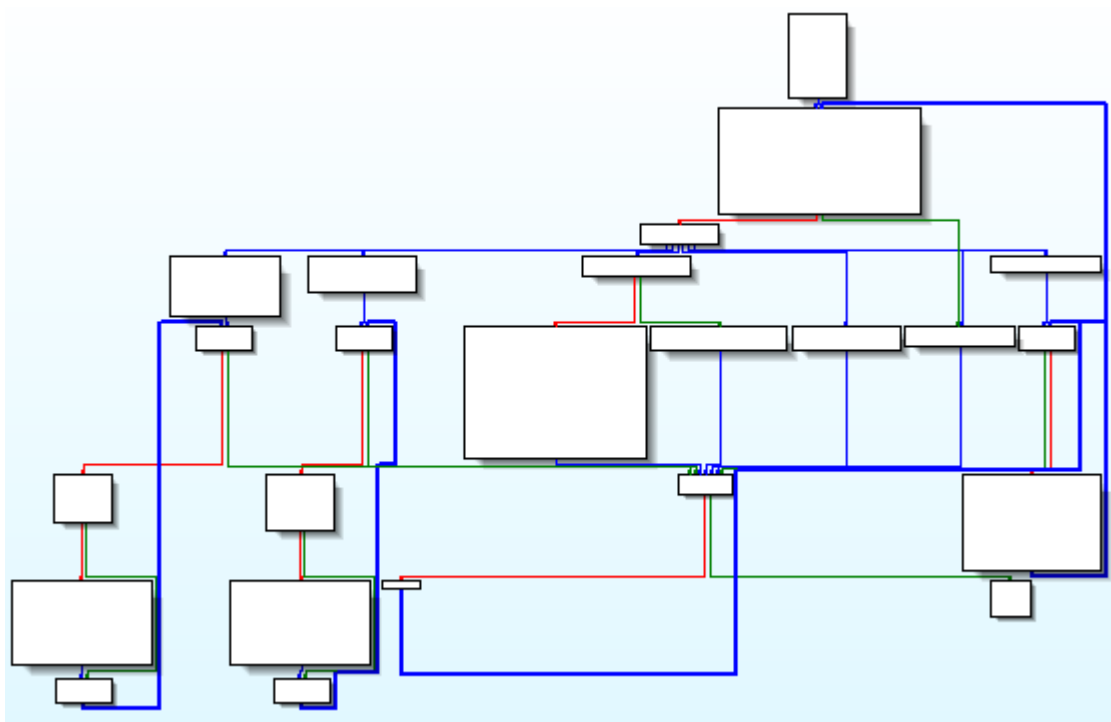


Solucion ejercicio 9, veremos que si lo encaramos de la forma correcta no se pone tan cuesta arriba, lo primero seria definir las variables y datos que maneja, para luego comenzar a analizarlo.

vamos a ver con que nos encontramos



Esto es lo que tenemos que reversear, después de jugar un rato se van a dar cuenta de que no es tan grandota como aparenta y hay mucho código repetido, así que va a ir rápida la cosa.

Vallamos al comienzo de la función y le ponemos el nombre de rigor



```
; Attributes: bp-based frame
```

```
funcion_investigada proc near
```

```
var_BBCC= dword ptr -0BBCCCh
var_BBC8= byte ptr -0BBC8h
var_BB94= dword ptr -0BB94h
var_BB90= dword ptr -0BB90h
var_BB8C= dword ptr -0BB8Ch
var_BB88= byte ptr -0BB88h
var_8= byte ptr -8
```

```
push    ebp
mov     ebp, esp
push    ebx
mov     eax, 0BBE4h
call    ___chkstk
mov     [ebp+var_BB8C], 0
```

Lo primero que les tiene que llamar la atención en este caso es la cantidad de memoria que usa el programa, esto se puede deber a una cosa... que use una estructura de datos que pide memoria estáticamente. Es decir nuestro querido “array de struct” ☺

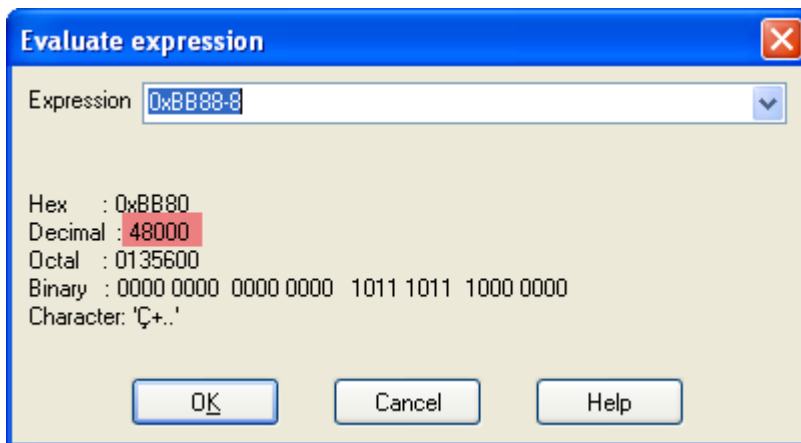
Veamos el stack-frame para ver cómo están acomodadas las variables

| | | |
|----------|----------|------------------|
| 0000BB94 | var_BB94 | dd ? |
| 0000BB90 | var_BB90 | dd ? |
| 0000BB8C | var_BB8C | dd ? |
| 0000BB88 | var_BB88 | db ? |
| 0000BB87 | | db ? ; undefined |
| 0000BB86 | | db ? ; undefined |
| 0000BB85 | | db ? ; undefined |
| 0000BB84 | | db ? ; undefined |
| 0000BB83 | | db ? ; undefined |
| 0000BB82 | | db ? ; undefined |
| 0000BB81 | | db ? ; undefined |
| 0000BB80 | | db ? ; undefined |
| 0000BB7F | | db ? ; undefined |
| 0000BB7E | | db ? ; undefined |
| 0000BB7D | | db ? ; undefined |
| 0000BB7C | | db ? ; undefined |
| 0000BB7B | | db ? ; undefined |
| 0000BB7A | | db ? ; undefined |
| 0000BB79 | | db ? ; undefined |
| 0000BB78 | | db ? ; undefined |
| 0000BB77 | | db ? ; undefined |
| 0000BB76 | | db ? ; undefined |
| 0000BB75 | | db ? ; undefined |
| 0000BB74 | | db ? ; undefined |
| 0000BB73 | | db ? ; undefined |
| 0000BB72 | | db ? ; undefined |
| 0000BB71 | | db ? ; undefined |
| 0000BB70 | | db ? ; undefined |

La marcada es la que nos tiene que llamar la atención, miremos cuanto ocupa restando la dirección del stack de el BB88 con la de la anterior variable.

```
00000000C      db ? ; undefined
00000000B      db ? ; undefined
00000000A      db ? ; undefined
000000009      db ? ; undefined
000000008  var_8 db ?
000000007      db ? ; undefined
000000006      db ? ; undefined
000000005      db ? ; undefined
000000004      db ? ; undefined
000000003      db ? ; undefined
000000002      db ? ; undefined
000000001      db ? ; undefined
000000000      db 4 dup(?)
000000004      r      db 4 dup(?)
000000008
000000008 ; end of stack variables
```

Esa que marque sigue siendo parte de la grandota, así que los restamos



48000 bytes, eso es bastante para una simple variable ¿no? Después veremos que es esta porción de memoria, por ahora sigamos mirando el código.



```
mov     dword ptr [esp], offset aIntroduceElNom ; "Introduce el nombre del fichero:"
call    printf
lea     ecx, [ebp+var_BB88] ; guarda la direccion de la estructura
mov     edx, [ebp+var_BB8C]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
mov     [esp], eax ; char *
call    gets
mov     dword ptr [esp], offset aIntroduceElTam ; "Introduce el tama"
call    printf
lea     eax, [ebp+var_BB88]
mov     [esp], eax ; char *
call    gets
lea     ecx, [ebp+var_BB88]
mov     edx, [ebp+var_BB8C]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
add     eax, 2Ch
mov     [esp+8], eax
mov     dword ptr [esp+4], offset aId ; "%ld"
lea     eax, [ebp+var_BB88]
mov     [esp], eax ; char *
call    scanf
lea     eax, [ebp+var_BB8C]
inc     dword ptr [eax]
jmp     loc_401626
```

Tratemos de interpretar que hace este código así poco a poco vamos conociendo las variables que entran en juego.

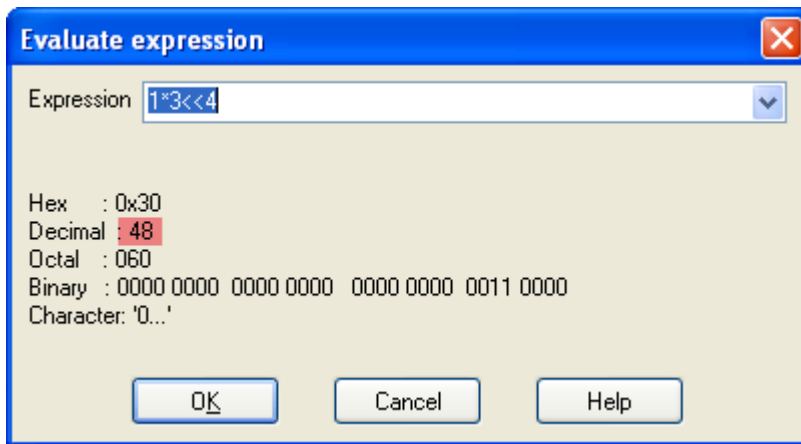


```
mov     dword ptr [esp], offset aIntroduceElNom ; "Introduce el nombre del
call    printf
lea     ecx, [ebp+var_BB88] ; guarda la direccion de la estructura
mov     edx, [ebp+var_BB8C]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
mov     [esp], eax ; char *
call    gets
```

Todos esos cálculos a partir del valor de var_BB8C generan un offset de la forma $48n$

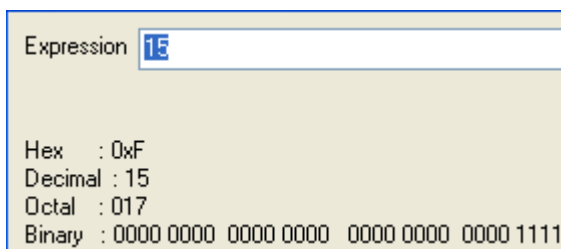
Siendo n un número de la forma $[0, +\infty) \in \mathbb{Z}$ o mejor dicho un entero positivo

bue, recuerden que lo de más infinito es relativo ya que nuestra memoria tiene límite ☺, me refiero a que va desde cero para arriba.

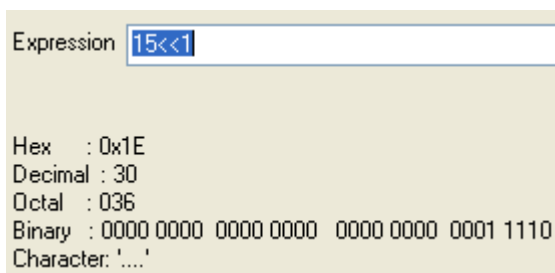


En este caso $n = 1 \therefore 48.1 = 48$ pero uno se preguntara como el compilador llega a esa clase de matemática esotérica con corrimientos. La respuesta es sencilla, y aunque no lo crean se debe a que la computadora es un poco vaga y quiere hacer la menor cantidad de operaciones posibles para dar un resultado.

Un corrimiento a la izquierda es una multiplicación por dos (porque están en base 2) si a un numero en base 10 yo le hago un corrimiento a la izquierda lograría una multiplicación por 10 (porque esta en base 10). Un ejemplo vale mas que mil palabras.



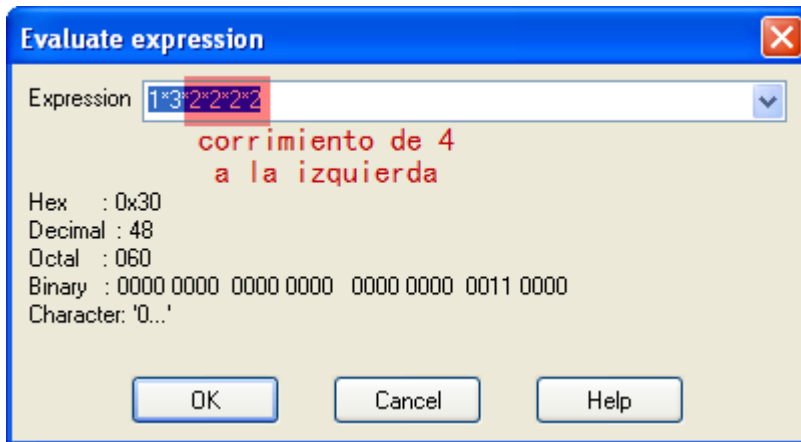
y 15 con un corrimiento



Vemos que en uno da 15 y en el otro 30, es una multiplicación por 2, y como a la computadora le es menos cantidad de pasos hacer un corrimiento que una multiplicación, se termina decidiendo por meter las multiplicaciones de potencias de 2 como corrimientos.

Esto tal vez es difícil de digerir para alguien que no sabe nada de matemáticas, pero donde se pongan un rato se dan cuenta que es una taradez

Volviendo a nuestro código podríamos pasar nuestro corrimiento a una multiplicación humanita



Y aplicando la super algebra, llegamos a la brillante deducción que...

$$n \cdot 3 \cdot 2^4 = n \cdot 3 \cdot 16 = n \cdot 48 = 48n$$

Ese valor es lo que le suma a la dirección de la estructura para llegar a los valor en base a n , es decir que esta es un índice para accesar el array. Ya sabemos algo... el tamaño de cada elemento es de 48 bytes, ahora hay que ver cuantos le pueden entrar.

Si recordamos un poquito (o miramos 3 paginas atrás) vemos que la estructura de datos tenia un largo de 48000 bytes, por lo tanto no hay que ser Einstein para hacer

$$\frac{48000}{48} = 1000$$

1000 serian la cantidad de elementos struct (cada uno de 48 bytes) que tendría el array. Miremos las string's para ver si vemos algo que apruebe nuestra hipótesis ☺

| Address | Length | T... | String |
|---------------------|----------|------|--------------------------------------------------------|
| "..." rdata:0040... | 0000000F | C | Escoja una opci |
| "..." rdata:0040... | 00000005 | C | 1.- A |
| "..." rdata:0040... | 00000020 | C | adir datos de un nuevo fichero\n |
| "..." rdata:0040... | 0000002F | C | 2.- Mostrar los nombres de todos los ficheros\n |
| "..." rdata:0040... | 00000036 | C | 3.- Mostrar ficheros que sean de mas de un cierto tama |
| "..." rdata:0040... | 0000001D | C | 4.- Ver datos de un fichero\n |
| "..." rdata:0040... | 00000008 | C | 5.- Salir\n |
| "..." rdata:0040... | 00000022 | C | Introduce el nombre del fichero: |
| "..." rdata:0040... | 00000011 | C | Introduce el tama |
| "..." rdata:0040... | 0000000A | C | o en KB: |
| "..." rdata:0040... | 00000024 | C | MBximo de fichas alcanzado (1000)!\n |
| "..." rdata:0040... | 00000010 | C | Nombre: %s; Tama |
| "..." rdata:0040... | 00000008 | C | o: %ld Kb\n |
| "..." rdata:0040... | 00000015 | C | +A partir de que tama |
| "..." rdata:0040... | 0000001A | C | o quieres que te muestre? |
| "..." rdata:0040... | 0000002D | C | +De qu  fichero quieres ver todos los datos? |
| "..." rdata:0040... | 00000012 | C | Fin del programa\n |
| "..." rdata:0040... | 00000010 | C | n desconocida!\n |
| "..." rdata:0040... | 0000002D | C | w32_sharedptr->size == sizeof(W32_EH_SHARED) |
| "..." rdata:0040... | 0000001E | C | %s:%u: failed assertion `%s'\n |
| "..." rdata:0040... | 0000002B | C | .../gcc/gcc/config/i386/w32-shared-ptr.c |
| "..." rdata:0040... | 00000027 | C | GetAtomNameA (atom, s, sizeof(s)) != 0 |

Estamos de suerte, hay algunas pistas por el camino.

En fin, esa parte del bloque que est bamos mirando guarda el nombre en un elemento de un array tipo estructura.

Miremos mas debajo de esas l neas

```

mov     dword ptr [esp], offset aIntroduceElTam ; "Introduce el tama"
call    printf
lea     eax, [ebp+var_BBC8]
mov     [esp], eax ; char *
call    gets
lea     ecx, [ebp+var_BB88]
mov     edx, [ebp+var_BB8C]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
add     eax, 44
mov     [esp+8], eax
mov     dword ptr [esp+4], offset aLd ; "%ld"
lea     eax, [ebp+var_BBC8]
mov     [esp], eax ; char *
call    sscanf
lea     eax, [ebp+var_BB8C]
inc     dword ptr [eax]
jmp     loc_401626

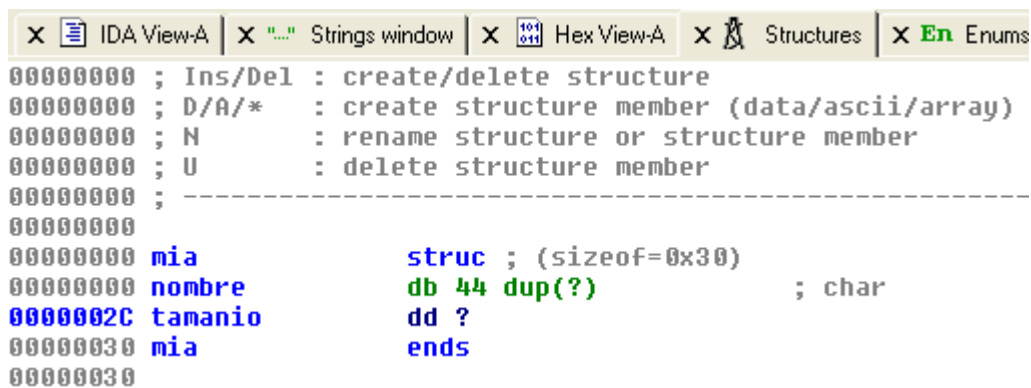
```

usa el mismo indice que antes, pero ahora usa un offset 44 para guardar un valor entero (fijense el %ld)

Esos 44 bytes que desplazo son los que le corresponden a nombre, es decir que nombre se inicializo con nombre[44] y como son 48 bytes en total por estructura... solo nos quedan 4 bytes para guardar un entero que representa el tamaño.

Fijense también que no lo hace directamente con gets porque este solo carga caracteres, para pasarlo a entero hace uso de sscanf que vendría a ser un scanf de toda la vida pero en vez de trabajar sobre la entrada estándar, trabaja con una variable.

Ya estamos en condiciones de hacer una de las partes mas complicadas del reto, definir la estructura



```

00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/* : create structure member (data/ascii/array)
00000000 ; N : rename structure or structure member
00000000 ; U : delete structure member
00000000 ; -----
00000000
00000000 mia          struc ; (sizeof=0x30)
00000000 nombre        db 44 dup(?) ; char
0000002C tamanio     dd ?
00000030 mia          ends
00000030

```

Y se la aplicamos a nuestra variable con "Alt+Q" definiéndola luego como array para que quede 1000

| Address | Variable | Type |
|-----------|--------------------------|------------------|
| -0000BB90 | var_BB90 | dd ? |
| -0000BB8C | var_BB8C | dd ? |
| -0000BB88 | ficha | mia 1000 dup(?) |
| -00000008 | var_8 | db ? |
| -00000007 | | db ? ; undefined |
| -00000006 | | db ? ; undefined |
| -00000005 | | db ? ; undefined |
| -00000004 | | db ? ; undefined |
| -00000003 | | db ? ; undefined |
| -00000002 | | db ? ; undefined |
| -00000001 | | db ? ; undefined |
| +00000000 | s | db 4 dup(?) |
| +00000004 | r | db 4 dup(?) |
| +00000008 | | |
| +00000008 | ; end of stack variables | |

Ya echa esta parte, sigamos con las demás variables

funcion_investigada proc near

```
var_BBCC= dword ptr -0BBCCCh
var_BBC8= byte ptr -0BBC8h
var_BB94= dword ptr -0BB94h
var_BB90= dword ptr -0BB90h
var_BB8C= dword ptr -0BB8Ch
ficha= mia ptr -0BB88h
var_8= byte ptr -8
```

Esa que tenemos marcada es la que corresponde al n que analizamos antes... es decir al “índice” del arreglo.



; Attributes: bp-based frame

funcion_investigada proc near

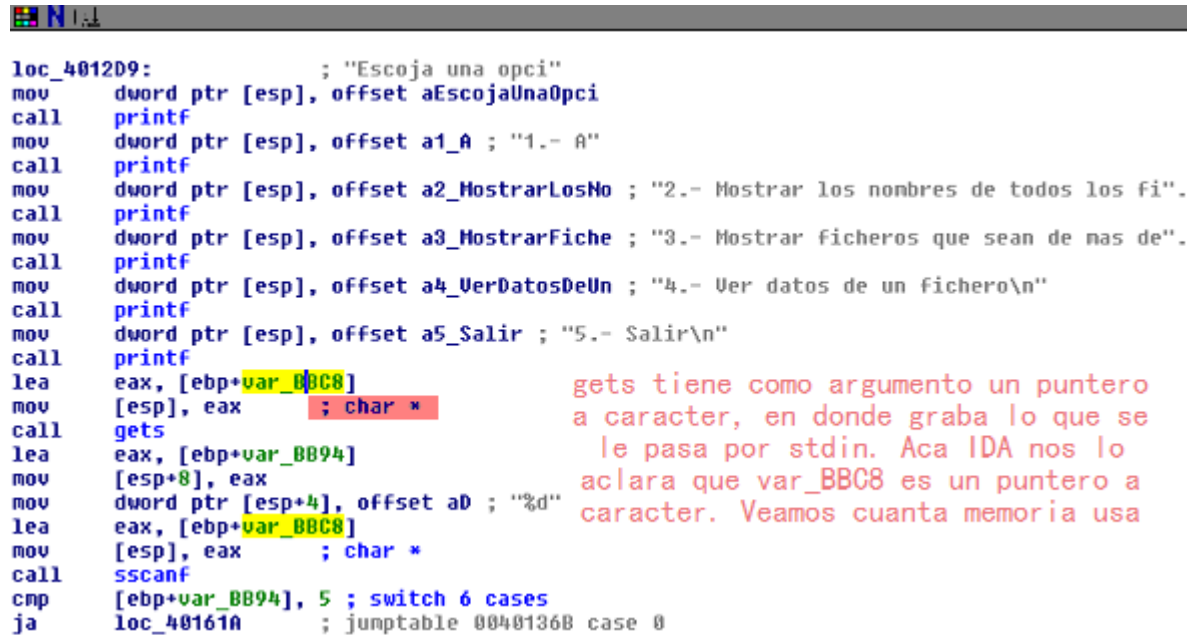
```
var_BBCC= dword ptr -0BBCCCh
var_BBC8= byte ptr -0BBC8h
var_BB94= dword ptr -0BB94h
var_BB90= dword ptr -0BB90h
indice= dword ptr -0BB8Ch
ficha= mia ptr -0BB88h
var_8= byte ptr -8
```

```
push    ebp
mov     ebp, esp
push    ebx
mov     eax, 0BBE4h
call    __chkstk
mov     [ebp+indice], 0
```

En la ultima línea del bloque vemos que arranca desde 0, es decir el comienzo de la estructura. Luego tomara distintos caminos que luego analizaremos en donde lo incrementaran o lo dejaran como esta.

Ya con esas variables que encontramos, estamos en condiciones de encarar el análisis del programa, y poco a poco cuando nos encontremos las otras las iremos definiendo.

Comenzemos



```
loc_4012D9:          ; "Escoja una opci"
mov     dword ptr [esp], offset aEscojaUnaOpci
call    printf
mov     dword ptr [esp], offset a1_A ; "1.- A"
call    printf
mov     dword ptr [esp], offset a2_MostrarLosNo ; "2.- Mostrar los nombres de todos los fi".
call    printf
mov     dword ptr [esp], offset a3_MostrarFiche ; "3.- Mostrar ficheros que sean de mas de".
call    printf
mov     dword ptr [esp], offset a4_VerDatosDeUn ; "4.- Ver datos de un fichero\n"
call    printf
mov     dword ptr [esp], offset a5_Salir ; "5.- Salir\n"
call    printf
lea     eax, [ebp+var_BBC8]
mov     [esp], eax
call    gets
lea     eax, [ebp+var_BB94]
mov     [esp+8], eax
mov     dword ptr [esp+4], offset aD ; "%d"
lea     eax, [ebp+var_BBC8]
mov     [esp], eax
call    scanf
cmp     [ebp+var_BB94], 5 ; switch 6 cases
ja      loc_40161A ; jumptable 0040136B case 0
```

gets tiene como argumento un puntero a caracter, en donde graba lo que se le pasa por stdin. Aca IDA nos lo aclara que var_BBC8 es un puntero a caracter. Veamos cuanta memoria usa

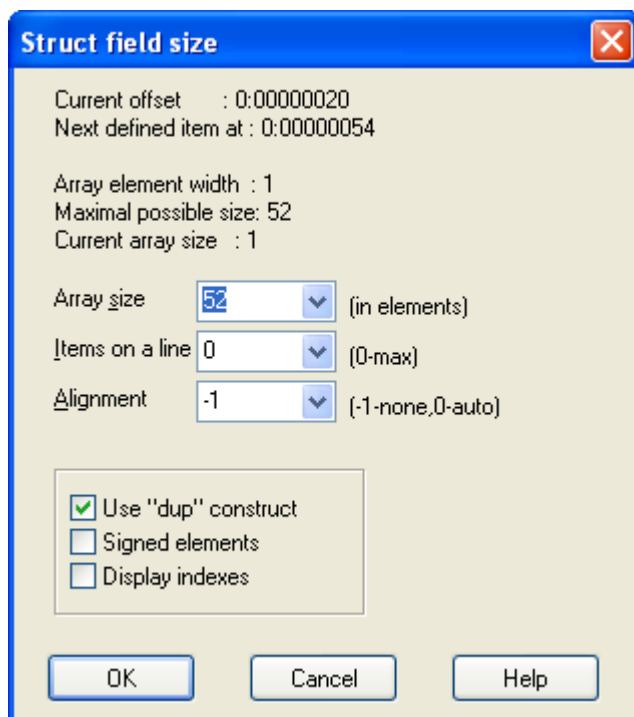
Si lo miramos en el stack frame

```

----- var_BBC8
-0000BBC8 db ?
-0000BBC7 db ? ; undefined
-0000BBC6 db ? ; undefined
-0000BBC5 db ? ; undefined
-0000BBC4 db ? ; undefined
-0000BBC3 db ? ; undefined
-0000BBC2 db ? ; undefined
-0000BBC1 db ? ; undefined
-0000BBC0 db ? ; undefined
-0000BBBF db ? ; undefined
-0000BBBE db ? ; undefined
-0000BBBD db ? ; undefined
-0000BBBC db ? ; undefined
-0000BBBB db ? ; undefined
-0000BBBA db ? ; undefined
-0000BBB9 db ? ; undefined
-0000BBB8 db ? ; undefined
-0000BBB7 db ? ; undefined
-0000BBB6 db ? ; undefined
-0000BBB5 db ? ; undefined
-0000BBB4 db ? ; undefined
-0000BBB3 db ? ; undefined
-0000BBB2 db ? ; undefined
-0000BBB1 db ? ; undefined
-0000BBB0 db ? ; undefined
-0000BBAF db ? ; undefined

```

Nos posicionamos sobre var_BBC8 y le damos a "*" para que nos lo transforme a un array (fijense que este definido como db)



52 bytes le entra, si vieron los ejemplos anteriores se habran percatado que cada vez que ricnar pedia 40 bytes, el compilador se lo mandaba a 52. Andara igual, pero a la hora de reescribir el código lo vamos a hacer de 40. Por ahora le damos a “OK”

```

-0000BBC8 guardado_stdin db 52 dup(?)
-0000BB94 var_BB94 dd ?
-0000BB90 var_BB90 dd ?
-0000BB8C indice dd ?
-0000BB88 ficha mia 1000 dup(?)
-00000008 var_8 db ?

```

Ahí la renombre ha “guardado_stdin” para verla directamente en el listado.

Si seguimos analizando el mismo bloque, veremos que abajo del gets esta una llamada a sscanf, a no desesperar si no la conocen porque es un scanf de toda la vida, pero en vez de trabajar en la entrada estándar trabaja con el puntero a caracter que se le pasa como primer argumento.

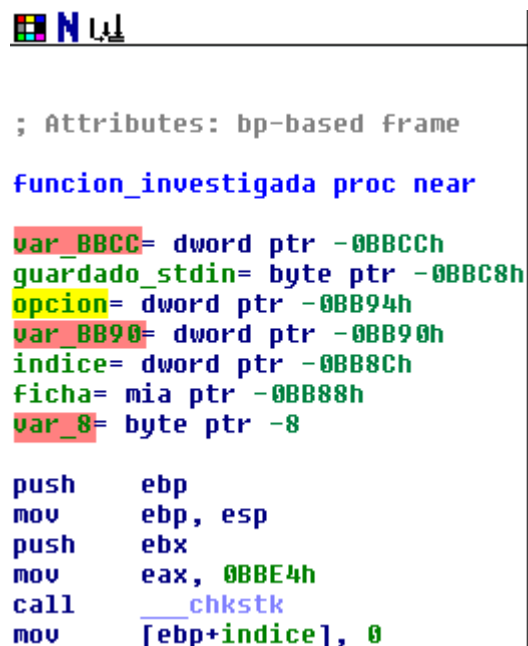
```
int sscanf(const char *cadena, const char *formato,...);
```

aca tenemos el formato, si vemos la llamada es de este tipo

```
sscanf(guardado_stdin, "%ld", &VARIABLE_INT);
```

lo de variable int lo se porque le mete un dígito (fíjense en el formato) y el “&” de adelante es para que se le pase la dirección de la variable así la guarda. Tenemos otra variable, la vamos a llamar “opción” porque es el dígito que nosotros le pasamos con gets formateado como dígito.

Miremos como van nuestras variables



```

; Attributes: bp-based frame

funcion_investigada proc near
var_BBCC= dword ptr -0BBCCCh
guardado_stdin= byte ptr -0BBC8h
opcion= dword ptr -0BB94h
var_BB90= dword ptr -0BB90h
indice= dword ptr -0BB8Ch
ficha= mia ptr -0BB88h
var_8= byte ptr -8

push    ebp
mov     ebp, esp
push    ebx
mov     eax, 0BBE4h
call    __chkstk
mov     [ebp+indice], 0

```

Ya no quedan tantas como al principio ☺ poquito a poquito nos lo vamos comiendo

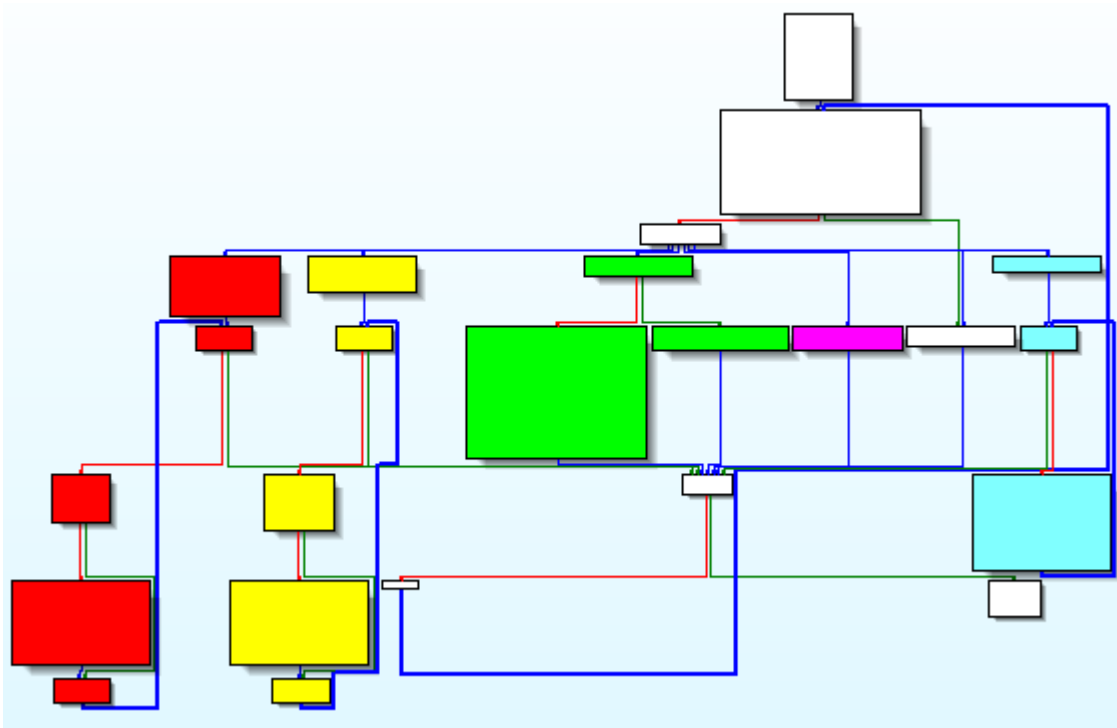
Continuemos analizando

```
loc_401209:                ; "Escoja una opci"
mov     dword ptr [esp], offset aEscojaUna0pci
call    printf
mov     dword ptr [esp], offset a1_A ; "1.- A"
call    printf
mov     dword ptr [esp], offset a2_MostrarLosNo ; "2.- Mostrar los nombres de todos los fi".
call    printf
mov     dword ptr [esp], offset a3_MostrarFiche ; "3.- Mostrar ficheros que sean de mas de".
call    printf
mov     dword ptr [esp], offset a4_VerDatosDeUn ; "4.- Ver datos de un fichero\n"
call    printf
mov     dword ptr [esp], offset a5_Salir ; "5.- Salir\n"
call    printf
lea     eax, [ebp+guardado_stdin]
mov     [esp], eax          ; char *
call    gets
lea     eax, [ebp+opcion]
mov     [esp+8], eax
mov     dword ptr [esp+4], offset aD ; "%d"
lea     eax, [ebp+guardado_stdin]
mov     [esp], eax          ; char *
call    scanf
cmp     [ebp+opcion], 5 ; switch 6 cases
ja      loc_40161A          ; jumtable 0040136B case 0
```

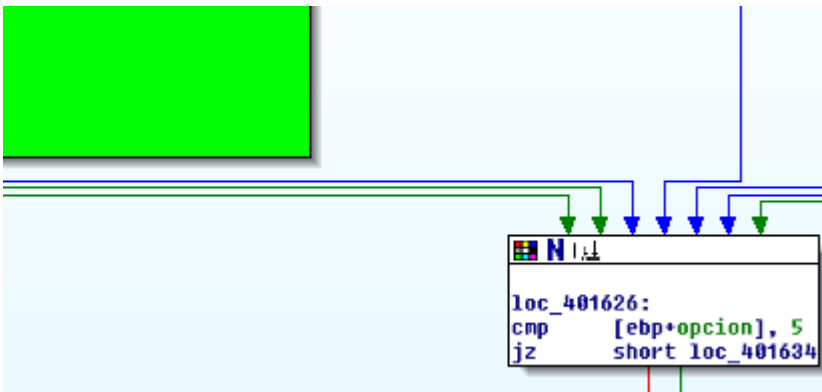
Si la opción que le metemos es mayor a 5 salta a...

```
loc_40161A:                ; jumtable 0040136B case 0
mov     dword ptr [esp], offset a0pci
call    printf
```

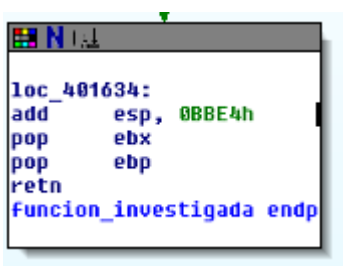
Ya notamos que hay un switch con las opciones de 1 a 5 definidas, aca vemos que a este bloque lo nombro como case 0 este es el definido como "default". Miremos todas las opciones que pone el switch y las coloreamos según cada opción.



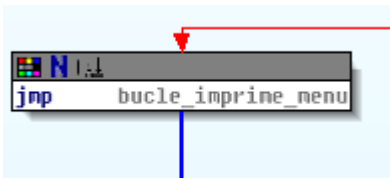
La roja corresponde a la opción **3**, la amarilla a la **4**, la verde a la **1**, la violeta a la **5** y la celeste a la opción **2**. Si miramos, todas las opciones terminan mediante una u otra condición en el bloque blanquito debajo de la opción 1. Vayamos para allá



Comprueba si la opción es “5”, si es así se va a este bloque



“opción de salida”, sino es 5 se va aquí



Es decir, si es 5 sale, sino vuelve a imprimir el menú y espera las entradas como al principio.

Miremos como decide lo del switch, que en este caso no es igual al que mostro ricnar en sus tutes

```

N | ↓
mov     eax, [ebp+opcion]
shl     eax, 2
mov     eax, ds:off_4031DC[eax]
jnp     eax          ; switch jump

```

En este bloque decide para donde va a ir, miremos que hace... a la opción que tenemos (en entero) le hace un corrimiento de 2 a la izquierda. Con lo que aprendimos al principio, ya tenes que saber que hizo un

$$opcion.2^2 = opcion.4$$

Asi indexa los dword del array 4031DC que tiene la dirección de los saltos

```

N | ↓
mov     eax, [ebp+opcion]
shl     eax, 2
mov     eax, ds:off_4031DC[eax]
jnp     eax          ; switch jump
off_4031DC dd offset loc_40161A ; jump table for switch
dd offset loc_40136D
dd offset loc_401416
dd offset loc_401484
dd offset loc_401553
dd offset loc_40160C

```

Ahora pasamos a procesar opción por opción, comienza lo duro...

Para esto no nos vamos a hacer los ultra reverser's, sino que lo ejecutamos y vemos que hace y como lo hace.

Opción 1

```

C:\Documents and Settings\Administrador\Escritorio\1315-C Y REVERSING (pa
Escoja una opción:
1.- Añadir datos de un nuevo fichero
2.- Mostrar los nombres de todos los ficheros
3.- Mostrar ficheros que sean de mas de un cierto tamaño
4.- Ver datos de un fichero
5.- Salir

```

Miremos como lo hace.

```
loc_40136D:                ; jumtable 00401368 case 1
cmp     [ebp+indice], 999
jg      loc_401405
```

Si el índice es mayor a 999, imprime lo siguiente

```
loc_401405:                ; "Máximo de fichas alcanzado (1000)!\n"
mov     dword ptr [esp], offset aMximoDeFichas
call    printf
jmp     loc_401626
```

Nos avisa que solo agunta 1000 (como ya vimos) y salta al bucle de comparación con 5 para decidir si sigue o no el programa.

Sino es mayor a 999 (es menor o igual), nos muestra esto.

```
mov     dword ptr [esp], offset aIntroduceElNom ; "Introduce el nombre del fichero: "
call    printf
lea     ecx, [ebp+ficha] ; guarda la direccion de la estructura
mov     edx, [ebp+indice]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
mov     [esp], eax ; char *
call    gets
mov     dword ptr [esp], offset aIntroduceElTam ; "Introduce el tama"
call    printf
lea     eax, [ebp+guardado_stdin]
mov     [esp], eax ; char *
call    gets
lea     ecx, [ebp+ficha]
mov     edx, [ebp+indice]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
add     eax, 4
mov     [esp+8], eax
mov     dword ptr [esp+4], offset aId ; "%ld"
lea     eax, [ebp+guardado_stdin]
mov     [esp], eax ; char *
call    scanf
lea     eax, [ebp+indice]
inc     dword ptr [eax]
jmp     loc_401626
```

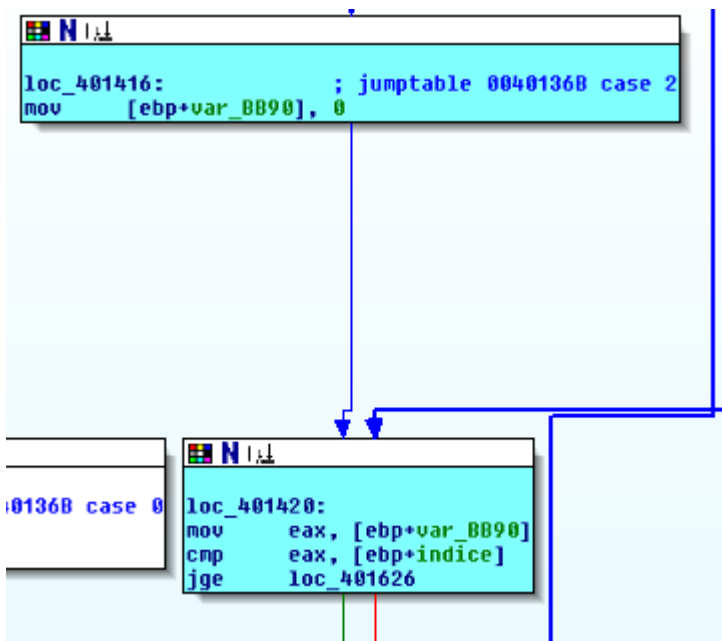
Esto ya lo habíamos visto cuando analizamos la estructura, guarda nombre y tamaño en la estructura indexada por “índice” válgase la redundancia jejeje. Aprecien como no usa gets directamente como dijimos, sino que lo guarda en formato entero de la mano de scanf

Opción 2

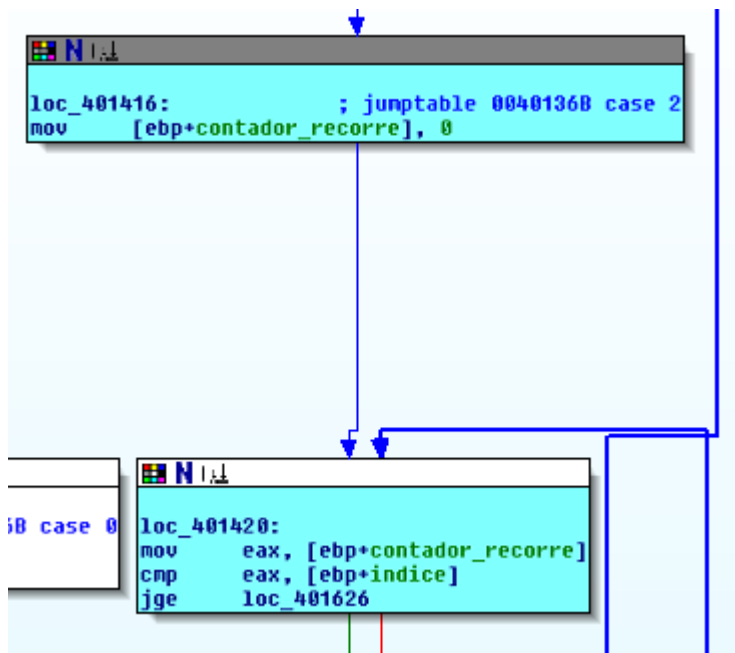
```
C:\Documents and Settings\Administrador\Escritorio\1315-C Y REVERSING (pa
Escoja una opción:
1.- Añadir datos de un nuevo fichero
2.- Mostrar los nombres de todos los ficheros
3.- Mostrar ficheros que sean de mas de un cierto tamaño
4.- Ver datos de un fichero
5.- Salir
```

Para mostrar todos los ficheros, lo mas lógico seria que valla desde 0 a índice indexando el valor del array y mostrarlo por pantalla.

Eso es justo lo que tenemos aca, var_BB90 que todavía no la habíamos mirado, es un contador interno (que para ser originales lo voy a llamar contador_recorre).



Quedamos así



Miremos abajo que mas tenemos



Si el contador recorre es menor a índice, imprime

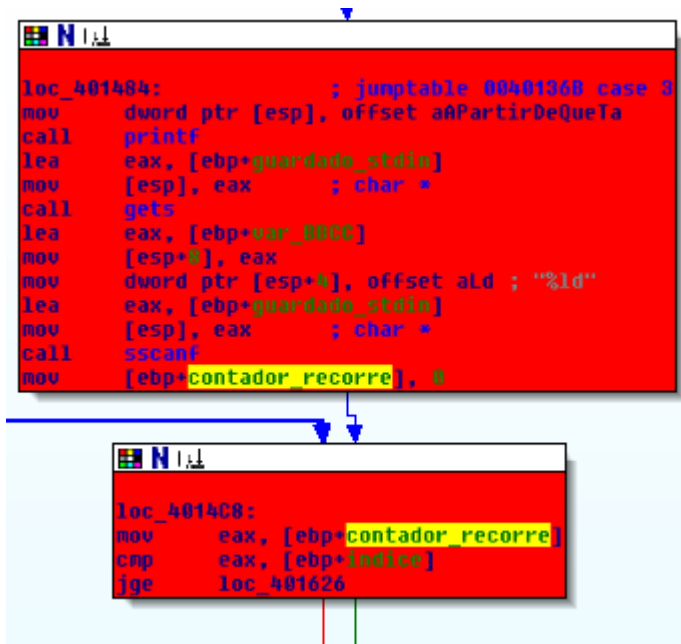
```
printf("Nombre: %s; Tamaño: %d\n", ficha[contador_recorre].nombre,
ficha[contador_recorre].tamano);
```

Fijense que para pasar de uno en uno usa contador_recorre como índice con la condición que este sea menor a "índice" para que no trate de mostrar cosas que no están. La estructura de control ya es conocida por nosotros y se trata de un for (vean como incrementa contador_recorre al final del bloque).

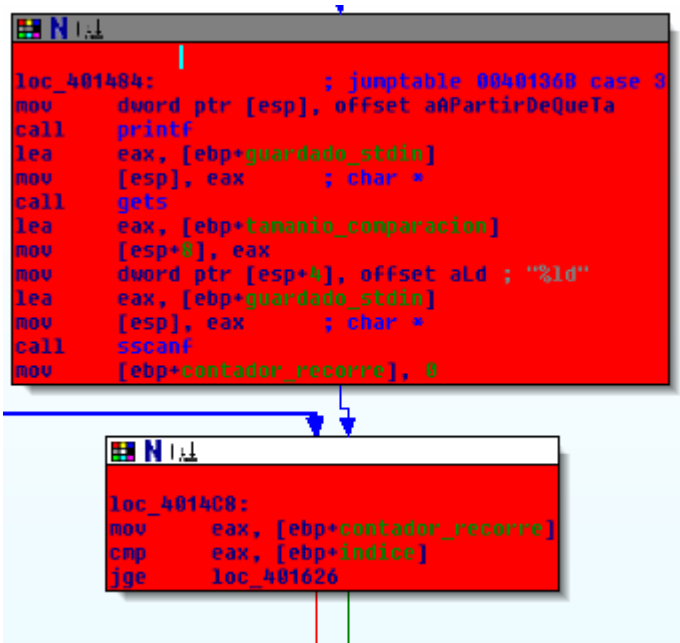
Opción 3

```
C:\Documents and Settings\Administrador\Escritorio\1315-C Y REVERSING (pa
Escoja una opción:
1.- Añadir datos de un nuevo fichero
2.- Mostrar los nombres de todos los ficheros
3.- Mostrar ficheros que sean de mas de un cierto tamaño
4.- Ver datos de un fichero
5.- Salir
```

Muestra los ficheros según una condición impuesta en el campo tamaño



Pregunta a partir de que tamaño quieres ver, lo recoge con gets y después lo guarda en una variable para comparar, vamos a reescribirla como tamaño_comparacion



Sigue aca

```

mov     edx, [ebp+contador_recorre]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     edx, [ebp+var_8]
add     eax, edx
sub     eax, 00060h
mov     eax, [eax+0Ch]
cmp     eax, [ebp+tamano_comparacion]
jb      short loc_401546

```

```

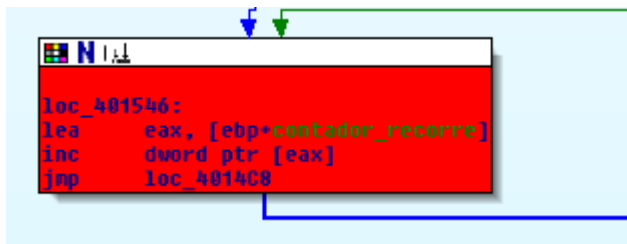
mov     edx, [ebp+contador_recorre]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     edx, [ebp+var_8]
add     eax, edx
sub     eax, 00060h
mov     eax, [eax+0Ch]
mov     [esp+8], eax
lea     ecx, [ebp+ficha]
mov     edx, [ebp+contador_recorre]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
mov     [esp+4], eax
mov     dword ptr [esp], offset aNombreTama ; "Nombre: %s; Tama"
call    printf

```

Esa var_8 que tenemos ahí, es una variable puesta por el compilador así que obviemosla, abajo podemos ver que compara el valor de eax con tamaño_comparacion, en eax tenemos el valor de

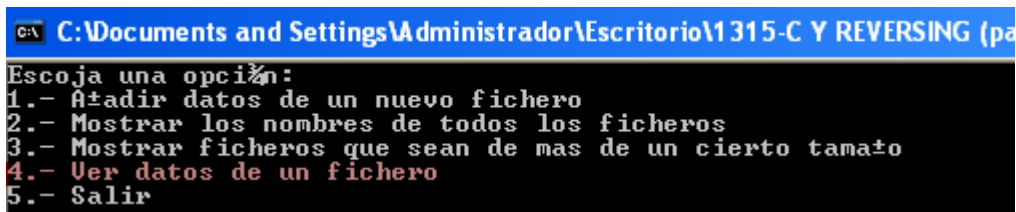
ficha[contador_recorre].tamaño

si el campo tamaño que estamos mirando es mayor que tamaño_comparacion, lo imprime con el printf que vimos en la opción 2, todo esto dentro de un for con la condición de que contador_recorre sea menor a índice como vimos antes

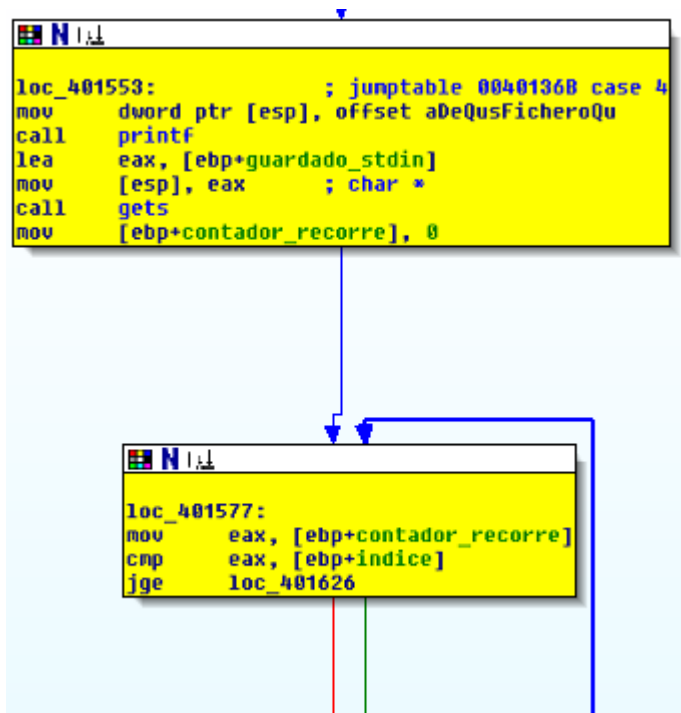


Aca aumenta contador_recorre para seguir el for

Opción 4



Ver datos de un fichero, según el nombre



Pregunta el nombre del fichero que queremos ver y lo guarda en guardado_stdin con gets, esto esta bien porque es un puntero a carácter, después viene el for de siempre

```

lea     ecx, [ebp+guardado_stdin]
lea     ebx, [ebp+ficha]
mov     edx, [ebp+contador_recorre]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ebx+eax]
mov     [esp+4], ecx    ; char *
mov     [esp], eax      ; char *
call    strcmp
test    eax, eax
jnz     short loc_4015FF

```

```

mov     edx, [ebp+contador_recorre]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     edx, [ebp+var_8]
add     eax, edx
sub     eax, 000600h
mov     eax, [eax+0Ch]
mov     [esp+8], eax
lea     ecx, [ebp+ficha]
mov     edx, [ebp+contador_recorre]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     eax, [ecx+eax]
mov     [esp+4], eax
mov     dword ptr [esp], offset aNombreSTama ; "Nombre: %s; Tama"
call    printf

```

Este es un código bastante parecido al anterior, pero ahora compara

ficha[contador_recorre].nombre

con el valor que le metimos por medio de un strcmp, si son iguales lo muestra con el printf de siempre, sino incrementa el contador y sigue hasta que contador_recorre alcance a índice.

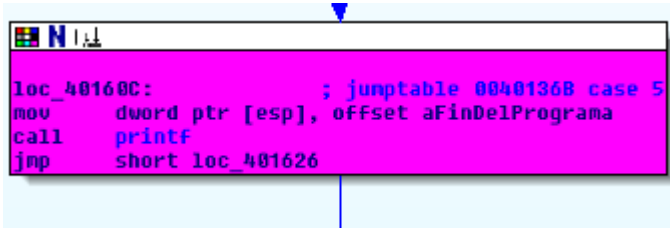
Opción 5

```

C:\Documents and Settings\Administrador\Escritorio\1315-C Y REVERSING (pa
Escoja una opción:
1.- Añadir datos de un nuevo fichero
2.- Mostrar los nombres de todos los ficheros
3.- Mostrar ficheros que sean de mas de un cierto tamaño
4.- Ver datos de un fichero
5.- Salir

```

El mítico salir, vamos a ver si tiene algo de código



Imprime un “Fin del programa” y va a la comparación de si opción es igual a 5 (en la que caen todas), como esta vez si son iguales ejecuta el return.

Reescribiendo el código, ya analizamos todas las opciones y tenemos la estructura de datos que maneja, dejemosnos de vueltas y abramos el dev-c++

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

void funcion_investigada() {
    struct mia {
        char nombre[44];
        int tamano;
    } ficha[1000];
    int indice;
    int contador_recorre;
    int opcion;
    char guardado_stdin[40];
    int tamano_comparacion;

    indice = 0;
    do {
        printf("Escoja una opcion:\n");
        printf("1.- Añadir datos de un nuevo fichero\n");
        printf("2.- Mostrar los nombres de todos los ficheros\n");
        printf("3.- Mostrar ficheros que sean de mas de un cierto tamaño\n");
        printf("4.- Ver datos de un fichero\n");
        printf("5.- Salir\n");

        gets(guardado_stdin);
        sscanf(guardado_stdin, "%d", &opcion);
        switch(opcion) {
            case 1:
                if(indice <= 999) {
                    printf("Introduce el nombre del fichero: ");
                    gets(ficha[indice].nombre);
                    printf("Introduce el tamaño: ");
                    gets(guardado_stdin);
                    sscanf(guardado_stdin, "%ld", &ficha[indice].tamano);
                    indice++;
                } else {
                    printf("Maximo de fichas alcanzado (1000)!\n");
                }
                break;
            case 2:
                for(contador_recorre = 0; contador_recorre < indice;
                    contador_recorre++) {
```



```

        printf("Nombre: %s; Tamaño: %d\n",
ficha[contador_recorre].nombre,
ficha[contador_recorre].tamaño);
    }
    break;
    case 3:
    printf("A partir de que tamaño quieres que te muestre: ");
    gets(guardado_stdin);
    sscanf(guardado_stdin, "%ld", &tamaño_comparacion);

    for(contador_recorre = 0; contador_recorre < indice;
contador_recorre++) {
        if(ficha[contador_recorre].tamaño >=
tamaño_comparacion) printf("Nombre: %s; Tamaño: %d\n",
ficha[contador_recorre].nombre,
ficha[contador_recorre].tamaño);
    }
    break;
    case 4:
    printf("De que ficheros quiere ver todos los datos?");
    gets(guardado_stdin);
    for(contador_recorre = 0; contador_recorre < indice;
contador_recorre++) {
        if(strcmp(ficha[contador_recorre].nombre,
guardado_stdin) == 0) {
            printf("Nombre: %s; Tamaño: %d\n",
ficha[contador_recorre].nombre,
ficha[contador_recorre].tamaño);
        }
    }
    break;
    case 5:
    printf("Fin del programa\n");
    break;
    default:
        printf("Opcion desconocida\n");
        break;
    }
} while(opcion != 5);
return;

}

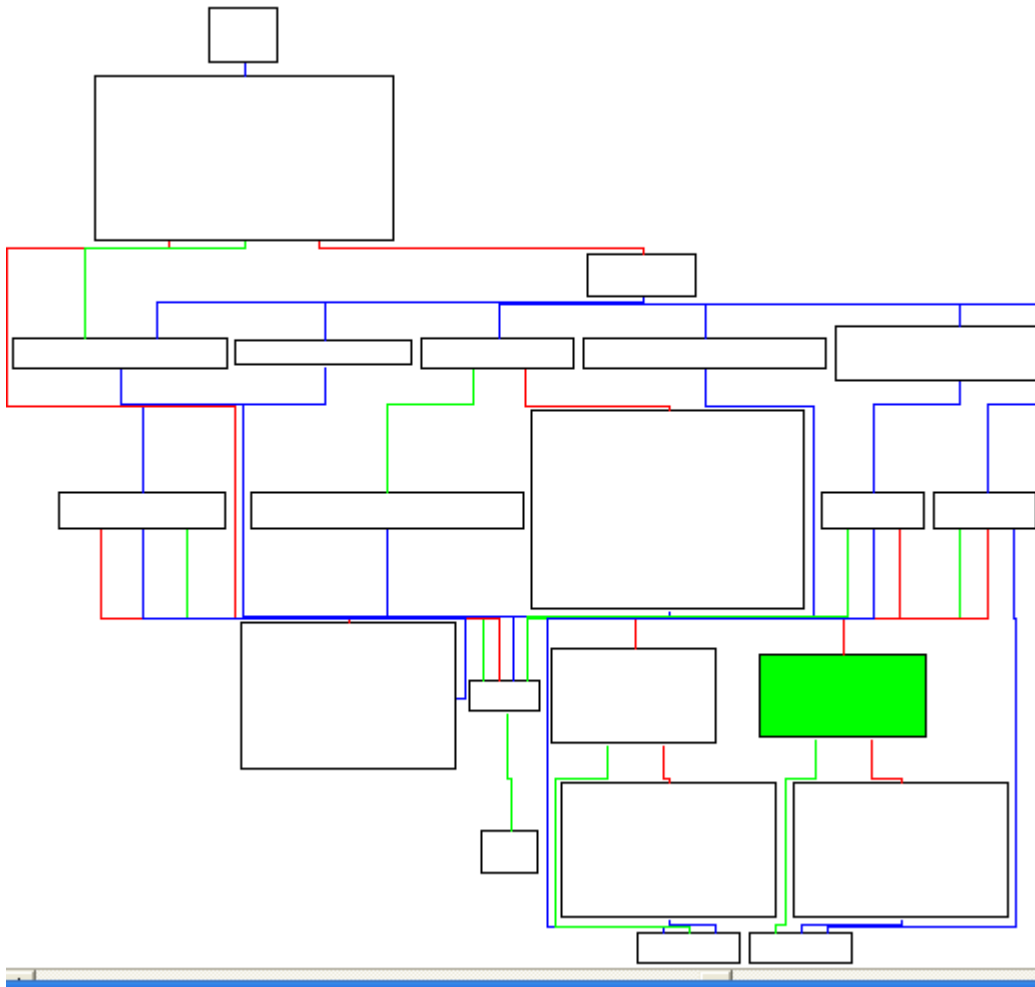
main() {
    funcion_investigada();
}

```

El código queda largo, pero no es nada del otro mundo, compilemoslo y comparémoslo con el original

| Address | Displacement | Symbol Name | Address | Symbol Name |
|---------------|--------------|------------------|---------|---------------------|
| identical | 401c20 | ExitProcess | 401c10 | ExitProcess |
| identical | 401c60 | __slli_init_ctor | 401c50 | __slli_init_ctor |
| suspicious + | 401290 | sub_401290 | 4012c1 | funcion_investigada |
| suspicious ++ | 40160c | _main | 401290 | _main |

Casi, pero no... veamos las diferencias



Esta todo igual, menos una partecita, veamos que es

```

ID_11
4014a9: chk=1138
mov     edx, [ebp+var_BB90]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     edx, [ebp+var_8]
add     eax, edx
sub     eax, 0BB60h
mov     eax, [eax+0Ch]
cmp     eax, [ebp+var_BBCC]
jl      short loc_401515

```

```

ID_11
4014da: chk=112e
mov     edx, [ebp+contador_recorre]
mov     eax, edx
add     eax, eax
add     eax, edx
shl     eax, 4
lea     edx, [ebp+bandera_comienzo_x_compilador]
add     eax, edx
sub     eax, 47968
mov     eax, [eax+12] ; ficha[contador_recorre].tamano
cmp     eax, [ebp+tamano_comparacion]
jb      short loc_401546

```

Nooooooooooooooooooooo... un jb contra un jl, eso es lo único.

Los dos saltan cuando están por debajo, pero jl usa números con signo y jb no. Ahí esta la diferencia, nosotros definimos

```
int tamaño;
```

dentro de struct, y en realidad es un entero sin signo (tiene criterio, el tamaño no va a ser negativo ☺), lo reescribimos como

```
unsigned int tamaño;
```

compilamos y comparamos

| | | | | |
|-----------|--------|-----------------------------|--------|------------------|
| identical | 401c00 | SetUnhandledExceptionFilter | 401c10 | SetUnhandledEx |
| identical | 401c10 | ExitProcess | 401c20 | ExitProcess |
| identical | 401c50 | __sjli_init_ctor | 401c60 | __sjli_init_ctor |
| identical | 4012c1 | funcion_investigada | 401290 | sub_401290 |
| identical | 401200 | ... | 401200 | ... |

Objetivo superado, no les voy a mentir, costo bastante y no salio a la primera, asi que a no desanimarse.

Adjunto el source code del programa.

Espero que les haya gustado