C Y REVERSING (parte 9) por Ricnar

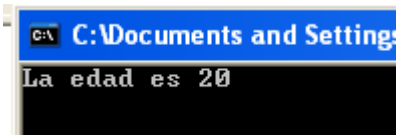
ARRAY DE ESTRUCTURAS

Como ya vimos lo que es un array y ya vimos lo que es una estructura, pues es fácil deducir que un array de estructuras, es un array cuyos campos son todos iguales y son estructuras jeje.

Si al código del ejemplo anterior lo cambiamos un poco

```
#include <stdio.h>

main(){

 funcion1();
 getchar();
}

funcion1(){

struct mia
  {
    char  inicial;
    int   edad;
    float nota;
  } persona[20];

  persona[1].inicial = 'J';
  persona[2].edad = 20;
  persona[11].nota = 7.5;
  printf("La edad es %d", persona[2].edad);
```

Vemos que ahora **persona** es un array de 20 campos del tipo **mia** cada uno, o sea que habrá un **persona[0]** donde se podrá setear su inicial, edad y nota, un **persona[1],** etc así sucesivamente, si lo corremos funciona pues imprimimos **persona[2].edad** que lo acabamos de inicializar antes.



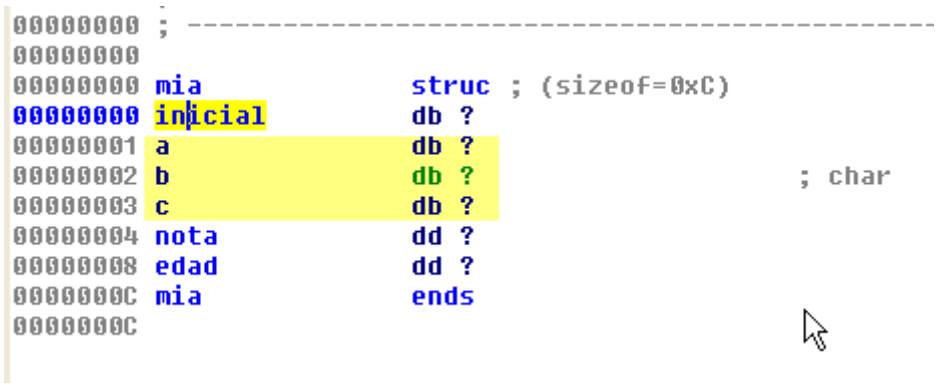
Veamos en IDA a ver como podemos mostrarlo en una forma que respete los indices y nos manejemos como la misma variable que el código fuente o sea un array de estructuras.

```
sub_4012C6 proc near

var_EC= byte ptr -0ECh
var_DC= dword ptr -0DCh
var_6C= dword ptr -6Ch

push    ebp
mov     ebp, esp
sub     esp, 108h
mov     [ebp+var_EC], 4Ah
mov     [ebp+var_DC], 14h
mov     eax, 40F00000h
mov     [ebp+var_6C], eax
mov     eax, [ebp+var_DC]
mov     [esp+4], eax
mov     dword ptr [esp], offset aLa
call    printf
leave
retn
sub_4012C6 endp
```

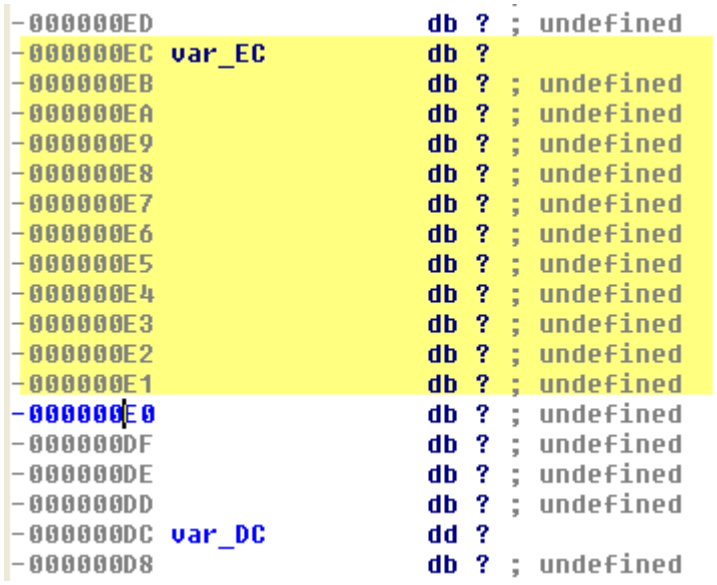
Las variables son como en el ejemplo anterior, una de un byte y dos dwords y como vimos el IDA crea variables, para los campos de un array o una estructura que va a usar, los otros que no usa quedan indefinidos, vemos la **var_EC** que es un **char** para guardar la letra **J** en **persona[1].inicial,** luego un montón de bytes indefinidos que son los campos que no usa o no se inicializan, luego lo mismo los dos dwords para guardar los valores de **persona[2].edad** y **persona[11].nota.**

```
-000000ED                db ? ; undefined
-000000EC var_EC         db ?
-000000EB                db ? ; undefined
-000000EA                db ? ; undefined
-000000E9                db ? ; undefined
-000000E8                db ? ; undefined
-000000E7                db ? ; undefined
-000000E6                db ? ; undefined
-000000E5                db ? ; undefined
-000000E4                db ? ; undefined
-000000E3                db ? ; undefined
-000000E2                db ? ; undefined
-000000E1                db ? ; undefined
-000000E0                db ? ; undefined
-000000DF                db ? ; undefined
-000000DE                db ? ; undefined
-000000DD                db ? ; undefined
-000000DC var_DC          dd ?
-000000D8                db ? ; undefined
```

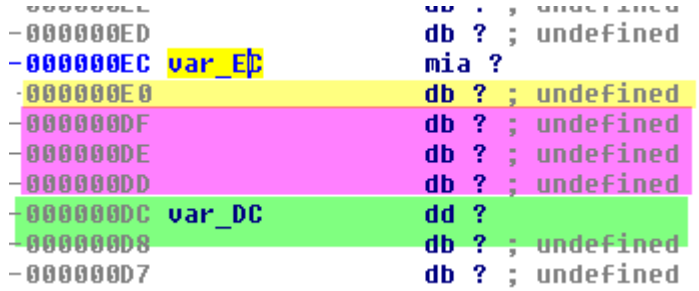
Vamos a estructuras y creamos la misma que antes con los tres bytes basura



Vemos que considerando los tres bytes basura la estructura tiene 0Ch de largo o sea 12 y que si la hacemos de ese tamaño, luego hay 4 bytes que son el primer char del segundo campo, luego los tres bytes basura o sea 4 bytes en total y ya viene la segunda variable que es un dword sin estar desfasada así que poner los bytes de relleno parece estar bien sigamos.

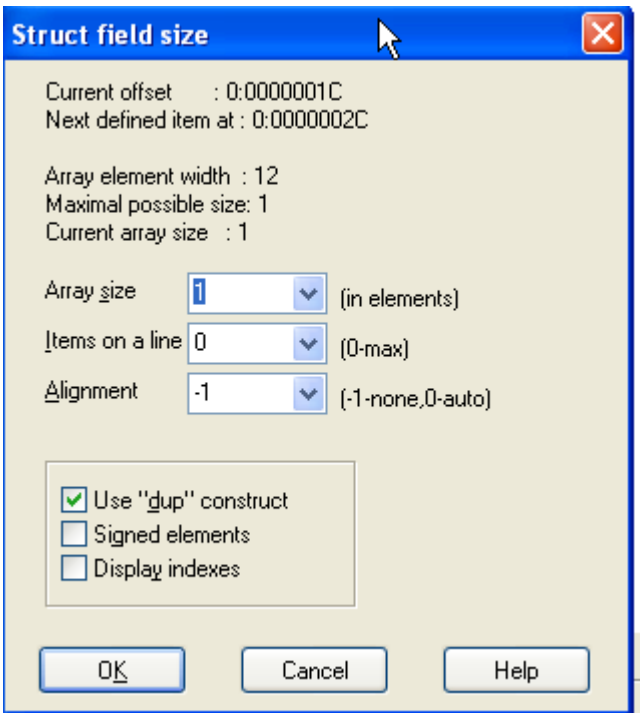


Hagamos que **var_EC** sea del tipo de la estructura mia.

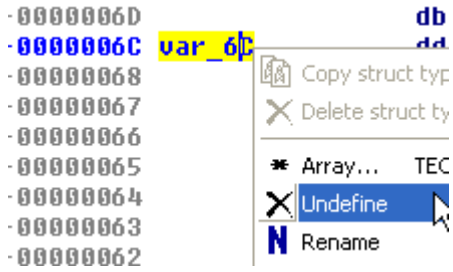


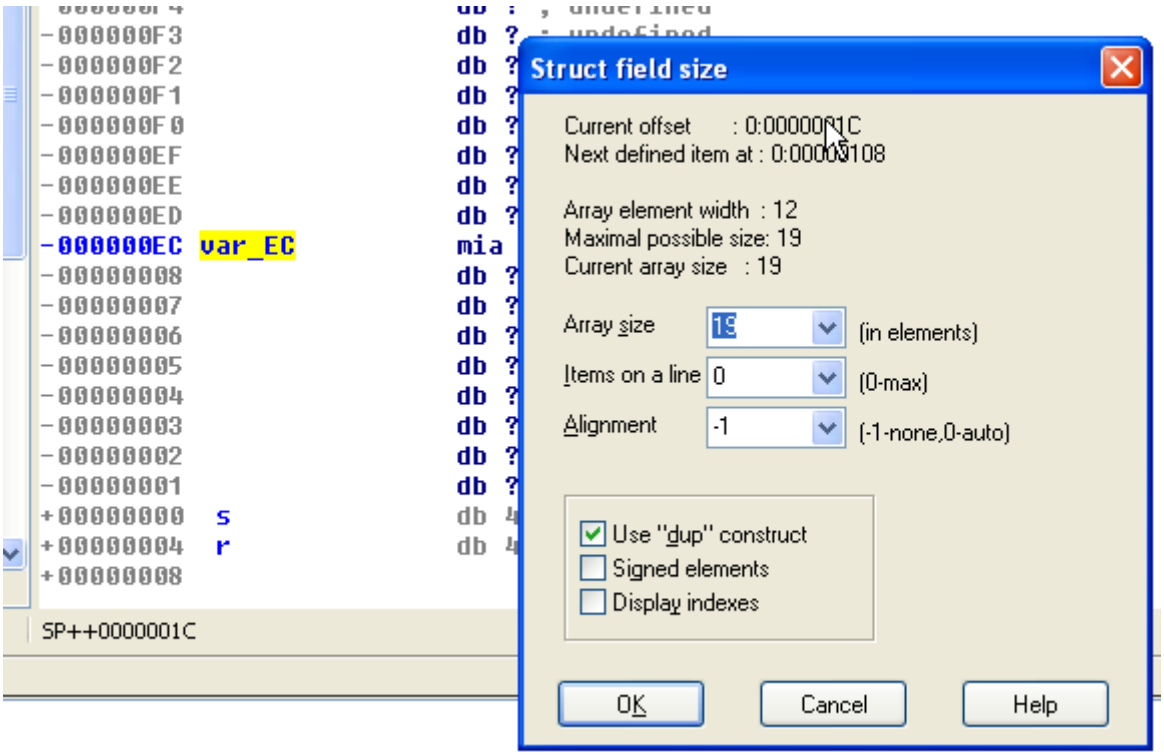
Vemos que coincide pues una ves que **var_EC** es del tipo **mia** y abarca 12 bytes como dije, en amarillo se ve en la imagen que viene el byte que corresponde al siguiente campo, sera **persona[2].inicial**, luego en rosa los tres bytes basura y en verde que es lo importante viene el dword para **persona[2].edad** que es el que debe coincidir para **var_DC.**

Ahora como cuando hicimos los arrays vimos que como son todos los campos similares cuando definimos el primero, solo hay que apretar asterisco en el mismo y poner el largo, en este caso, ponemos el cursor en **var_EC,** apretamos asterisco y nos sale.

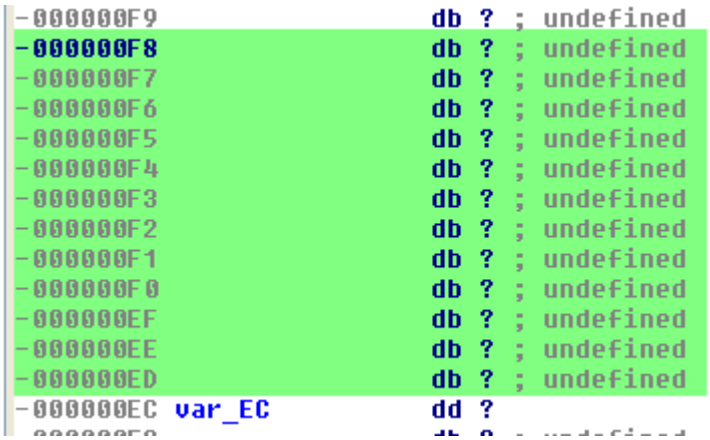


Si no tenemos ganas de hacer cuentas como sabemos que las otras dos variables que hay debajo pertenecen a este array de estructuras, las undefinimos para que desaparezcan y luego volvemos a apretar el asterisco.

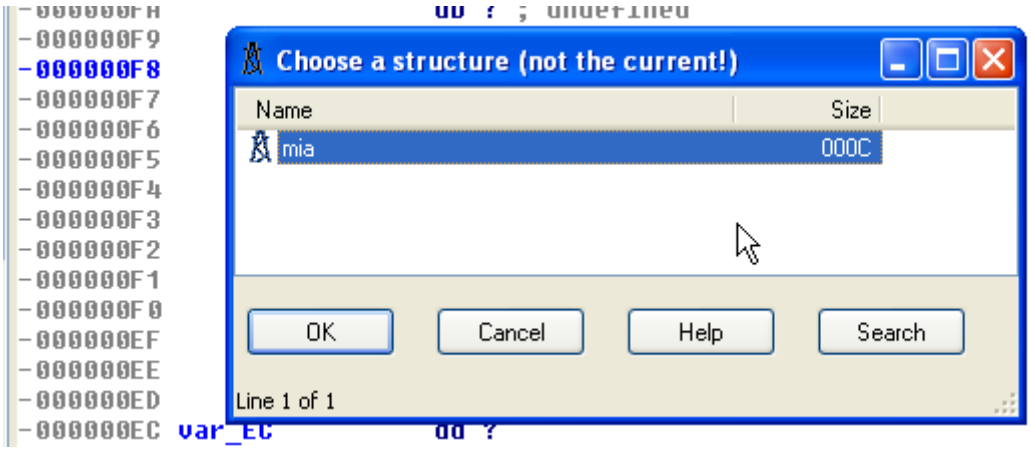


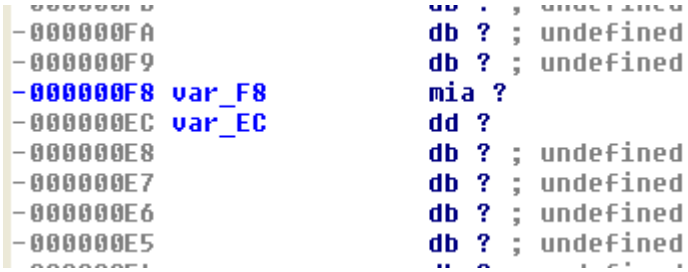


Vemos que nos dice que puede haber solo 19 campos como máximo, jeje hemos cometido un error, y eso es porque no nos dimos cuenta que la primera variable que creo IDA **var_EC**, corresponde a **persona[1]**, pero hay un **persona[0]** justo arriba de 12 bytes que no esta dentro del array que creamos.

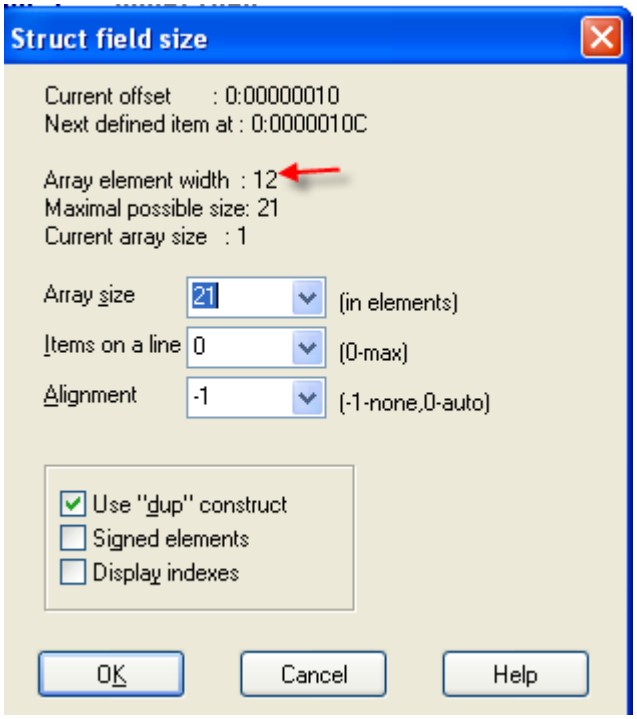


Ahí esta si contamos 12 hacia arriba desde **var_EC**, encontramos el inicio verdadero que esta en F8, allí apretamos ALT mas Q y le asignamos un tipo de estructura **mia**.



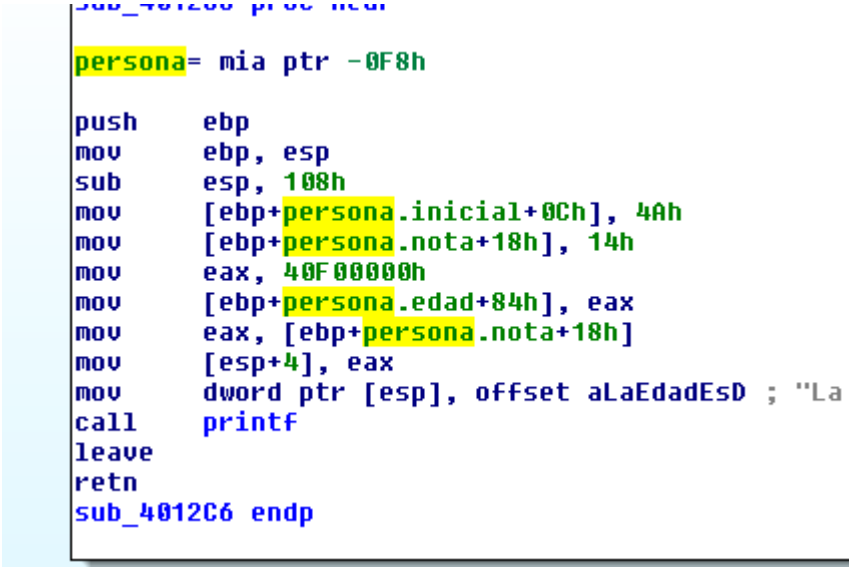


Ahora podemos undefinir la variable **var_EC** para ver cuanto espacio queda al apretar asterisco.



Ahora si ponemos 20 y vemos que cada campo tiene 12 bytes de largo.

Renombramos a persona.



```
persona= mia ptr -0F8h

push    ebp
mov     ebp, esp
sub     esp, 108h
mov     [ebp+persona.inicial+0Ch], 4Ah
mov     [ebp+persona.nota+18h], 14h
mov     eax, 40F00000h
mov     [ebp+persona.edad+84h], eax
mov     eax, [ebp+persona.nota+18h]
mov     [esp+4], eax
mov     dword ptr [esp], offset aLaEdadEsD ; "La
call    printf
leave
retn
sub_4012C6 endp
```

Allí vemos que ahora los indices coinciden pues como cada campo tiene 0ch de largo al dividir el indice por 0c nos dice que campo es.

Como 0Ch / 0Ch es igual a **1**

persona.inicial+0Ch es **persona[1].inicial**

Como 18h / 0Ch es igual a **2**

persona.nota+18h es persona[2].nota

Como 84h / 0Ch es igual a Bh o sea **11**

persona.edad+84h es persona[11].edad

Coinciden los indices con la estructura de nuestro código fuente.

Ahí puse un ejemplo para reversear que incluye en un solo ejecutable casi todo lo que vimos hasta ahora, tiene switchs, estructuras, strings, loops, bah un poco de todo veamos si pueden hacerlo, sino lo haré yo en la parte siguiente.

Suerte que es bravo jeje
Hasta la parte 10
Ricardo Narvaja