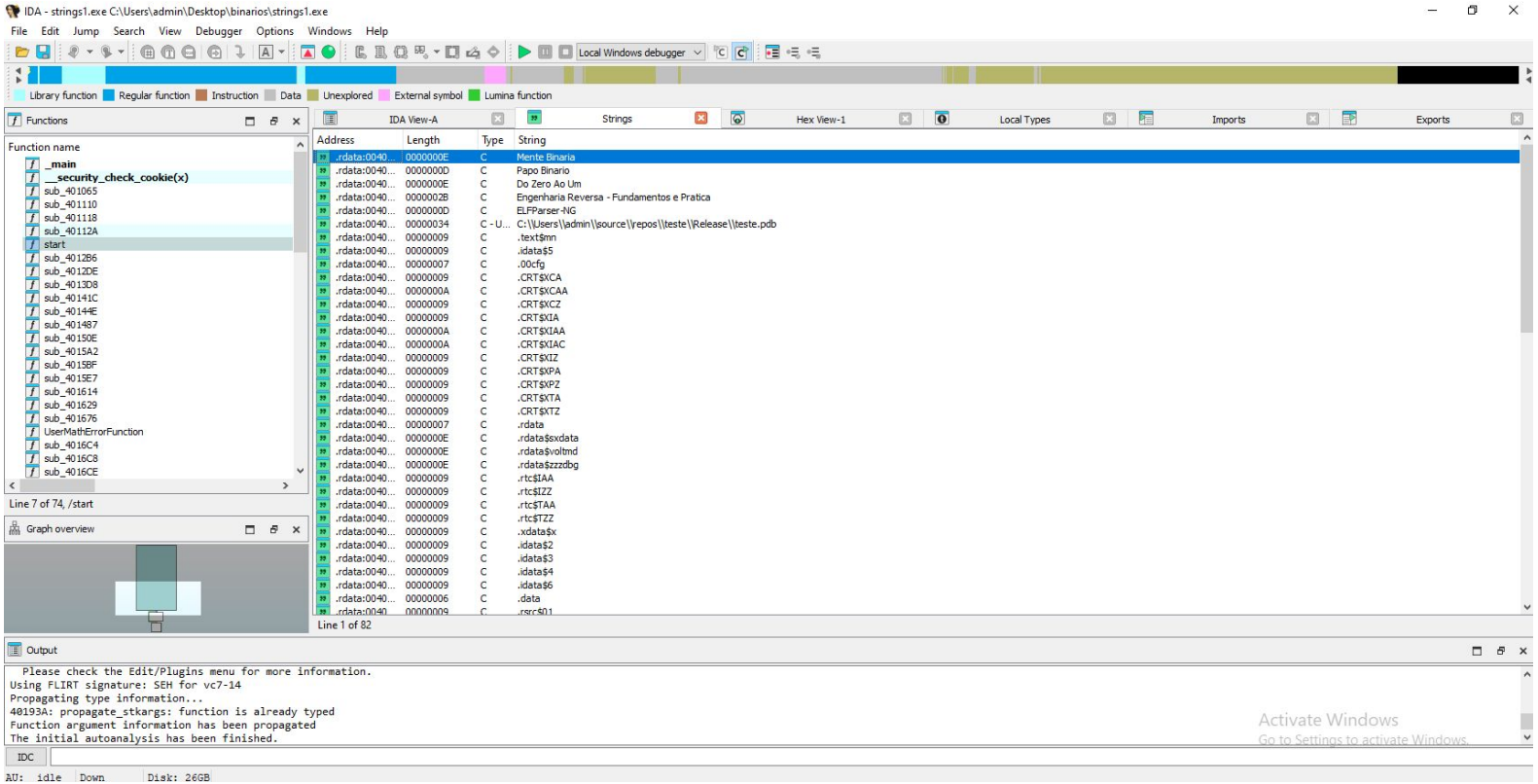
Lab 05 - Manipulando strings - Extra



Lab 05 - Manipulando strings - Extra

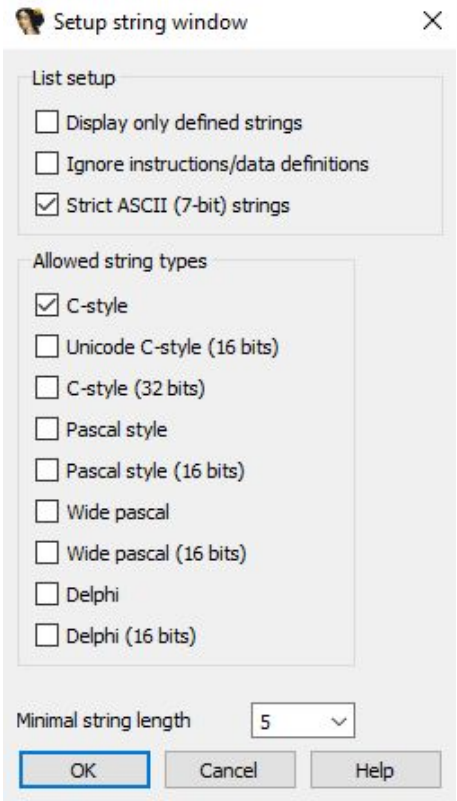


IDA - Manipulando strings - Extra



IDA - Manipulando strings - Extra

Address	Length	Type	String
.rdata:0040...	0000000E	C	Mente Binaria
.rdata:0040...	0000000D	C	Papo Binario
.rdata:0040...	0000000E	C	Do Zero Ao Um
.rdata:0040...	0000002B	C	Engenharia Reversa - Fundamentos e Pratica
.rdata:0040...	0000000D	C	ELFParser-NG
.rdata:0040...	00000034	C - U...	C:\\Users\\admin\\source\\repos\\teste\\Release\\teste.pdb
.rdata:0040...	00000009	C	.text$mn
.rdata:0040...	00000009	C	.idata$5
.rdata:0040...	00000007	C	.00cfg
.rdata:0040...	00000009	C	.CRT$XCA
.rdata:0040...	0000000A	C	.CRT$XCAA
.rdata:0040...	00000009	C	.CRT$XCZ
.rdata:0040...	00000009	C	.CRT$XIA
.rdata:0040...	0000000A	C	.CRT$XIAA
.rdata:0040...	0000000A	C	.CRT$XIAC
.rdata:0040...	00000009	C	.CRT$XIZ
.rdata:0040...	00000009	C	.CRT$XPA
.rdata:0040...	00000009	C	.CRT$XPZ
.rdata:0040...	00000009	C	.CRT$XTA
.rdata:0040...	00000009	C	.CRT$XTZ
.rdata:0040...	00000007	C	.rdata
.rdata:0040...	0000000E	C	.rdata$sxdata
.rdata:0040...	0000000E	C	.rdata$voltmd
.rdata:0040...	0000000E	C	.rdata$zzzdbg
.rdata:0040...	00000009	C	.rtc$IAA
.rdata:0040...	00000009	C	.rtc$IZZ
.rdata:0040...	00000009	C	.rtc$TAA
.rdata:0040...	00000009	C	.rtc$TZZ
.rdata:0040...	00000009	C	.xdata$x
.rdata:0040...	00000009	C	.idata$2
.rdata:0040...	00000009	C	.idata$3
.rdata:0040...	00000009	C	.idata$4
.rdata:0040...	00000009	C	.idata$6
.rdata:0040...	00000006	C	.data

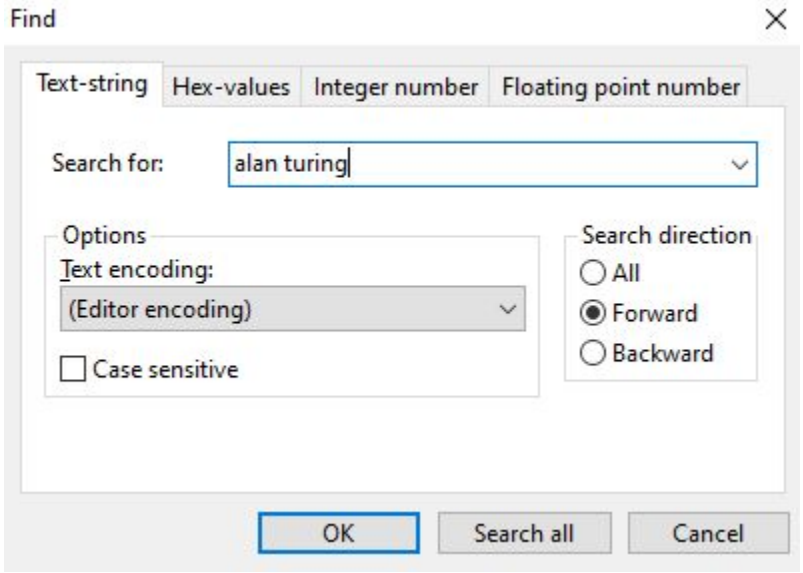


Lab 05 - Manipulando strings - Respostas

2.

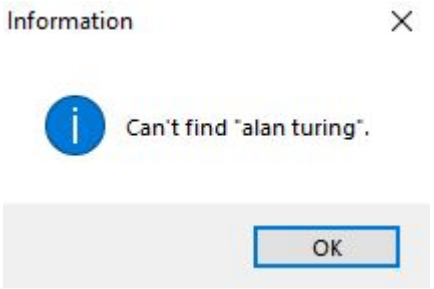
Lab 05 - Manipulando strings - Respostas

2.



Lab 05 - Manipulando strings - Respostas

2.

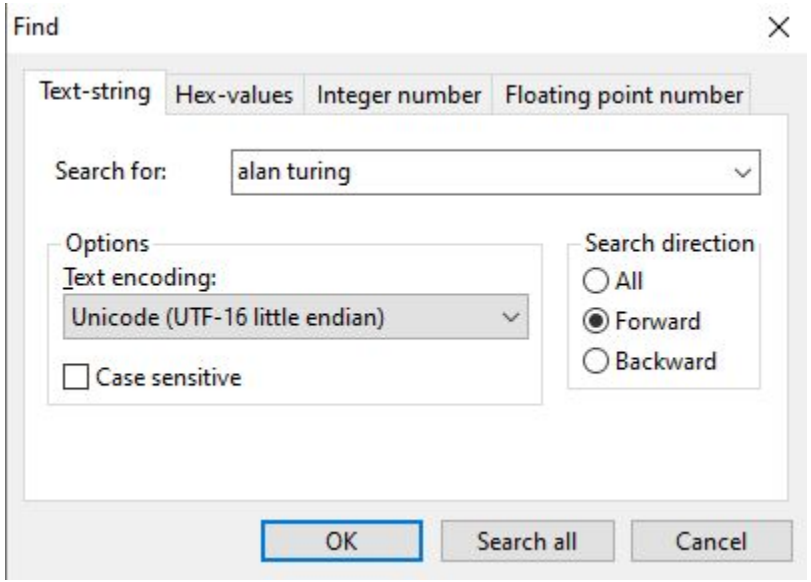


Lab 05 - Manipulando strings - Respostas

2.

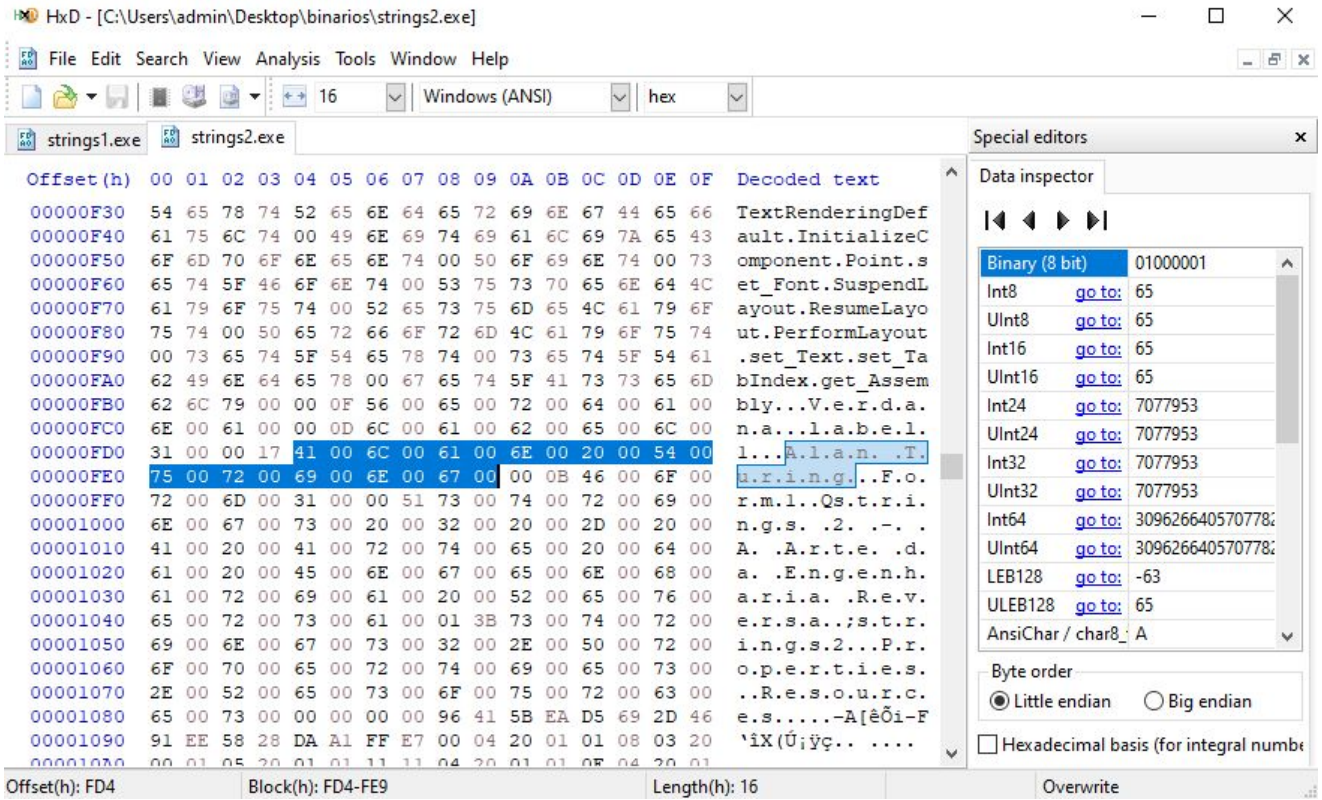
Lab 05 - Manipulando strings - Respostas

2.



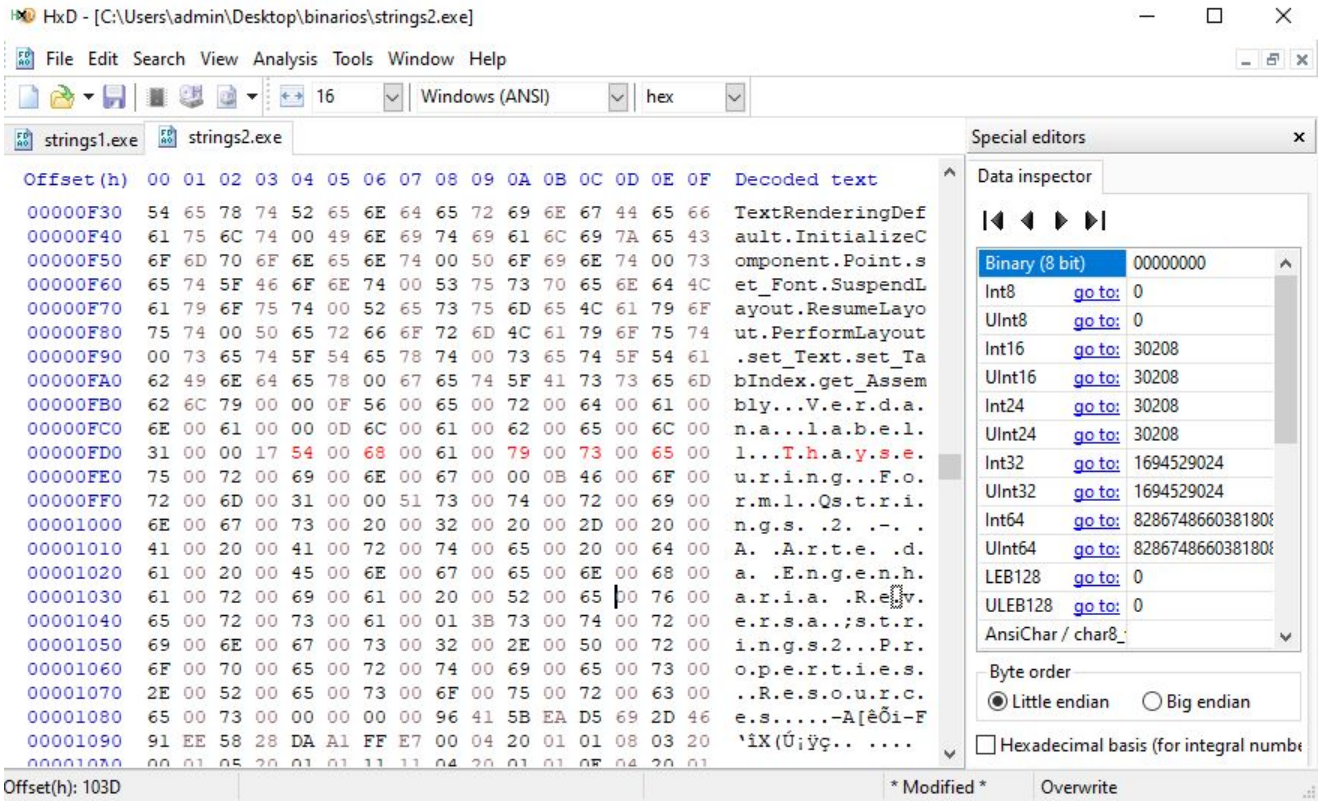
Lab 05 - Manipulando strings - Respostas

2.



Lab 05 - Manipulando strings - Respostas

2.

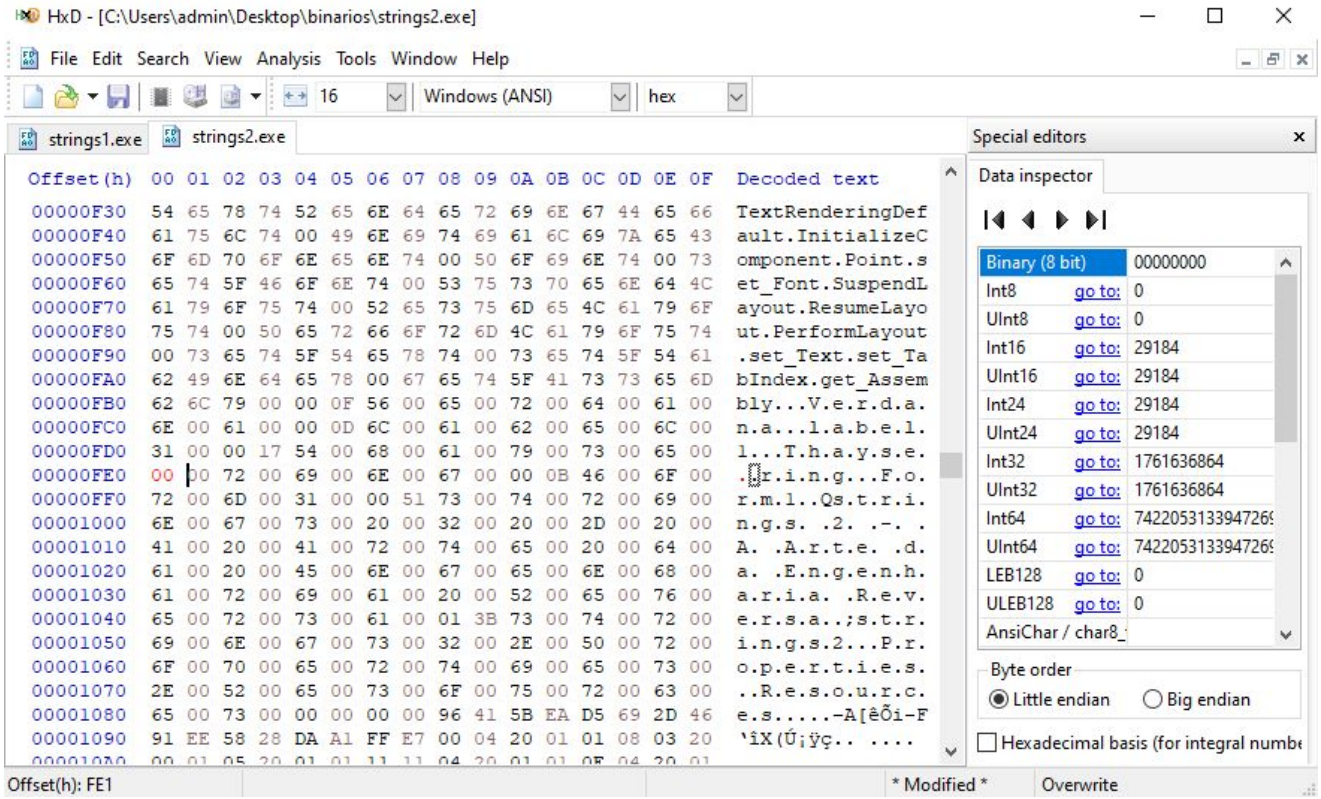


Lab 05 - Manipulando strings - Respostas

2.

Lab 05 - Manipulando strings - Respostas

2.



Lab 05 - Manipulando strings - Respostas

2.