

C Y REVERSING (parte 6) por Ricnar

MOLDES O CASTING (CASTEO)

En C tenemos la posibilidad de forzar y transforma una variable que esta declarada de un tipo a otro tipo usando el llamado casteo o moldeo de la misma, veamos varios ejemplos:

Ejemplo 1:

Veamos este código:

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

```
main(){
```

```
    funcion1();  
    getchar();  
}
```

```
funcion1(){
```

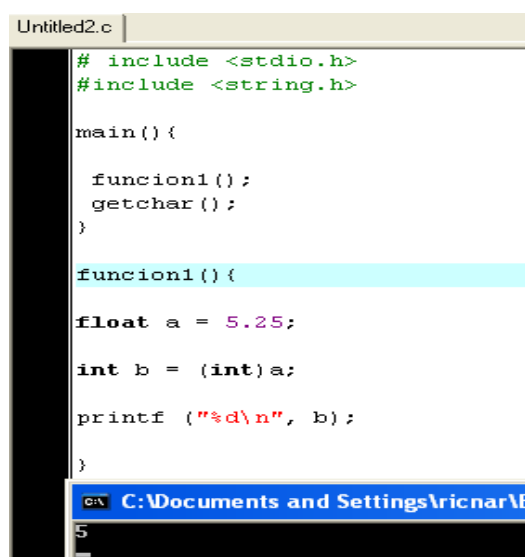
```
    float a = 5.25;
```

```
    int b = (int)a;
```

```
    printf ("%d\n", b);
```

```
}
```

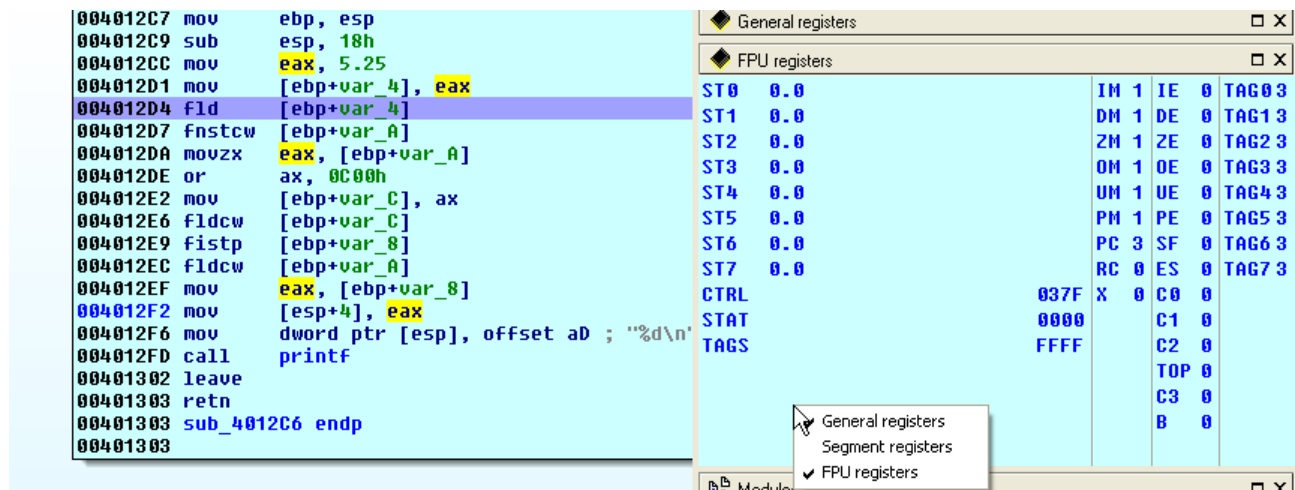
Es una variable llamada **a** declarada como **float** y asignado un **float 5.25** que se castea poniéndole el **(int)** delante y **a** no cambia sigue siendo **float** pero al usar **(int)a** convertirá el valor a **int** y lo asignara en **b** como **int**, vemos al imprimir el valor de **b** que vale 5.



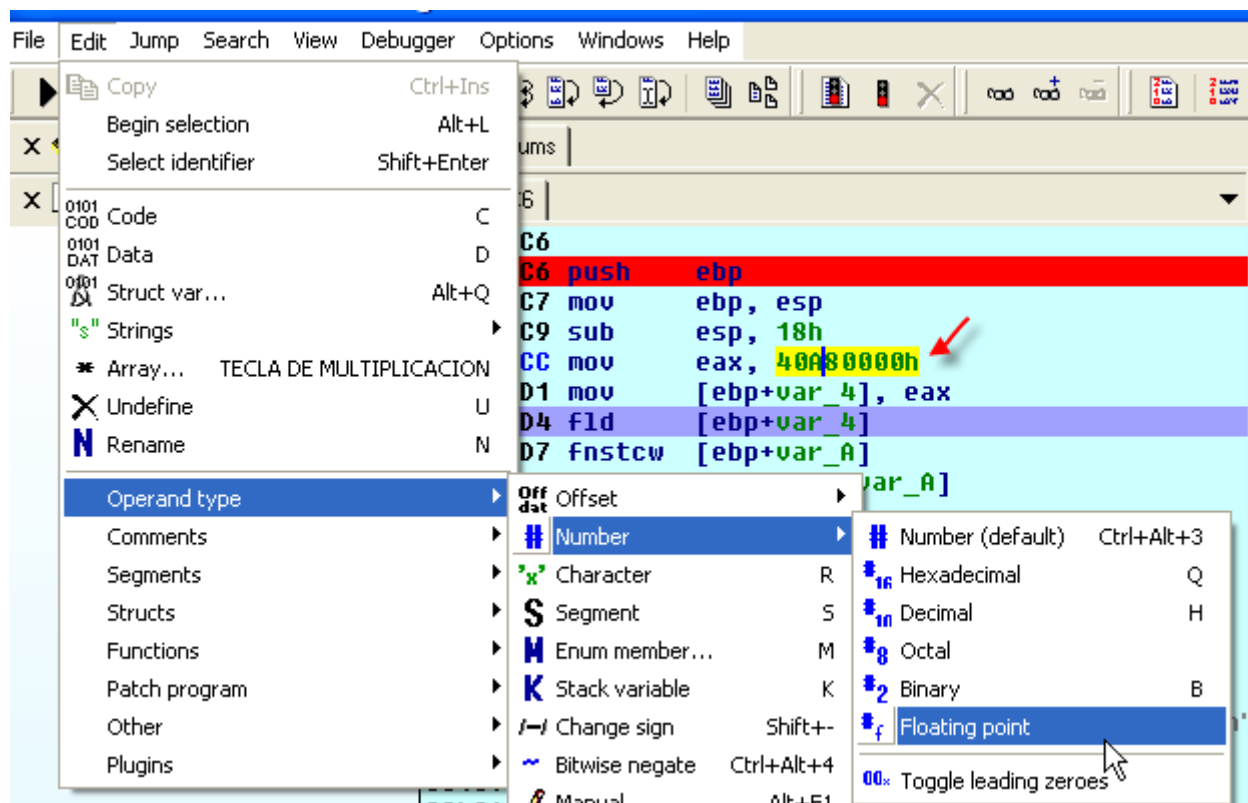
```
Untitled2.c  
#include <stdio.h>  
#include <string.h>  
  
main() {  
    funcion1();  
    getchar();  
}  
  
funcion1() {  
    float a = 5.25;  
    int b = (int)a;  
    printf ("%d\n", b);  
}
```

C:\Documents and Settings\ricnar\B
5
_

Si arrancamos el debugger del IDA y paramos en el inicio



Cambiamos el stack para ver el stack de punto flotante.



Cambiamos el valor para que lo muestre como punto flotante (yo ya lo había hecho en la primera imagen, lo volví a repetir para que vea como se hace).

Vemos que **var_4** es inicializada con el valor 5.25 **float**, así que la renombramos como en el código fuente con el nombre **a**, recordamos que el **sizeof** en C de una variable **float** es 4, así que **dword** esta correcto en IDA como largo de la variable **a**.

ST0	5.25	IM 1	IE	TAG0 3
ST1	0.0	DM 1	DE	TAG1 3
ST2	0.0	ZM 1	ZE	TAG2 3
ST3	0.0	OM 1	OE	TAG3 3
ST4	0.0	UM 1	UE	TAG4 3
ST5	0.0	PM 1	PE	TAG5 3
ST6	0.0	PC 3	SF	TAG6 3
ST7	0.0	RC 0	ES	TAG7 0
CTRL		037F	X 0	C0
STAT		2000		C1

FLD carga el valor de **a** en **ST0** del stack de punto flotante.

Hay un par de instrucciones con el Control Word que no son relativas a nuestro valor y aquí.

fistp [ebp+var_8]

FIST convierte a **int** el **float** y lo **popea** de **ST0** a **var_8** en este caso.

Quiere decir que podemos renombrar **var_8** con el nombre **b** como en el código fuente, las otras dos variables auxiliares son usadas para trabajar con el control word del punto flotante y no son del código fuente nuestro así que las dejamos como están.

Luego de otra instrucción con el control word seguimos la variable **b** que la pasa al stack para usarlo como argumento de **printf**, ya vimos que es un **int** que vale **5** y allí mediante el format string **%d** imprimirá su valor.

```
.text:004012EF mov     eax, [ebp+b]
.text:004012F2 mov     [esp+4], eax
.text:004012F6 mov     dword ptr [esp], offset aD ; "%d\n"
.text:004012FD call    printf
```

El siguiente ejemplo es un casteo de char a int.

#include <stdio.h>

main(){

```
    funcion1();
    getchar();
}
```

funcion1(){

char b = 'a';

```
int x = (int)b;
printf ("%d", x);
```

}

```
# include <stdio.h>


main(){

    funcion1();
    getchar();
}

funcion1(){

    char b = 'a';


    int x = (int)b;
    printf ("%d", x);
}
```



Allí vemos el casteo de la variable **char** a **int** y imprime el valor ASCII del carácter **a** en decimal, veamos como se ve en el IDA.

```
funcion1 proc near
var_8= dword ptr -8
var_1= byte ptr -1

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp+var_1], 61h
movsx   eax, [ebp+var_1]
mov     [ebp+var_8], eax
mov     eax, [ebp+var_8]
mov     [esp+4], eax
mov     dword ptr [esp], offset :
call    printf
leave
retn
funcion1 endp
```



Allí vemos como mueve a la variable declarada como char que aquí es un byte el valor 61h que es la **a** minúscula lo cambiamos para que muestre la a con el menú del click derecho y renombramos la variable como **b**.

```
movsx   eax, [ebp+b]
mov     [ebp+var_8], eax
```

Allí vemos como toma el valor y lo convierte a **int** mediante el movsx y lo guarda en **var_8** que es la otra variable **int** así que la renombramos a **x**.

```

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp+b], 'a'
movsx   eax, [ebp+b]
mov     [ebp+x], eax
mov     eax, [ebp+x]
mov     [esp+4], eax
mov     dword ptr [esp], offset aD ; "%d"
call    printf
leave
retn

```

Luego vemos que sencillamente imprime su valor.

Muchas veces al imprimir el valor haciendo **printf** este hace un casting on the fly para hacer format string, depende de lo que le coloquemos luego del %, pero hay que saber que esto es solo visual, si una variable no fue casteada antes, y solo se muestra por consola su salida en un formato diferente, si luego se siguen realizando operaciones con la misma variable sin castear, esta seguirá en su tipo original, por lo cual es importante realizarlo antes del **printf** en nuestro código fuente, en el caso anterior vemos que al pasar la variable **x** casteada como **int** al hacer **printf** se vera el valor **int** y además **x** queda siendo una variable **int**, mientras que si no casteamos antes y hacemos el **printf** de la variable **b** que es un **char** si ponemos que en el **printf** lo muestre como **%d** lo hará, mostrara el valor como si fuera un **int**, por el casting on the fly, pero no transformara la variable **b** la cual seguirá siendo de tipo **char** y para futuras operaciones podría fallar si la usamos como **int**.

The screenshot shows a code editor window with the following C code:

```

#include <stdio.h>

main(){
    funcion1();
    getchar();
}

funcion1(){
    char b = 'a';

    int x = (int)b;
    printf ("%d\n", x);
    printf ("%d\n", b);
}

```

Below the code editor, a console window displays the output of the program:

```

C:\Documents and Settings\ricnar\...
97
97

```

Aquí vemos ese ejemplo claramente el resultado de ambas variables mostrado es similar por el casteo al vuelo para hacer **printf**, pero si vemos en IDA veremos que **x** y **b** son variables de diferente tamaño y tipo.

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

```
main(){
```

```
    funcion1();  
    getchar();  
}
```

```
funcion1(){
```

```
    char b = 'a';
```

```
    int x = (int)b;  
    printf ("%d\n", x);  
    printf ("%d\n", b);  
}
```

```
x= dword ptr -8  
b= byte ptr -1  
  
push    ebp  
mov     ebp, esp  
sub     esp, 18h  
mov     [ebp+b], 61h  
movsx   eax, [ebp+b]  
mov     [ebp+x], eax  
mov     eax, [ebp+x]  
mov     [esp+4], eax  
mov     dword ptr [esp], 0  
call    printf  
movsx   eax, [ebp+b]  
mov     [esp+4], eax  
mov     dword ptr [esp], 0  
call    printf  
leave  
retn
```

Vemos que en el primer **printf** que usa la variable **x** que es un **int**, la pasa directamente como argumento tipo **dword** para el **printf** usando **%d**, mientras que en el segundo **printf**, lo convierte temporalmente a **dword** al vuelo, sin cambiar la variable que es un **byte** al moverlo a **EAX** el cual queda como **dword** y es pasado como argumento de **printf** y **%d**, por lo tanto el casteo es **on the fly** y no fue guardado ni cambiado nada, **b** sigue siendo un **char** de un **byte** y seguirá por el resto del programa siendo de un **byte**, así que no hay que confundirse, no pensar que porque **printf** lo muestre como **int** la variable es un **int**, aunque visualmente el resultado sea el mismo.

Bueno terminamos lo de casting, quisiera que resuelvan los ejercicios ambos tienen casting, y quisiera que no solo alguien me enviara el código fuente sino el tute con el reversing explicado así me salvo de hacerlo yo, jeje

Hasta la parte 7

Ricardo Narvaja