

## C Y REVERSING (parte 3) por Ricnar

Continuaremos paso a paso con diferentes códigos en C, agregaremos ahora la comprobación de si se cumple una condición o `if`. Dejamos para mas adelante el manejo de cadenas de caracteres, para seguir el mismo orden que el Curso de C que estamos usando como base para reversear.

Veamos este código:

```
#include <stdio.h>

main(){
    funcion2();
    getchar();
}

funcion2(){

    int numero;

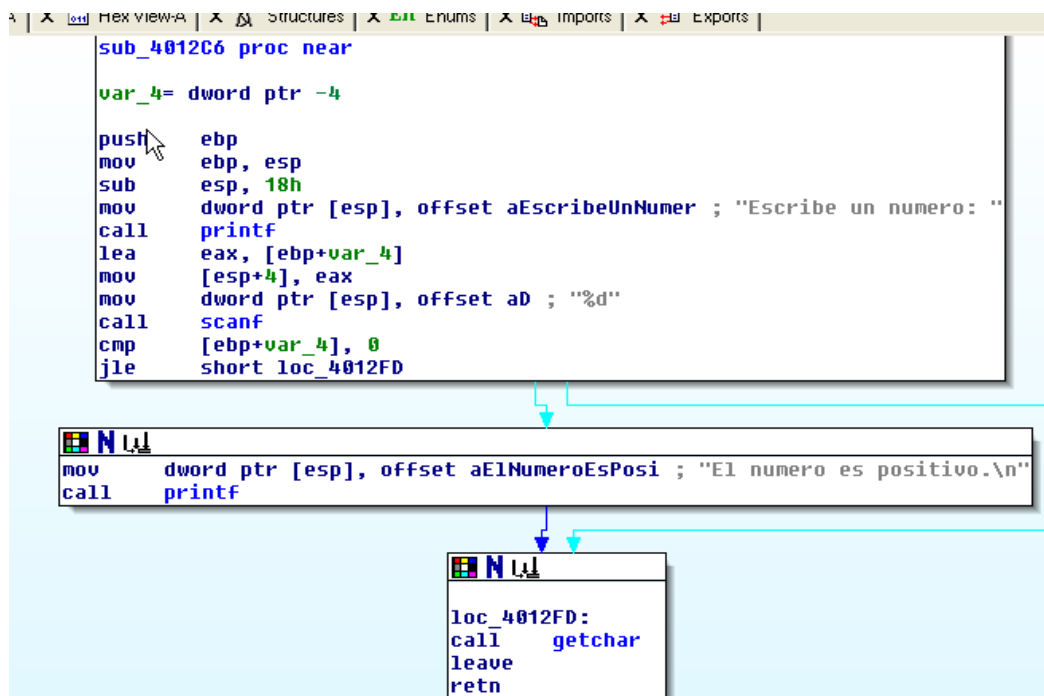
    printf("Escribe un numero: ");
    scanf("%d", &numero);
    if (numero>0) printf("El numero es positivo.\n");

    getchar();
}
```

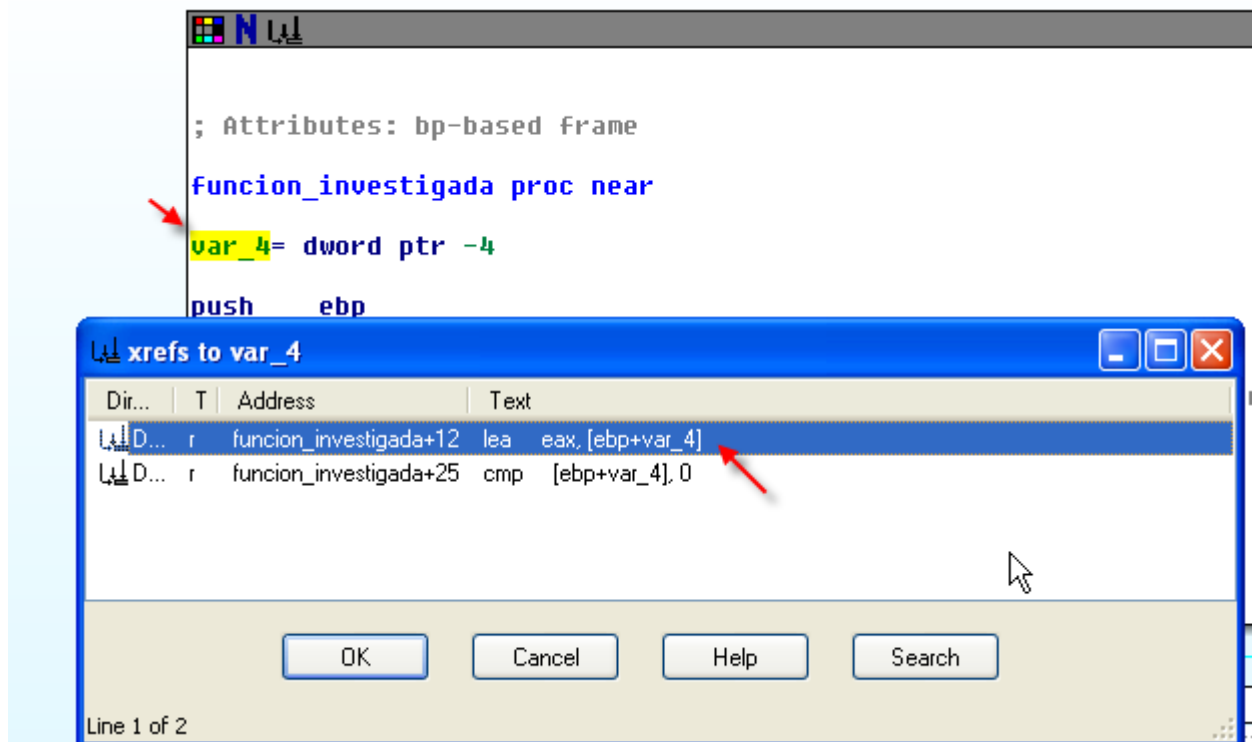
Vemos que luego de crear una variable **int** y mostrar el mensaje de que imprima un numero usando **printf**, usa **scanf** para que el usuario tipee un numero el cual guarda en la variable pasandole la dirección de la misma.

A continuación el **if**, en el paréntesis siguiente al mismo verifica la condición, en este caso si el numero tipeado es mayor que 0 o sea positivo y si es así va a ejecutar el **printf** que muestra el texto “**El numero es positivo**”, sino continua el programa y va al **getchar** para luego de que tipeemos ENTER se cierre.

Veamos como se ve este código sencillo en el IDA, compilemos y abramoslo.



Si yo no conozco el código fuente veo que es una funcion con una sola variable, así que le cambio el nombre a la funcion, por algo propio, muchos objetan que cambiemos al iniciar el nombre de la funcion, pero tenemos que entender que muchas veces estaremos reverseando funciones de programas muy extensos, y el hecho de cambiarle el nombre hará que resalte mas si estamos en otra parte del programa y vemos que llama a la susodicha funcion, que con el nombre original sera una mas entre todas las funciones con nombres numericos.



A continuación miraremos la variable, al marcar la misma y apretar X para las referencias vemos que hay un lea o sea que como ya explicamos allí obtendrá la dirección de la variable para pasársela como argumento a otra funcion o api que la inicializará, vayamos allí.

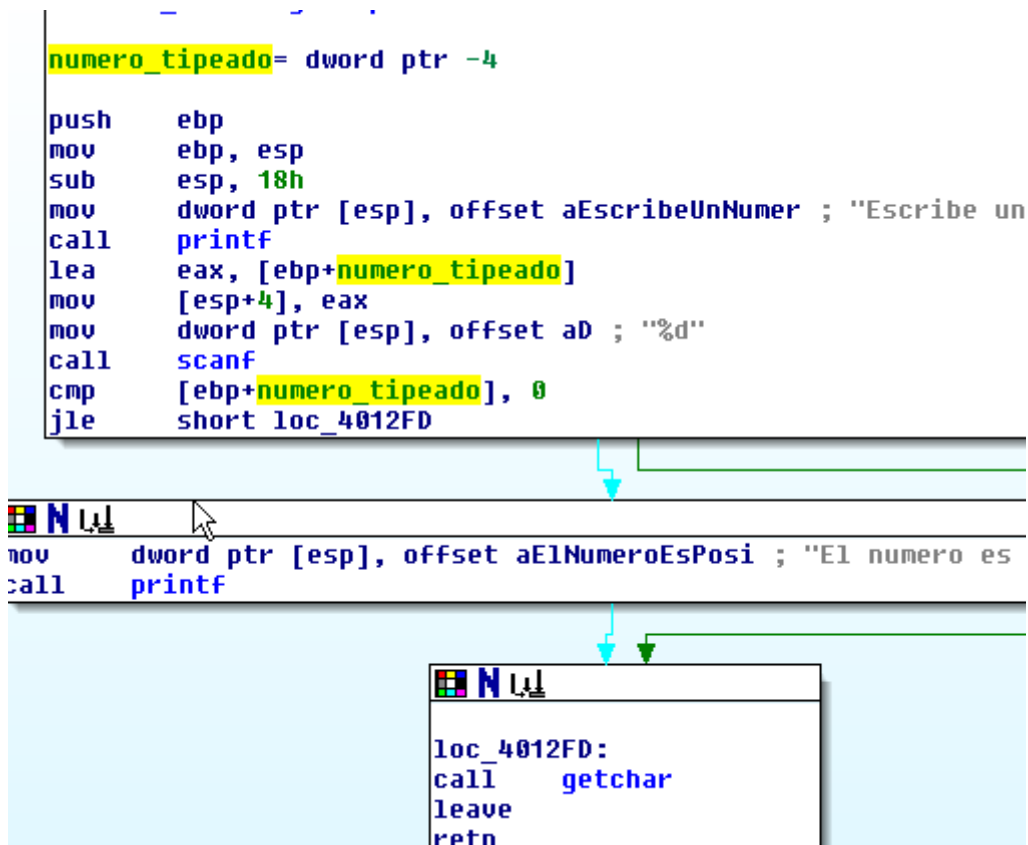
```

lea    eax, [ebp+var_4]
mov     [esp+4], eax
mov     dword ptr [esp], offset aD ; "%d"
call    scanf

```

Como vemos **EAX** obtendrá mediante el **LEA** la dirección de dicha variable **int**, y la misma se inicializará con el valor que tipee el usuario usando **scanf**, el mismo será un entero ya que como argumento de la api se usa **%d**.

Así que sabemos que la variable **int** creada sirve para eso, así que pongamosle un nombre acorde a su uso.



Como dicho **int** al no aclarar nada es signed, puede ser positivo o negativo, por ello el if que colocamos en el código el compilador lo transforma en las dos instrucciones siguientes que contienen la misma condición de verificar si el numero es mayor que 0 y decidir.

Aquí compara si es mayor que 0.

```

cmp     [ebp+numero_tipeado], 0

```

y en la siguiente linea se tomara la decisión de saltar o no según el resultado de esta comparación, vemos que el compilador elijio de todos los saltos condicionales posibles el **JLE**.

```

jle     short loc_4012FD

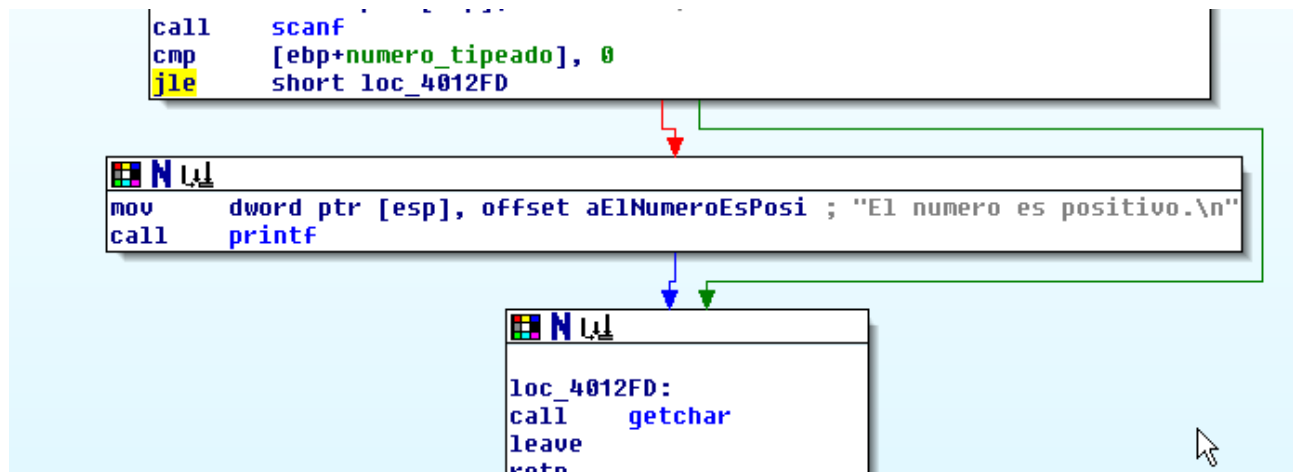
```

Los que no conocen de saltos condicionales pueden consultar

<http://moisesrbb.tripod.com/unidad5.htm#u5111>

Allí verán que el salto **JLE** toma en cuenta el signo al comparar, por lo cual a todos los efectos saltara si es menor o igual a cero, y no saltara si es positivo o sea mayor que cero.

Vemos en la imagen que el IDA nos muestra una bifurcación en el flujo del programa, que nos da la idea de que puede ir por estos dos caminos, que son las flechas, una verde que muestra el camino si la comparación es cierta o sea es negativo y entonces saltea el printf y va directamente al getchar, y la roja es el camino si la comprobación es falsa o sea no es negativo, o sea es positivo, con lo cual no salta y sigue el flujo del programa por el printf donde saca el mensaje de que el numero es positivo.



Una vez que vemos que la funcion sirve para determinar si el numero tipeado es positivo, cambiamos el nombre de la misma.

```
, attributes: bp-based frame

numero_tipeado_es_positivo proc near
numero_tipeado= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     dword ptr [esp], offset aEscribeUnNumer ; "E
call    printf
lea     eax, [ebp+numero_tipeado]
mov     [esp+4], eax
mov     dword ptr [esp], offset aD ; "%d"
call    scanf
cmp     [ebp+numero_tipeado], 0
jle     short loc_4012FD
```

El HexRays es una herramienta que nos sirve como apoyo, veamos como interpreta el código fuente de nuestra función.

```
int __cdecl numero_tipeado_es_positivo()
```

```

{
  int numero_tipeado; // [sp+14h] [bp-4h]@1

  printf("Escribe un numero: ");
  scanf("%d", &numero_tipeado);
  if ( numero_tipeado > 0 )
    printf("El numero es positivo.\n");
  return getchar();
}

```

Vemos que en este caso la interpretación es parecida.

Si cambiamos un poco el código y el agregamos un **else** para que si es negativo imprima un cartel de ello..

```

#include <stdio.h>

main(){
  funcion2();
  getchar();
}

funcion2(){

  int numero;

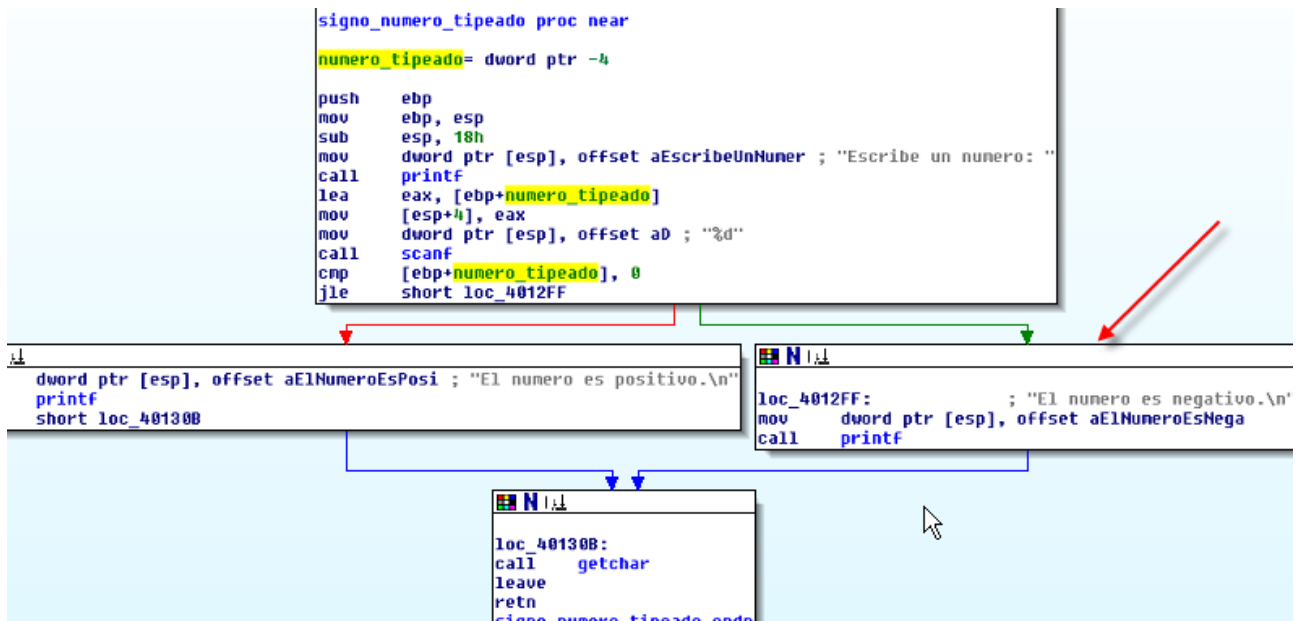
  printf("Escribe un numero: ");
  scanf("%d", &numero);
  if (numero>0) printf("El numero es positivo.\n");
  else printf("El numero es negativo.\n");
  getchar();
}

```

**if-else** respeta la comparación inicial que se realiza en el **if**, si esta es verdadera, ejecuta lo que esta al lado del if (o entre llaves si son varias instrucciones), si es falsa ejecuta lo que esta al lado del **else** ( o entre llaves si son varias instrucciones)

De esta forma si el numero es negativo va a ejecutar el printf que mostrara “**El numero es negativo**”

No lo reversearemos de nuevo pero veamos en IDA la diferencia.



La comparación es similar no cambio nada, lo único que en el camino de la flecha verde o sea si la comparación es verdadera (numero negativo) en vez de ir directamente al **getchar**, ahora agrego un bloquecito que imprime el mensaje de que **El numero es negativo** y luego si va al **getchar**.

En las comparaciones se utilizan los operadores relacionales siguientes:

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No igual a (distinto de)

Si alguien tiene duda de esto por supuesto puede profundizar en la pagina 3 del curso de Cabanes.

<http://www.nachocabanes.com/c/curso/cc03.php>

A continuación veremos un **switch** en el código siguiente, como siempre si quieren una completa y perfecta explicación de como funciona un switch en C, pues ahí esta en la pagina 3 del curso de Cabanes, aquí daremos una muy somera.

El switch se utiliza cuando hay una gran cantidad de posibilidades de bifurcación, seria como si evaluamos con muchos if uno tras otro, por ejemplo si un numero es cero, toma este camino, si es

uno otro camino, si es dos, otro camino, esto haría el código engorroso si lo hacemos con muchos **if**, de esta forma el **switch** evaluará una sola vez una expresión y según su resultado sigue el camino correspondiente.

Veamos el código así se entiende mas fácilmente.

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

```
main(){  
    funcion2();  
    getchar();  
    getchar();  
    getchar();  
}
```

```
funcion2(){
```

```
    char tecla;
```

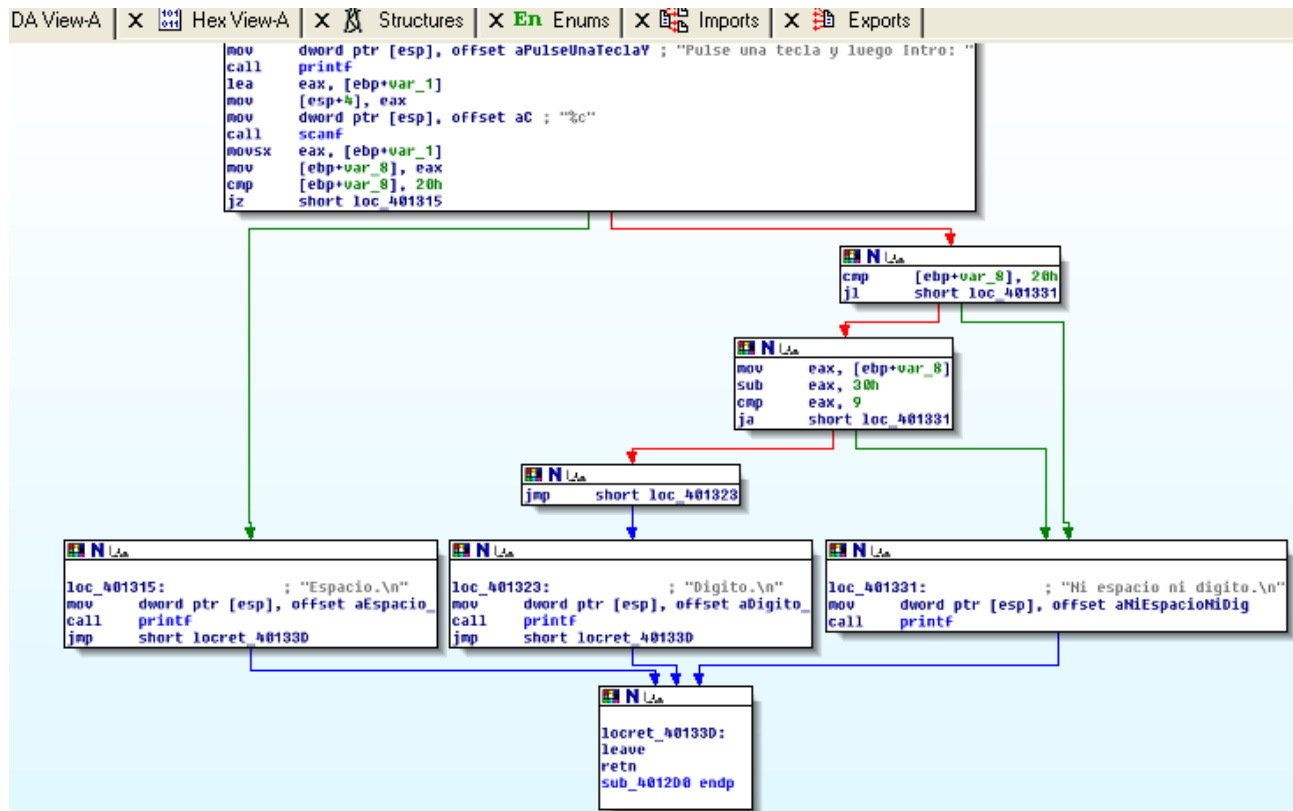
```
    printf("Pulse una tecla y luego Intro: ");  
    scanf("%c", &tecla);  
    switch (tecla)  
    {  
        case ' ': printf("Espacio.\n");  
                break;  
        case '1':  
        case '2':  
        case '3':  
        case '4':  
        case '5':  
        case '6':  
        case '7':  
        case '8':  
        case '9':  
        case '0': printf("Digito.\n");  
                break;  
        default: printf("Ni espacio ni digito.\n");  
    }
```

}

Se escribe a continuación de “**switch**” la expresión a analizar, entre paréntesis. Después, tras varias órdenes “**case**” se indica cada uno de los valores posibles. Los pasos (porque pueden ser varios) que se deben dar si se trata de ese valor se indican a continuación, terminando con “**break**”. Si hay que hacer algo en caso de que no se cumpla ninguna de las condiciones, se detalla tras “**default**”.

Vemos que los casos que no tienen **break** continúan ejecutando el siguiente que viene debajo pues no salen, tal el caso de si aprieto **1**, entra al case **1**, como no hay código ni break sigue al 2 y así hasta abajo que muestra el cartel **Digito** y si sale porque hay un **break**, veámoslo en IDA a ver como se ve el **switch**.

Vemos en el IDA la estructura que es un poco mas compleja.



Bueno vamos por partes, vemos abajo tres posibilidades ya que realmente en el código C vimos que o va a **Espacio**, o va a **Digito** o va a **Ni espacio ni digito**




```

printf("Pulse una tecla y luego Intro: ");
scanf("%c", &tecla);
switch (tecla)
{
    case ' ': printf("Espacio.\n");
               break;

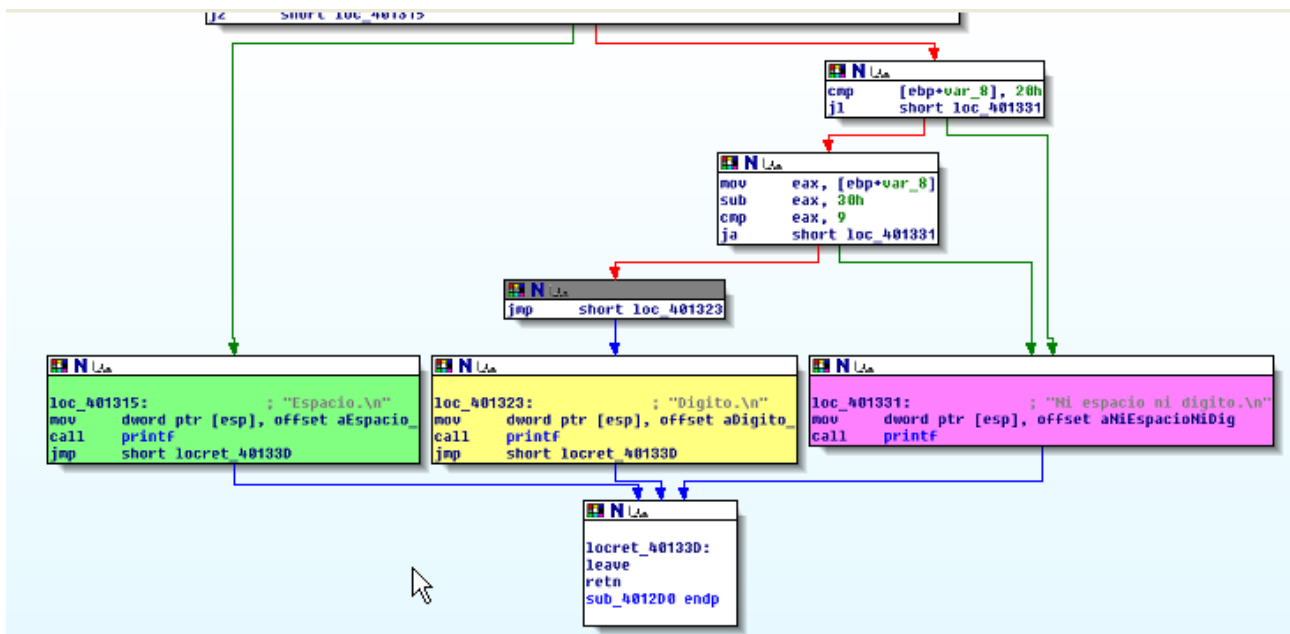
    case '1':
    case '2':
    case '3':
    case '4':
    case '5':
    case '6':
    case '7':
    case '8':
    case '9':
    case '0': printf("Digito.\n");
               break;

    default:   printf("Ni espacio ni digito.\n");
}

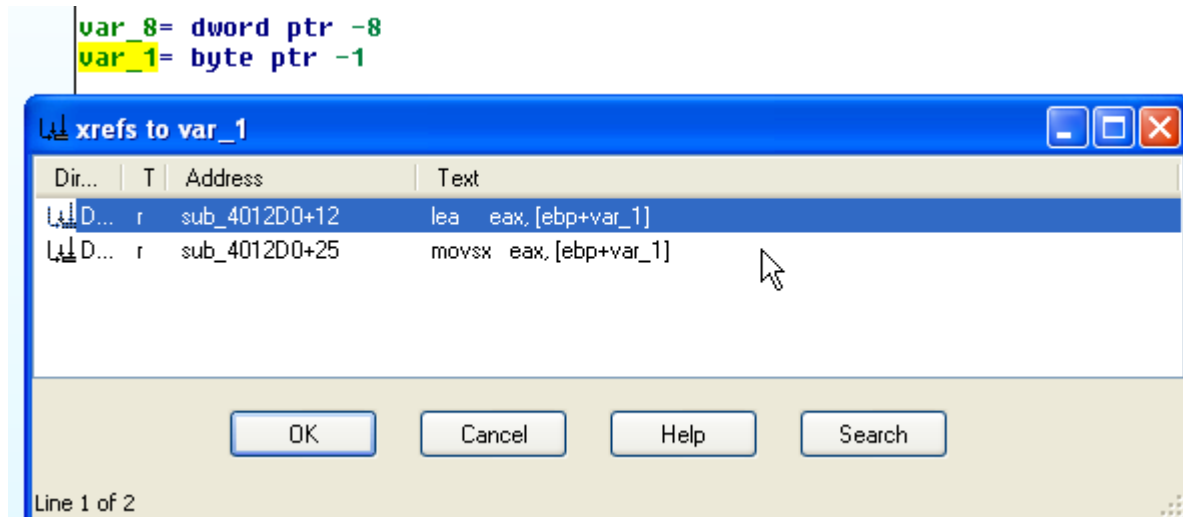
```



Los tres casos están claros en el IDA, puedo pintar de colores los bloques, que también es otra ayuda para reversear así los vemos mas resaltados.



Renombro la funcion y vemos que hay dos variables, yo solo use una del tipo **char** que ocupa solo un byte, la otra la agrego el compilador, así que si vemos la variable de 1 byte.



Vemos el LEA que obtiene la dirección de la variable para guardar el carácter que tipea el usuario, vemos que `%c` hace que la guarde como carácter al pasar `scanf`.

```

funcion_investigada proc near
    var_8= dword ptr -8
    caracter_tipeado= byte ptr -1

    push    ebp
    mov     ebp, esp
    sub     esp, 18h
    mov     dword ptr [esp], offset aPulseUnaTecla
    call    printf
    lea     eax, [ebp+caracter_tipeado]
    mov     [esp+4], eax
    mov     dword ptr [esp], offset aC ; "%c"
    call    scanf
    movsx   eax, [ebp+caracter_tipeado]
    mov     [ebp+var_8], eax
    cmp     [ebp+var_8], 20h
    jz      short loc_401315

```

A continuación usa la variable dword que creo, para tomar el valor de nuestro carácter y transformarlo en un dword usando **MOVSX** lo cual al ser un carácter tipeado podría haber comparado directamente el byte original, seguramente alguna causa de economía de código.

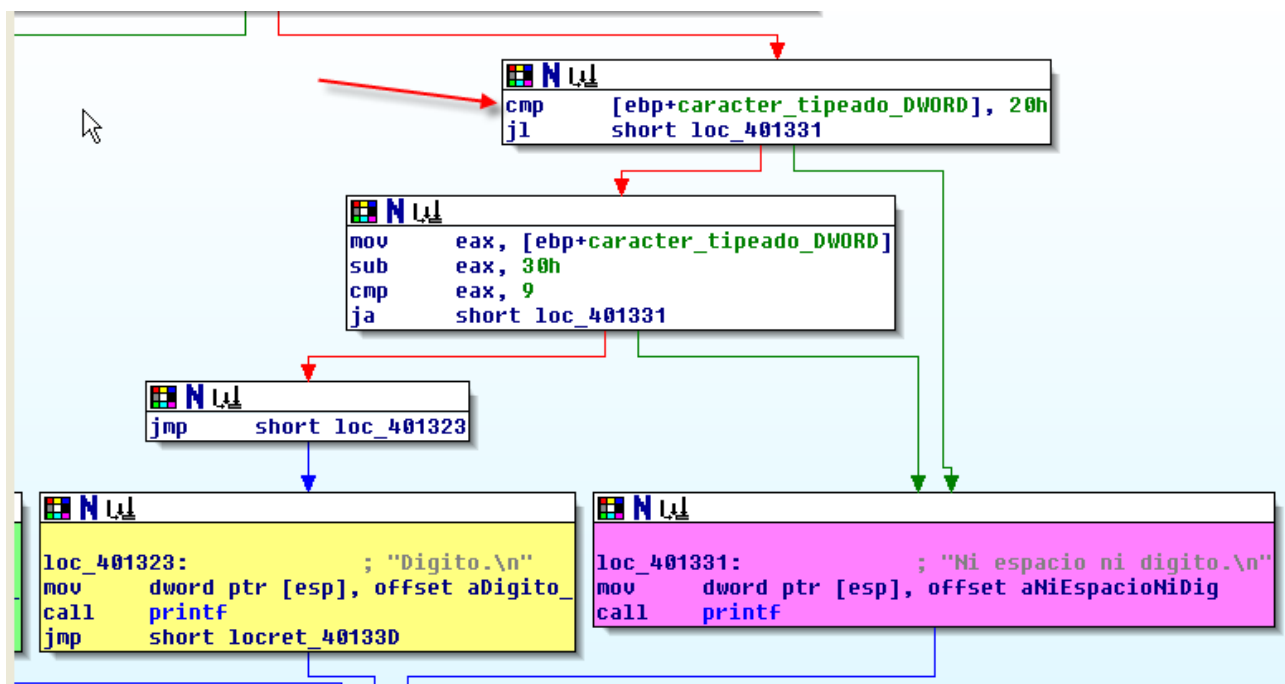
```

movsx    eax, [ebp+caracter_tipeado]
mov      [ebp+var_8], eax

```

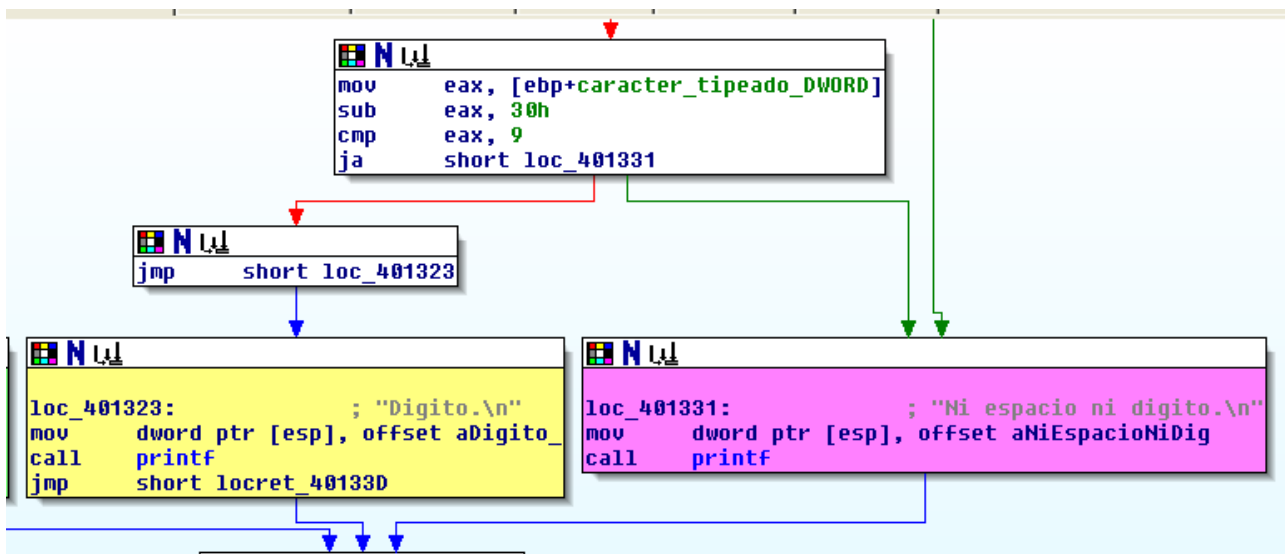
Así que renombramos esta otra variable dword.





Vemos que compara nuevamente con 20 lo cambiamos también a espacio., si es menor que 20 no es ni espacio ni dígito por lo cual el JL me lleva al bloque rosado, con lo cual ya tenemos dos de las tres posibilidades del switch.

Si no es menor que 20 seguirá por la flecha roja a la siguiente comparación.



Allí le resta 30 al valor tipeado y lo compara con nueve y si es mas alto JA nos manda al bloque rosado ya que seria mas alto que 39 hexa que es el carácter 9, si es menor va al cartel amarillo que imprime **Dígito** pues se supone que es un valor entre 30 y 39 o sea los caracteres del 0 al 9.

Los caracteres con valor hexa del 21 al 30 son suprimidos también pues al restar con 30 queda un numero alto mas grande que 9 ya que el JA no considera signo, así que el resultado negativo producto de esa resta para el JA son solo valores positivos mas grandes que 9 y saltan al bloque rosado.

Veamos este otro switch

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

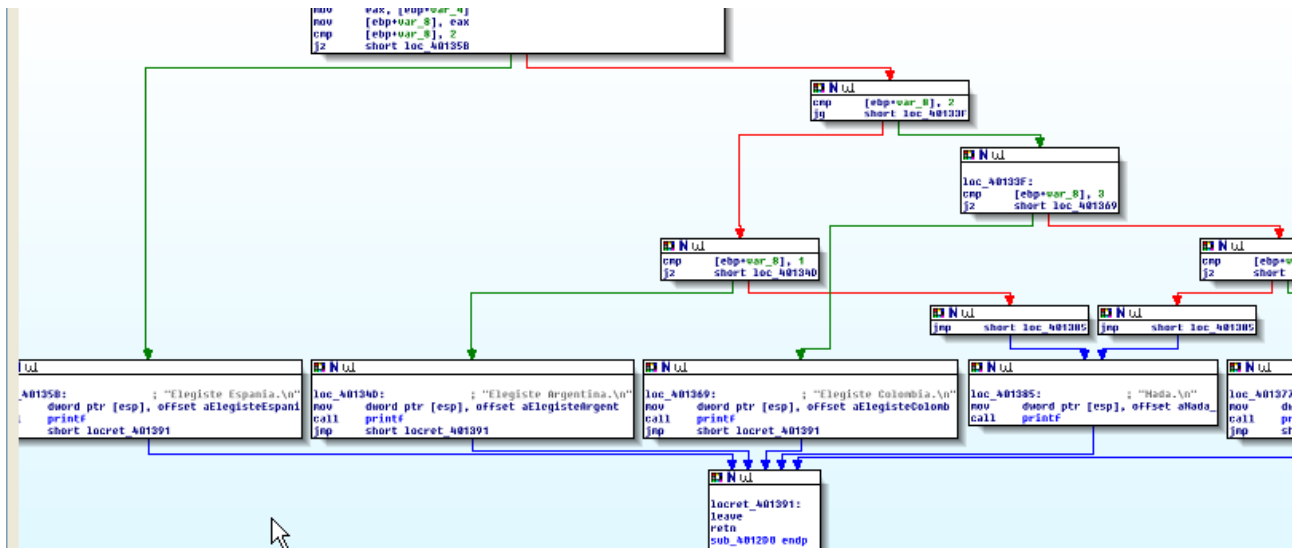
```
main(){  
    funcion2();  
    getchar();  
    getchar();  
    getchar();  
}
```

```
funcion2(){
```

```
    int opcion;
```

```
    printf("Donde nacio?\n");  
    printf("1: Argentina\n");  
    printf("2: Espania\n");  
    printf("3: Colombia\n");  
    printf("4: Uruguay\n");  
    scanf("%d", &opcion);  
    switch (opcion)  
    {  
        case 1: printf("Elegiste Argentina.\n");  
                break;  
        case 2: printf("Elegiste Espania.\n");  
                break;  
        case 3: printf("Elegiste Colombia.\n");  
                break;  
        case 4: printf("Elegiste Uruguay.\n");  
                break;  
  
        default: printf("Nada.\n");  
    }  
}
```

Vemos aquí un **switch** con un valor numérico que proviene del scanf y se guarda como **int** en la variable **opción**.



Vemos que es similar solo que en este caso compara directamente con el entero que tipeamos aunque eso no evita que el compilador cree una nueva variable y copie el contenido de lo tipeado de la nuestra a esa, sera que cuando usa switch lo debe hacer con una variable mas vaya a saber, la cuestión es que es similar al caso anterior, pueden analizarlo si quieren.

Aquí el código que nos devuelve el HexRays muy parecido al original

```
int __cdecl sub_4012D0()
{
    int v1; // [sp+14h] [bp-4h]@1

    printf("Donde nacio?\n");
    printf("1: Argentina\n");
    printf("2: Espania\n");
    printf("3: Colombia\n");
    printf("4: Uruguay\n");
    scanf("%d", &v1);
    if ( v1 == 2 )
        return printf("Elegiste Espania.\n");
    if ( v1 > 2 )
    {
        if ( v1 == 3 )
            return printf("Elegiste Colombia.\n");
        if ( v1 == 4 )
            return printf("Elegiste Uruguay.\n");
    }
}
```

```

}
else
{
    if ( v1 == 1 )
        return printf("Elegiste Argentina.\n");
    }
    return printf("Nada.\n");
}

```

Aquí vemos un ejemplo usando **while**, el cual se repite mientras la condición sea verdadera o sea mientras que el numero que tipeamos sea distinto de cero seguirá loopeando y saldrá del ciclo cuando sea igual a cero.

```

# include <stdio.h>

main(){
    funcion2();
}

funcion2(){
    int numero;

    printf("Teclea un numero (0 para salir): ");
    scanf("%d", &numero);
    while (numero!=0)
    {
        if (numero > 0) printf("Es positivo\n");
        else printf("Es negativo\n");
        printf("Teclea otro numero (0 para salir): ");
        scanf("%d", &numero);
    }
}

```

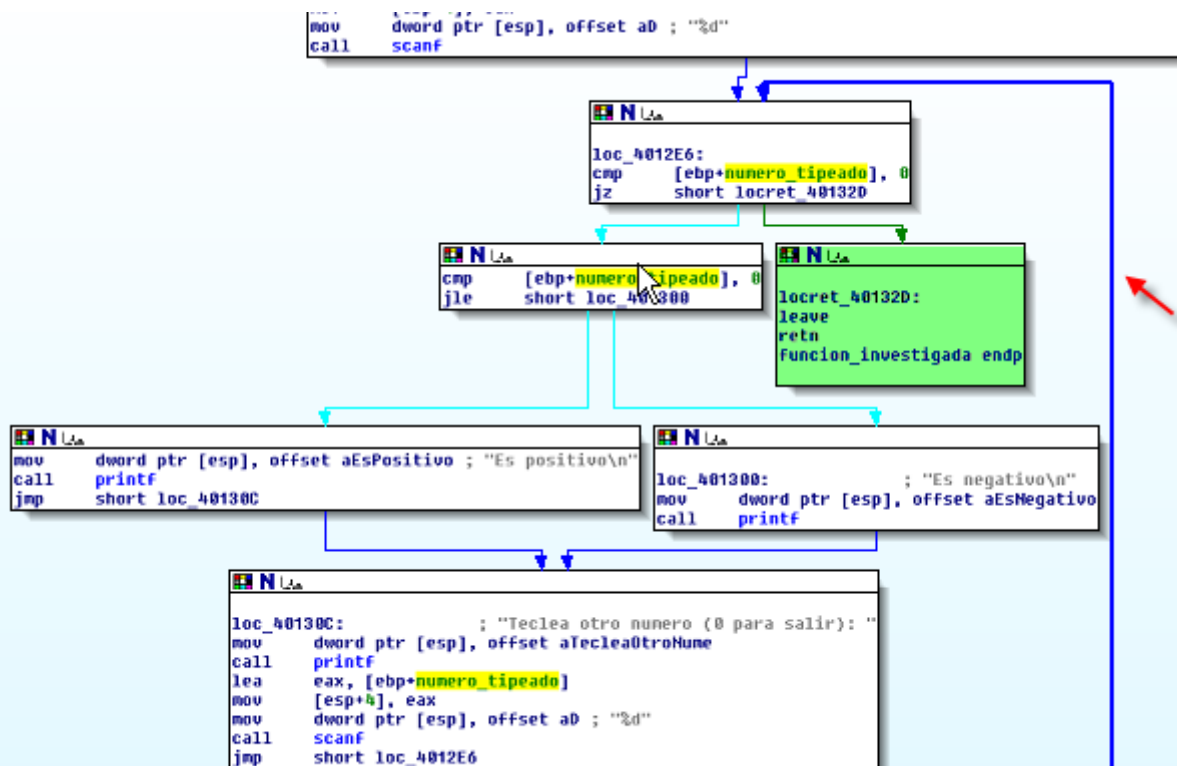
Veamoslo en el IDA.

```
; Attributes: bp-based frame

funcion_investigada proc near
numero_tipeado= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     dword ptr [esp], offset aTecleaUnNumero ; "Teclea un numero (0 para salir): "
call    printf
lea     eax, [ebp+numero_tipeado]
mov     [esp+4], eax
mov     dword ptr [esp], offset aD ; "%d"
call    scanf
```

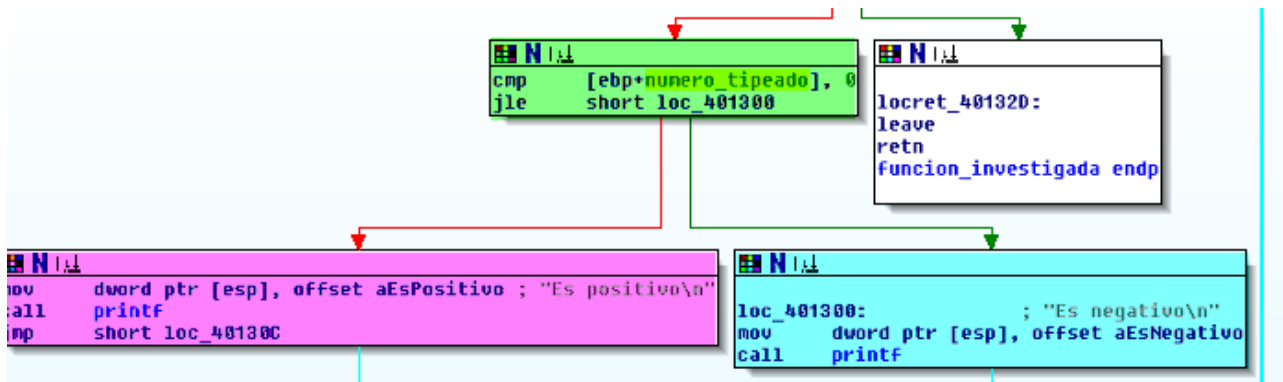
Hasta aquí nada nuevo ya renombre la variable y como hemos visto el lea obtiene la dirección de la misma que contendrá el numero tipeado por el usuario usando scanf.



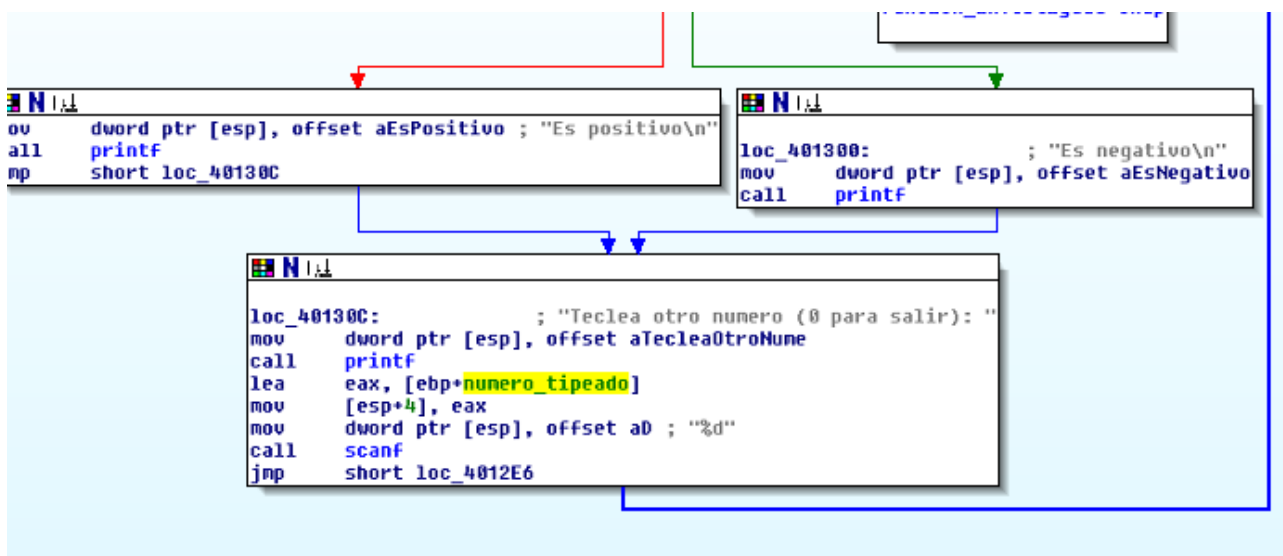
Allí vemos el ciclo que se repetirá indefinidamente a no ser que la variable `numero_tipeado` valga cero en ese caso ira al bloque verde que es la salida de mi funcion a través del `retn`.

Dentro del loop hay una comparación para ver si el numero es positivo o negativo y según eso va al bloque rosado o al celeste imprimiendo el cartel correspondiente.





Luego vuelve a leer otro valor tipeado por el usuario y a guardarlo nuevamente en la variable numero\_tipeado y vuelve al inicio del ciclo donde chequea si es cero para salir del mismo o seguirá repitiéndolo.



El código fuente de la funcion que nos da el HexRays es el siguiente salvo que crea una variable mas para el valor de retorno de scanf, que nosotros no creamos.

```

int __cdecl funcion_investigada()
{
    int result; // eax@1
    int numero_tipeado; // [sp+14h] [bp-4h]@1

    printf("Teclea un numero (0 para salir): ");
    result = scanf("%d", &numero_tipeado);
    while ( numero_tipeado )
    {
        if ( numero_tipeado <= 0 )
            printf("Es negativo\n");
        else

```

```

    printf("Es positivo\n");
    printf("Teclea otro numero (0 para salir): ");
    result = scanf("%d", &numero_tipeado);
}
return result;
}

```

El siguiente es un ejemplo con **for**

```

#include <stdio.h>

main(){
    funcion2();
    getchar();
}

funcion2(){

    int contador;

    for (contador=1; contador<=10; contador++)
        printf("%d ", contador);

}

```

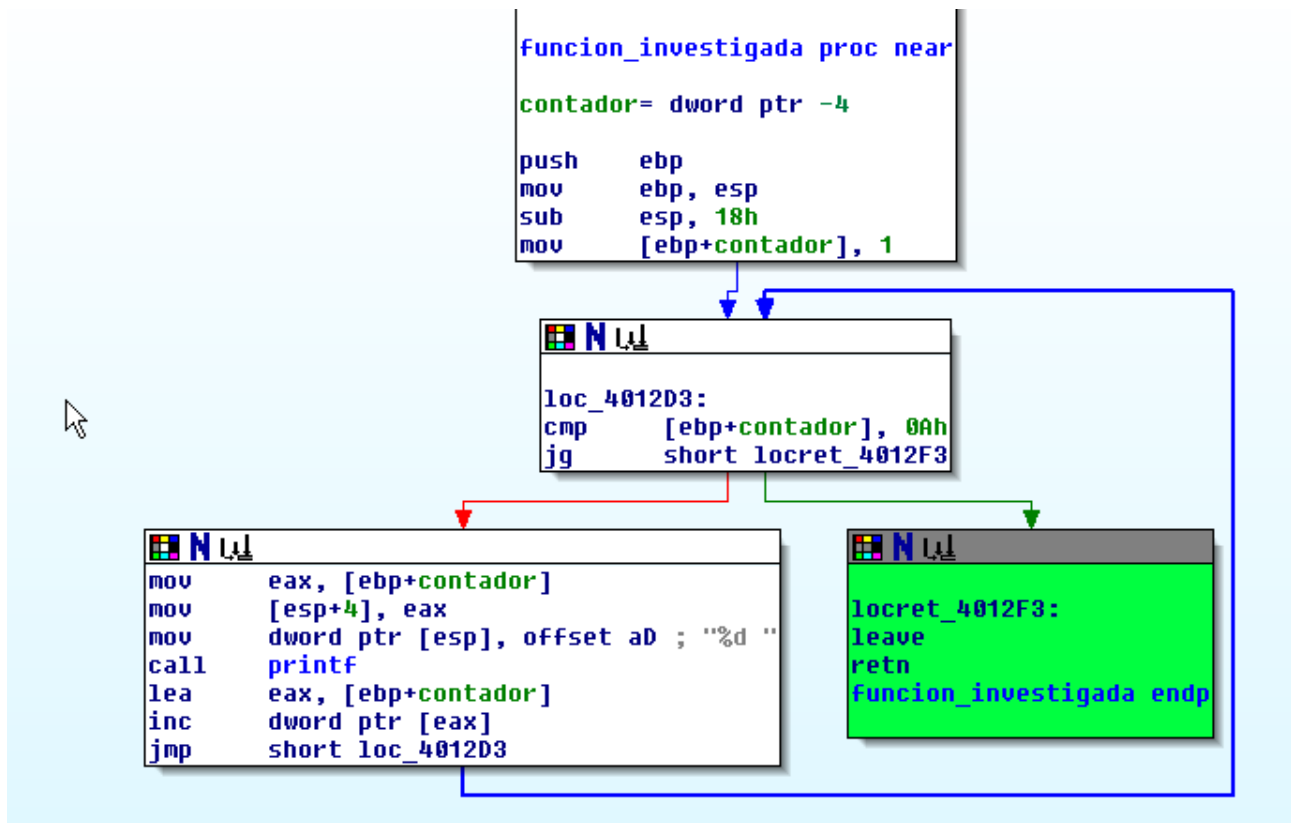
Es un ciclo que tiene como primer argumento el valor donde se inicia el contador, como segundo la comparación para chequear si continua looppeando, y el tercero es el incremento del contador.

En este caso el contador empezara desde 1 inclusive seguirá looppeando mientras sea menor o igual a 10, incrementándose de a uno.

Como dentro del loop se imprime el valor del mismo contador, pues imprimirá del 1 al 10 en este caso.



Verlo en IDA es muy sencillo, la variable que renombramos como contador, se inicializa en el valor 1, y se compara si es mayor que 10 (0a hexa) para ver si sale del ciclo y va al bloque verde.



Si es menor continua en el ciclo e imprime su valor y lo incrementa obteniendo en EAX su dirección mediante lea y incrementando el contenido de EAX o sea el mismo valor del contador.

```

mov     eax, [ebp+contador]
mov     [esp+4], eax
mov     dword ptr [esp], offset aD ; "%d "
call    printf
lea     eax, [ebp+contador]
inc     dword ptr [eax]
jmp     short loc_4012D3
  
```

Adjunto tres ejemplos de ejecutables con **goto**, **continue** y **break** metidos en el medio, a ver si los pueden reversear sin tener el código fuente, cualquier duda ya saben que **break** sale del ciclo, **continue** vuelve al inicio del ciclo sin resetear el contador y **goto** va a una etiqueta como un salto directo a la misma.

Bueno que tenga suerte y hasta la próxima

Ricardo Narvaja

Salute Crackslatinos