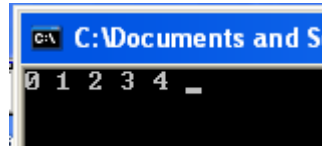


Soluciones de los ejercicios propuestos en la parte 3

Bastante triste porque solo una persona o bien lo intento o bien pudo solucionar, lo cual significa que este curso o no interesa o bien no explico bien, pero bueno explicare como reversear esos tres casos.

Comenzaremos con el ejemplo a, al ejecutarlo vemos:



Sabemos dado que la ayuda que nos dieron dice que uno usa **break**, otro **continue**, y otro **goto**.

Veamoslo en el IDA

```

;_main- dword ptr [ebp+var_4]
push    ebp
mov     ebp, esp
sub     esp, 8
and     esp, 0FFFFFF0h
mov     eax, 0
add     eax, 0Fh
add     eax, 0Fh
shr     eax, 4
shl     eax, 4
mov     [ebp+var_4], eax
mov     eax, [ebp+var_4]
call    ___chkstk@7C901206
call    __main@7C901206
call    sub_4012C6
call    getchar@7C901206
leave
retn
_main endp
```

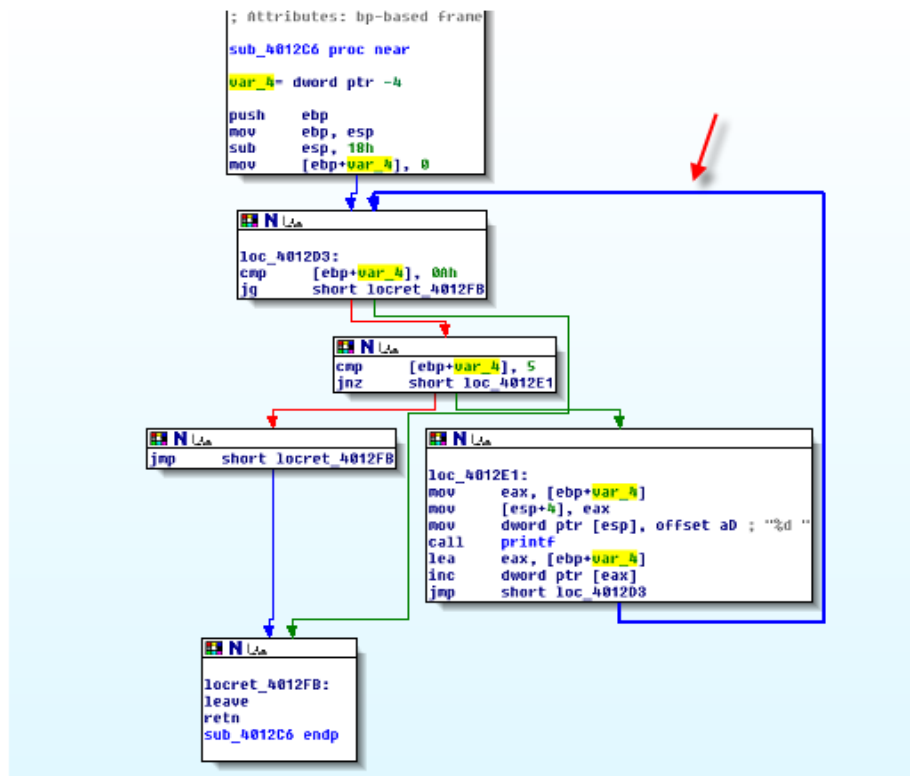
Vemos que el main en los tres casos es similar a los ejemplos que venimos estudiando, la funcion marcada en celeste no tiene argumentos, pues no hay pushes antes de la misma ni se guarda nada en el stack, y no tiene valor de retorno pues al volver de dicha funcion no hace nada, solo va a getchar y sale del programa al tipear ENTER, reconocemos en los tres casos el código agregado por el compilador para el **main**, por lo cual para los tres casos el main sera **asi**:

include <stdio.h>

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

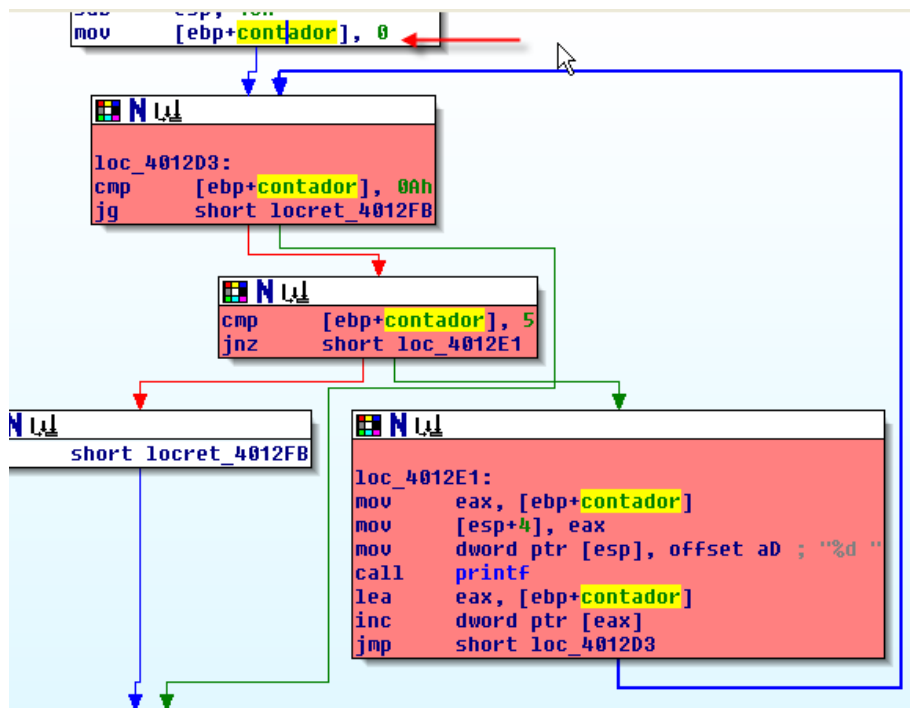
```
funcion(){
    .....
    .....
}
```

Solo faltara que rellenemos la parte de los puntos suspensivos con el código de nuestra función, entremos a la misma a ver que hace.



Vemos una funcion de una sola variable, aquí se confirma que no tiene argumentos si no estarían allí marcados por IDA debajo de las variables, hay un ciclo allí señalado con la flecha, ahora debemos ver como diferenciamos un **while** de un **for**.

Recordemos que en el **for**, antes de entrar al mismo se inicializaba un contador, que se comparaba al inicio para salir del ciclo y que al terminar cada ciclo variaba su valor según como hayamos declarado en el mismo **for**, este es el mismo caso, podemos renombrar a la variable como **contador** y vemos que se inicializa a cero antes de entrar, por lo que sera una variable **int** en nuestro código.



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

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

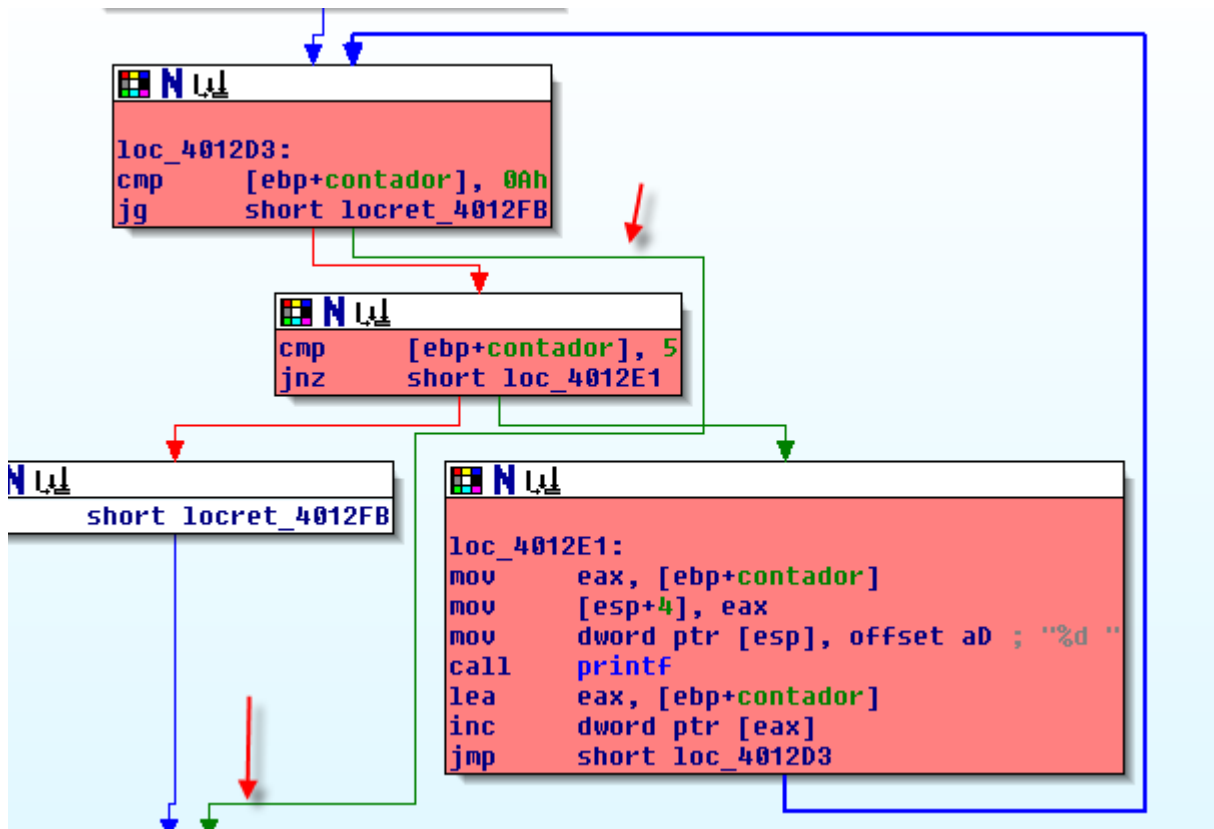
```
funcion(){
```

```
    int contador;
```

```
    for (contador=0...
```

```
}
```

Vemos que el **contador** en el for se inicializa a cero, así que vamos armando la función, lo siguiente en un **for** es el chequeo de la condición de salida que se realiza dentro del mismo ciclo pues la condición debe chequearse en cada repetición.



Vemos que si **contador** es mas grande que 10 sale por la flecha verde que marcamos en la imagen sino por el otro camino continua dentro del ciclo al siguiente bloque rosado y loopeando, así que sabemos que la condición de salida es que contador sea mas grande que 10, o sea que seguirá loopeando si es menor o igual a 10.

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

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

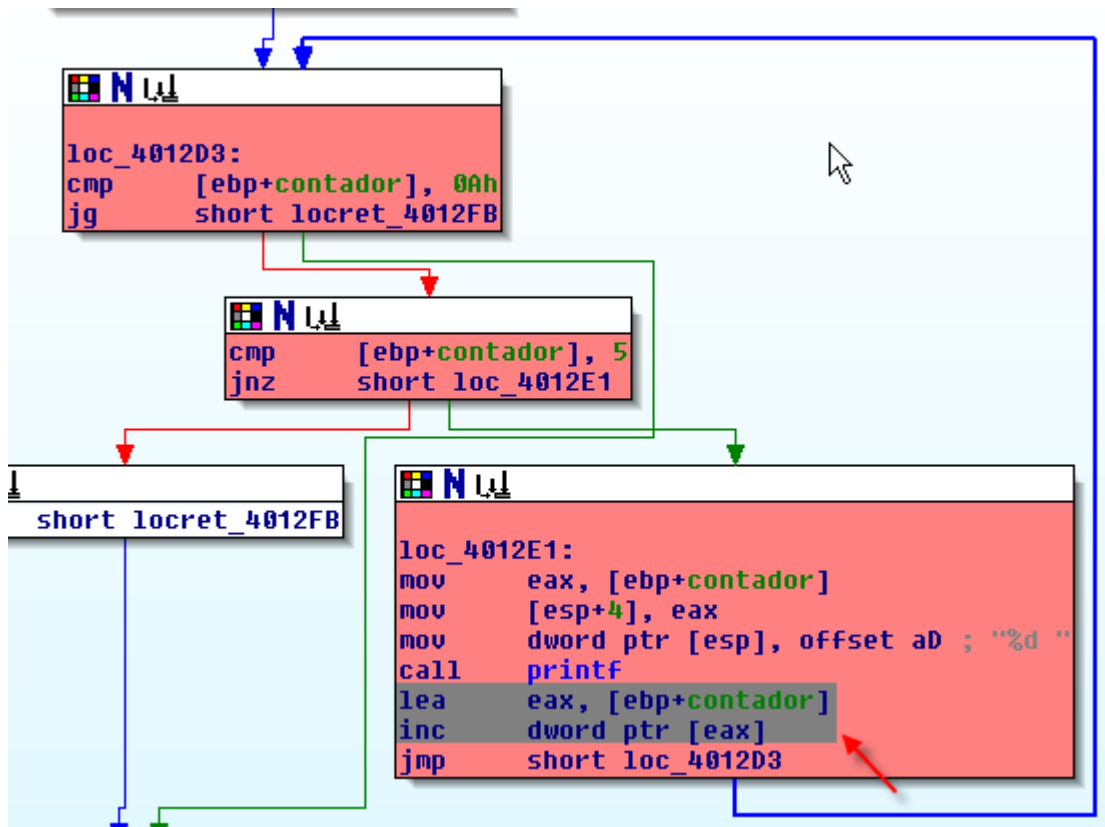
```
funcion(){
```

```
    int contador;
```

```
    for (contador=0; contador<=10 .....
```

```
}
```

Solo nos queda en la declaración del **for**, analizar como se incrementa la variable **contador**, vemos que antes de repetir el ciclo con el **lea** y el **inc** se incrementa de uno en uno.



Así que sabemos que se incrementa de uno en uno el código quedara:

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

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

```
funcion(){
```

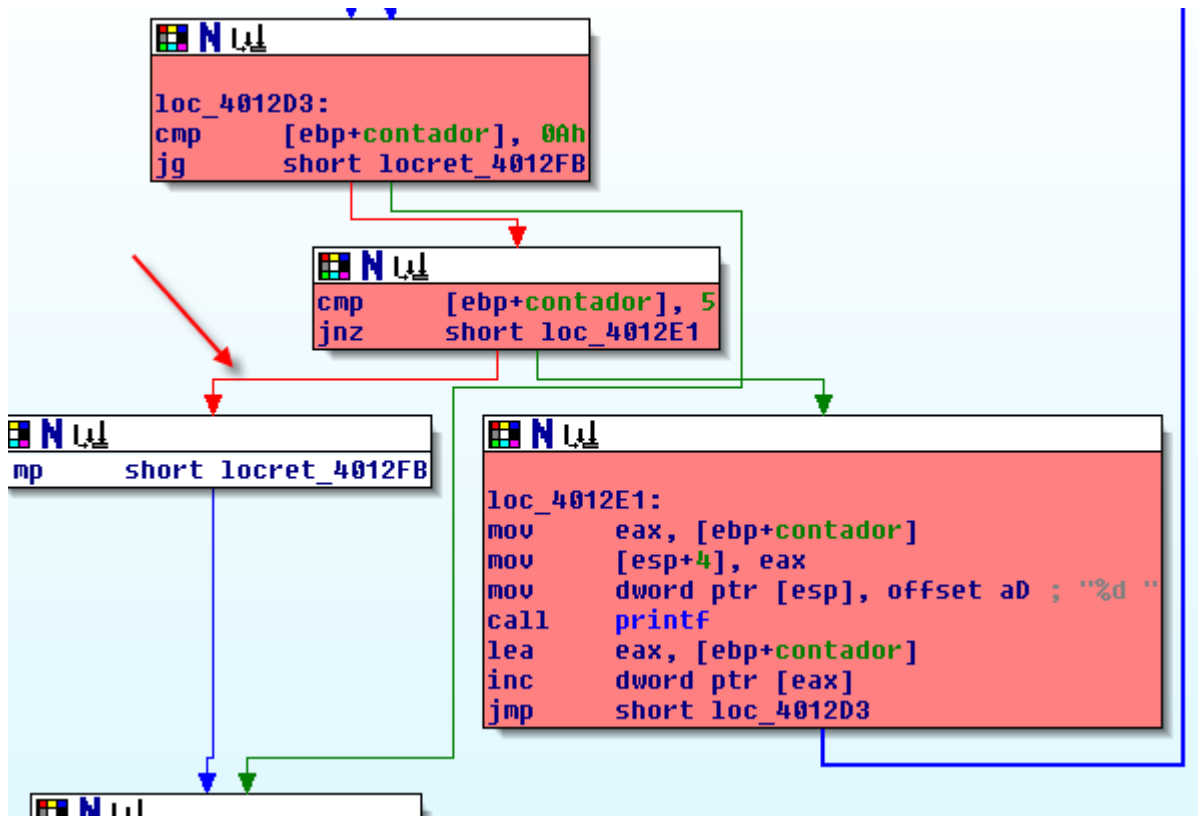
```
int contador;
```

```
for (contador=0; contador<=10;contador++)
```

```
{
.....
}
```

```
}
```

Así que ya tenemos el for completo solo nos falta ver el código que ejecuta dentro del mismo que ira dentro de las llaves.



Vemos que compara **contador** con 5 si no es cinco, continua al siguiente bloque rosado del loop, pero si es igual a cinco, fuerza la salida del mismo sin respetar que contador debería llegar hasta 10 para terminar el ciclo, así que por definición sabemos que este es el comportamiento del **break**.

```
if (contador==5) break;
```

Hará que al llegar al **break** cuando **contador** valga cinco, salga del loop sin importar la condición definida en la declaración del mismo que sería al llegar a valer 10.

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

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

```
funcion(){
```

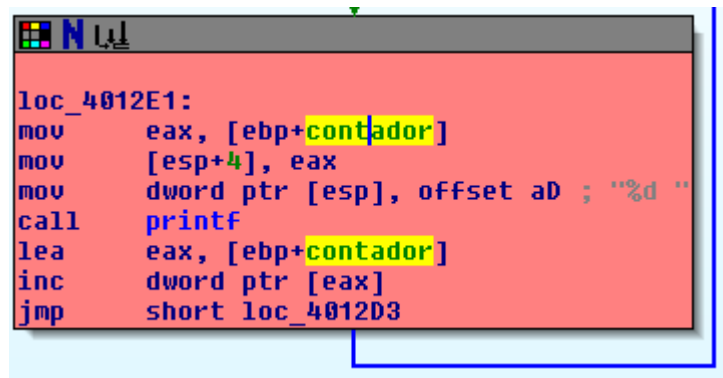
```
    int contador;
```

```
    for (contador=0; contador<=10;contador++)
```

```
    {
        if (contador==5) break;
        .....
    }
```

```
}
```

lo que nos queda ver es que pasa cuando contador no es igual a 5, si recordamos que el ejecutable imprimía 0 1 2 3 y 4, calculamos que eso hará imprimir el valor del contador de 0 a 4 y al llegar a cinco saldría mediante el break, veamos en el IDA,



Vemos que el ultimo bloque antes de volver al inicio del loop lo que hace es pasar el valor del **contador** como argumento para que **printf** lo imprima usando **%d** o sea imprimirá su valor numérico.

Así que eso es todo el código que nos falta.

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

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

```
funcion(){
```

```
    int contador;
```

```
    for (contador=0; contador<=10;contador++)
    {
        if (contador==5) break;
        printf("%d ",contador);
    }
```

```
}
```

Probemos a ver si el código compila y funciona como el original:

```

ejemplo 2.0
# include <stdio.h>

main(){
    funcion();
    getchar();
}

funcion

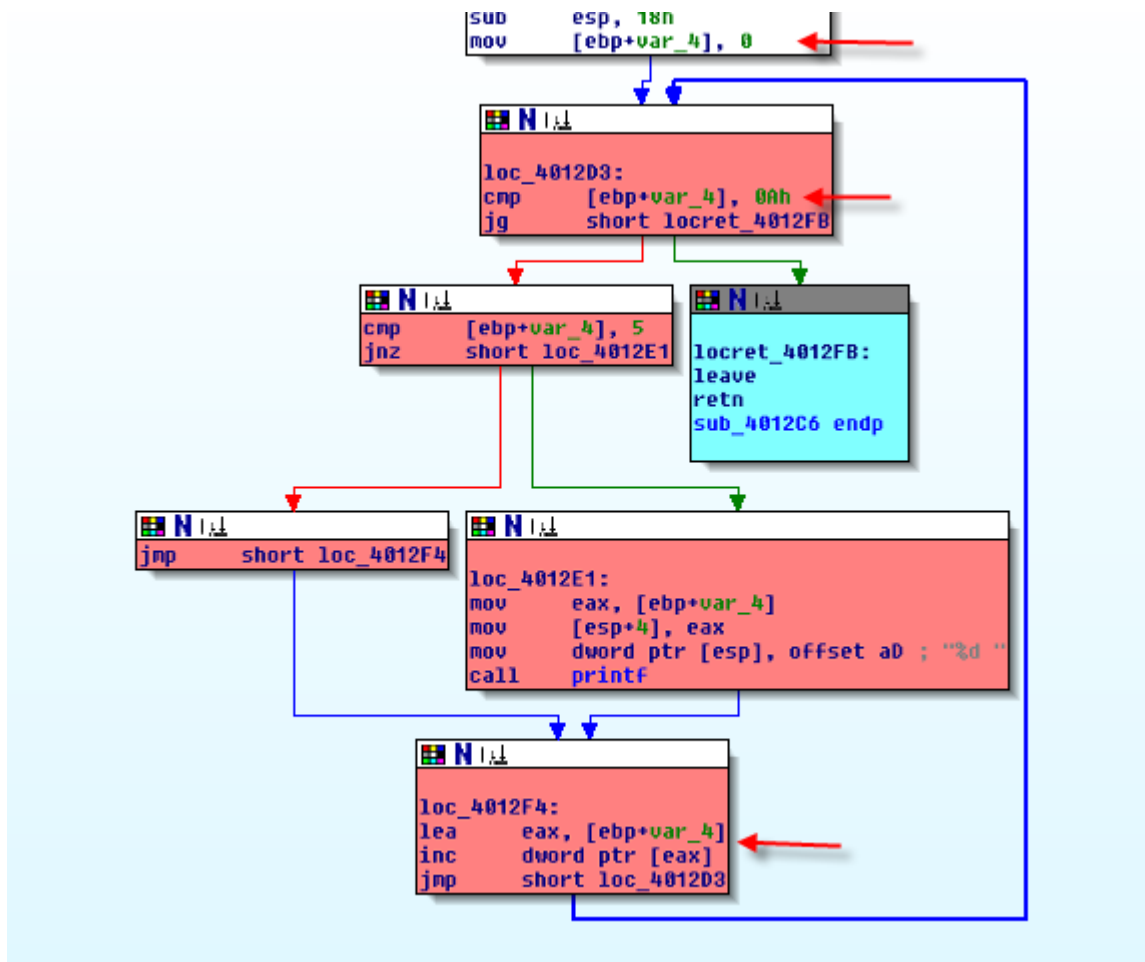
int con

for

```

Perfecto hemos reverseado el primero.

Veamos el segundo caso:



Vemos que el main es similar y nuestra funcion tiene una sola variable que es un int que se usa como contador de un for al igual que el caso anterior, vemos los bloques del ciclo en rosado y la salida en celeste.

Vemos en la imagen las tres flechas que indican la inicializacion de contador en cero, la salida del for cuando sea mayor que 10, y el incremento de a uno lo mismo que en el caso anterior, así que podríamos escribir el código:


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

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

```
funcion(){
```

```
    int contador;
```

```
    for (contador=0; contador<=10;contador++)
```

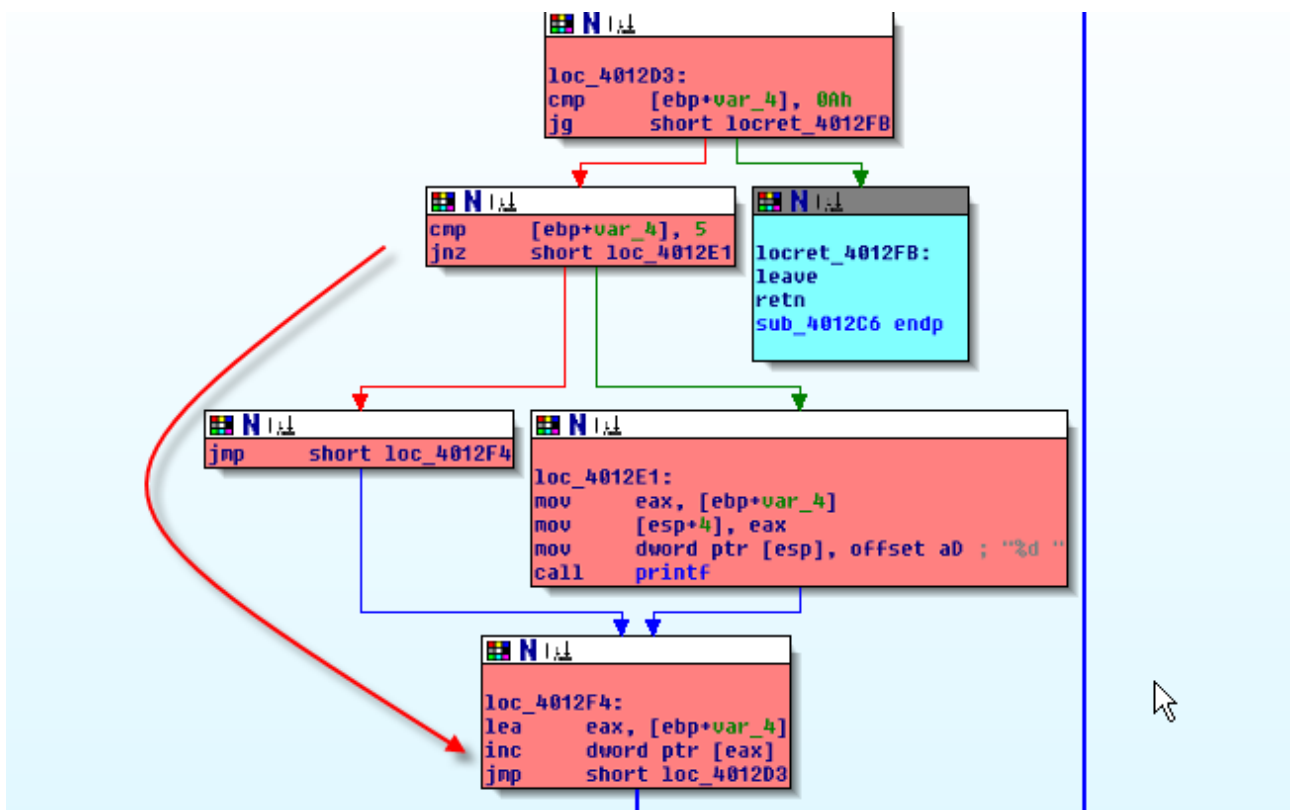
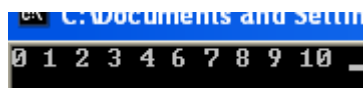
```
    {
```

```
        .....
```

```
    }
```

```
}
```

digamos que el for es similar, vemos cuando lo ejecutamos que imprime del 0 al 10 pero el cinco lo saltea, eso es exactamente lo que hace la instrucción **continue** que fue la otra de que nos dijeron en la ayuda que podía utilizarse, la misma cuando se llega a ella, vuelve al inicio del ciclo incrementa el contador, pero no ejecuta el código a continuación, que es lo que vemos aquí en el IDA.



Cuando el **contador** vale cinco saltea el resto del código dentro del loop evitando el **printf** y se

incrementa y vuelve a repetirse como si no hubiera pasado nada, de esta forma imprime todos los números del 0 al 10, pero el cinco no.

```
if (contador==5) continue;  
printf("%d ", contador);
```

el código completo seria:

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

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

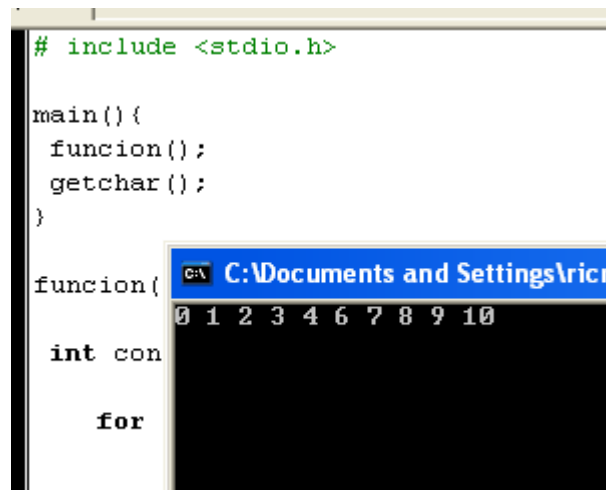
```
funcion(){
```

```
int contador;
```

```
for (contador=0; contador<=10;contador++)  
{  
    if (contador==5) continue;  
    printf("%d ", contador);  
}
```

```
}
```

Compilemoslo y probemos a ver si funciona como el original:



The screenshot shows a C program being compiled and run. The code in the background is the same as the one provided in the text. The output window, titled 'C:\Documents and Settings\ricn', displays the numbers 0 1 2 3 4 6 7 8 9 10, confirming that the number 5 was skipped due to the 'continue' statement.

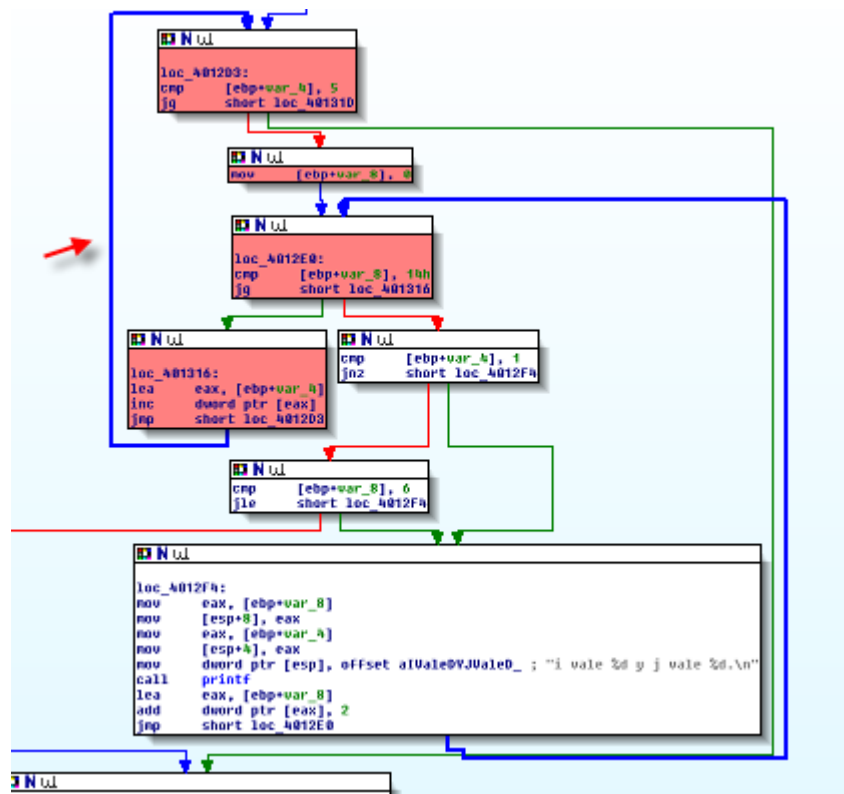
Perfecto vamos al tercer ejemplo.

```

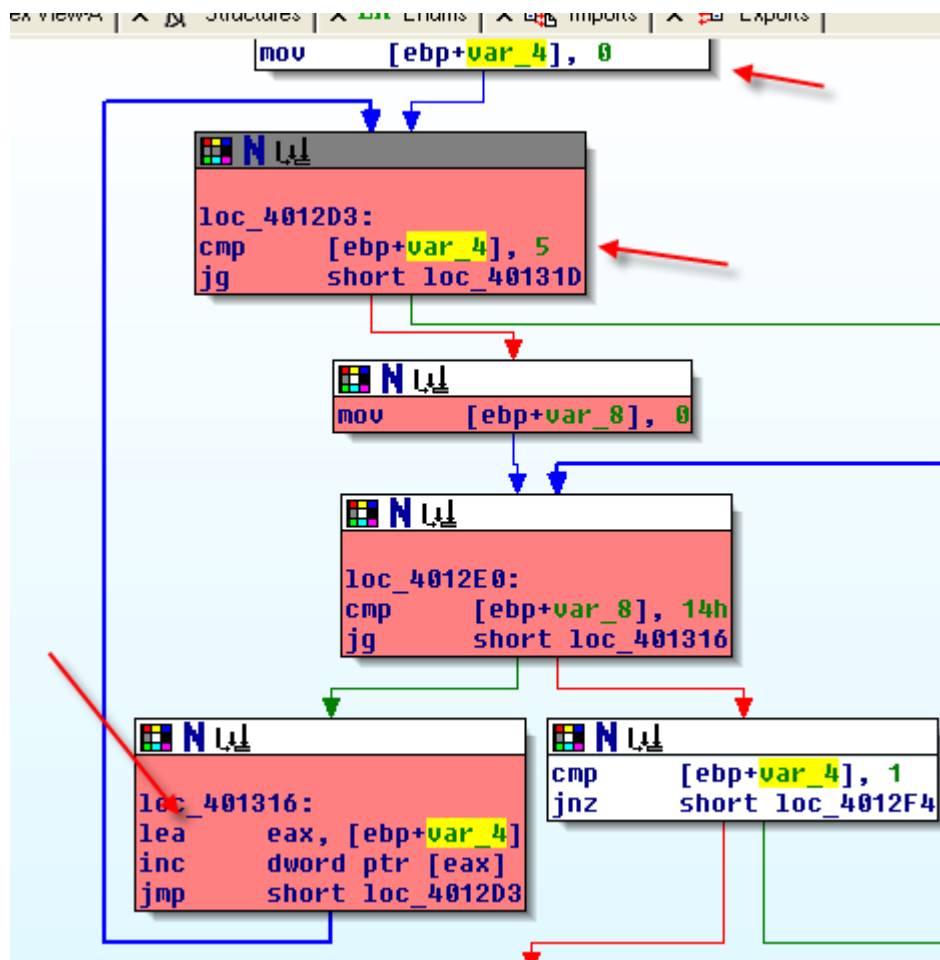
i vale 0 y j vale 0.
i vale 0 y j vale 2.
i vale 0 y j vale 4.
i vale 0 y j vale 6.
i vale 0 y j vale 8.
i vale 0 y j vale 10.
i vale 0 y j vale 12.
i vale 0 y j vale 14.
i vale 0 y j vale 16.
i vale 0 y j vale 18.
i vale 0 y j vale 20.
i vale 1 y j vale 0.
i vale 1 y j vale 2.
i vale 1 y j vale 4.
i vale 1 y j vale 6.
Fin del programa

```

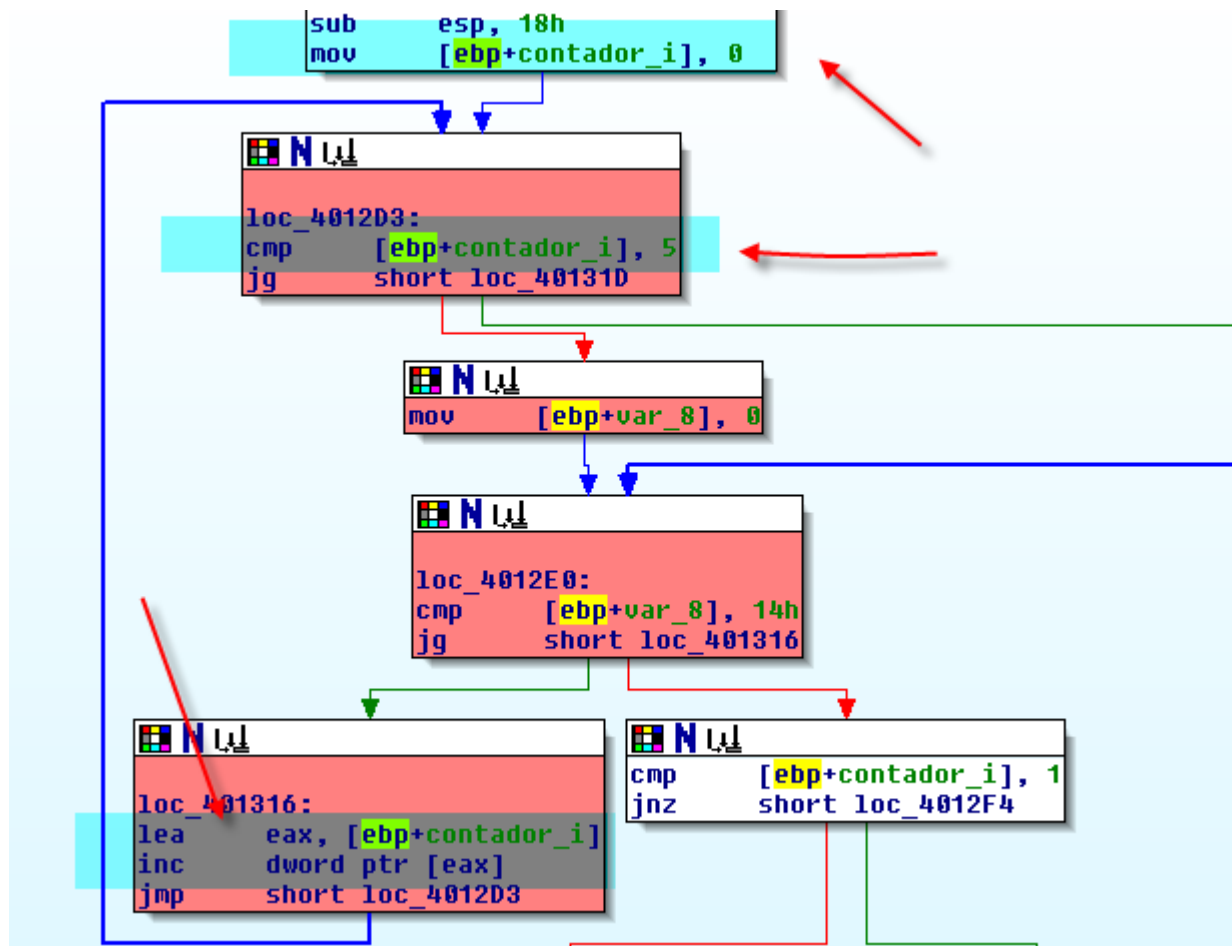
Vemos dos posibles variables **i** y **j** que se incrementan en forma diferente, ya que **i** solo vale 0 y 1, y **j** va de dos en dos y al llegar a 20 vuelve a empezar de cero, jeje, esto parecen dos **for** anidados uno dentro de otro, veamos en el IDA.



Allí vemos que el IDA nos muestra la funcion con dos variables como pensábamos, seguramente para los valores de **i** y **j**, vemos marcados en rosado los bloques que pertenecen al loop principal veamos como se inicializa, como se incrementa y cual es su condición de salida.



Vemos este for tiene un contador que se inicializa en 0 y que sale del mismo cuando es mayor que 5, y que se incrementa de uno en uno, por lo cual esta variable solo puede ser el contador i, pues j se incrementa de 2 en 2, así que renombramos la variable.



Así que ya podemos escribir el primer for de nuestro código.

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

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

```
funcion(){
```

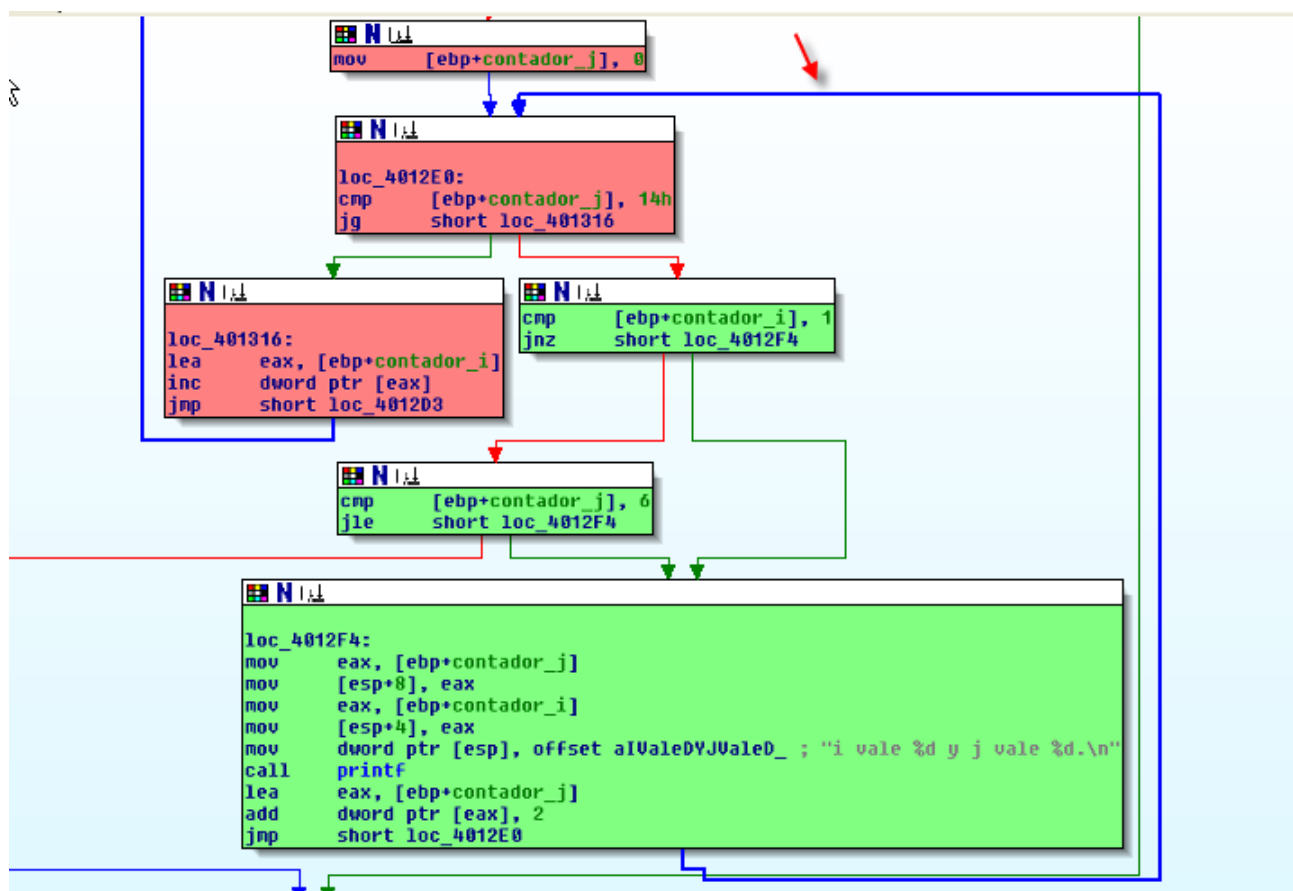
```
int i, j;
```

```
    for (i=0; i<=5; i++)
```

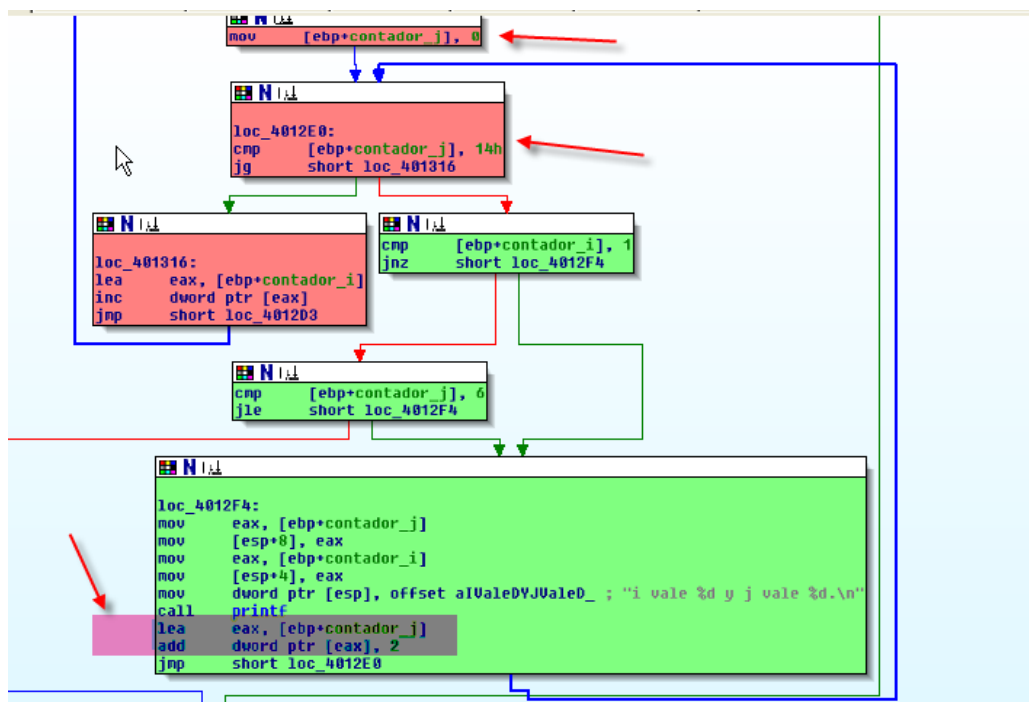
```
}
```

Declaramos como int las dos variables que aquí llamamos **i** y **j** en el IDA para que se vea mejor las llamo **contador_i** y **contador_j** si no se ve feo.

Veamos el código que se ejecuta dentro de este primer ciclo:



Vemos que inicializa a cero la otra variable **contador_j** y ello es otro loop ya que vemos la flecha azul que indica el mismo, este loop tiene como valor inicial cero, como salida si es mas grande que 14 hexa o sea 20, y se incrementa de a dos, veamos eso en el IDA.



Vemos el incremento al final que mediante el **lea** y el **add [eax],2** se incrementa como vimos de dos en dos, así que tenemos un for dentro de otro, escribamoslo.

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

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

```
funcion(){
```

```
    int i, j;
```

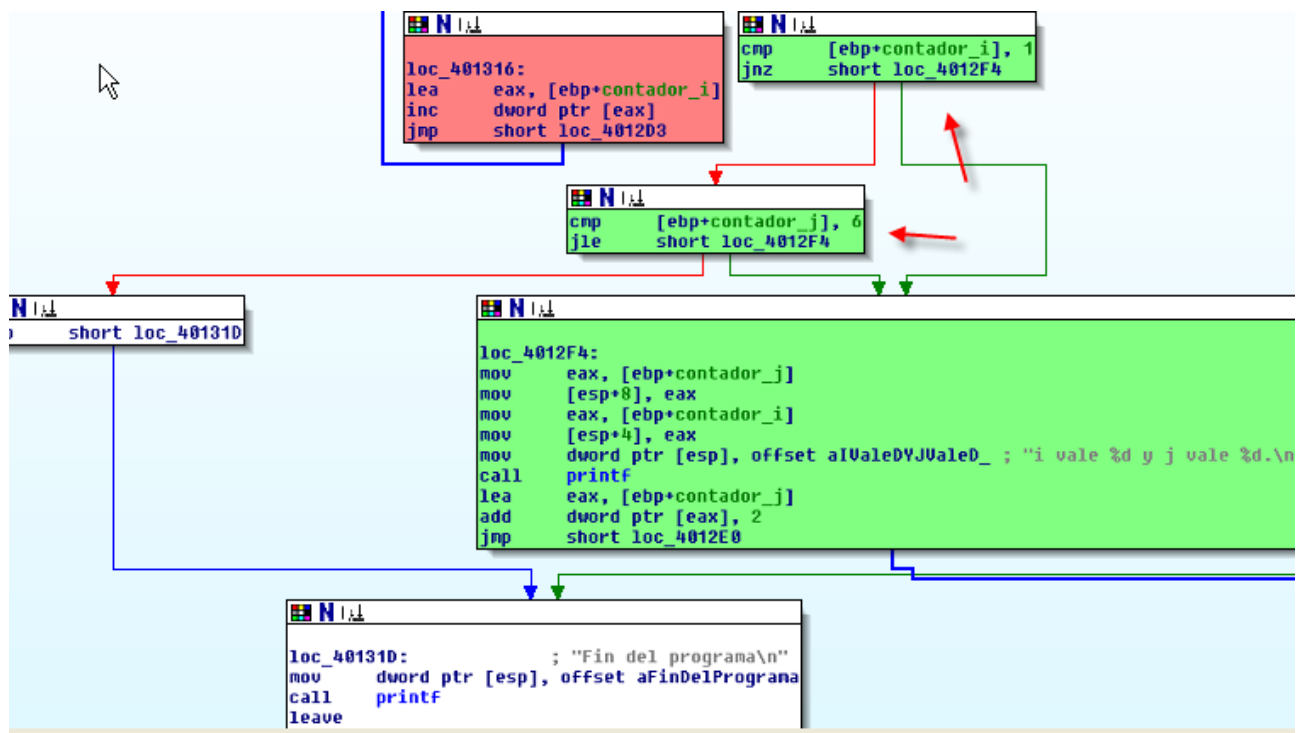
```
        for (i=0; i<=5; i++)
```

```
        for (j=0; j<=20; j+=2)
```

```
    }
```

Allí vemos el otro for anidado, que se incrementa de a dos (**j+=2**)

Ahora queda ver que código tiene dentro este for para ejecutar.



Vemos que compara primero si **i=1**, si no es uno va al otro bloque verde y sigue loopeando pero si es uno, compara la otra variable **j** si es **menor o igual que 6**, si es así continua loopeando y si no sale absolutamente fuera de ambos loops, esto no puede realizarse con un solo break pues solo te sacaría del loop mas interno, pero no de los dos a la vez, la forma es usando **goto**, que es una forma

horrible y no se aconseja pero bueno hay que verla.

```
if ((i==1) && (j>=7)) goto salida;
```

La instrucción **goto** se usa con una etiqueta en este caso **salida**, cuando ambas condiciones se den, forzamos la salida de ambos loops.

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

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

```
funcion(){
```

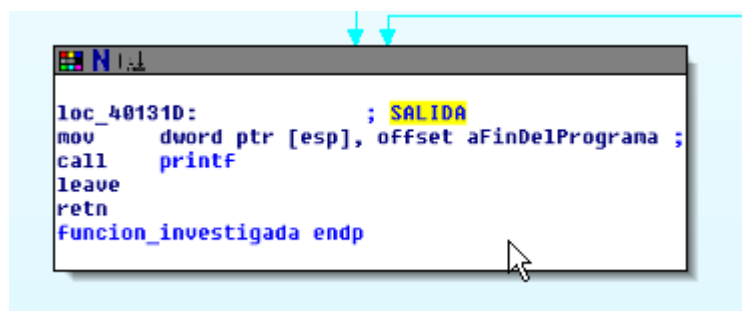
```
    int i, j;
```

```
        for (i=0; i<=5; i++)  
        for (j=0; j<=20; j+=2)  
        {  
            if ((i==1) && (j>=7)) goto salida;  
        }
```

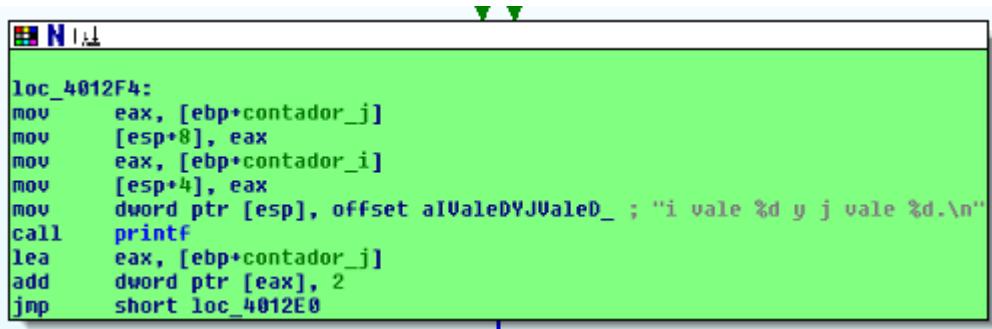
```
    salida:
```

```
        printf("Fin del programa\n");  
}
```

Vemos que cuando sale, mediante el **goto** va al fin del programa en este bloque, al que aclaro en IDA que es la etiqueta SALIDA, apretando punto y coma y escribiendo.



Por supuesto nos quedaba lo que hace el programa mientras esta dentro de ambos loop esto es este bloquecito donde imprime el mensaje que vimos del valor de **i** y **j**, lo agregamos al código.



```
loc_4012F4:
mov     eax, [ebp+contador_j]
mov     [esp+8], eax
mov     eax, [ebp+contador_i]
mov     [esp+4], eax
mov     dword ptr [esp], offset aI0Vale0YJ0Vale0_ ; "i vale %d y j vale %d.\n"
call    printf
lea     eax, [ebp+contador_j]
add     dword ptr [eax], 2
jmp     short loc_4012E8
```

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

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

```
funcion(){
```

```
int i, j;
```

```
    for (i=0; i<=5; i++)
    for (j=0; j<=20; j+=2)
    {
        if ((i==1) && (j>=7)) goto salida;
        printf("i vale %d y j vale %d.\n", i, j);
    }
```

```
    salida:
```

```
        printf("Fin del programa\n");
    }
```

Así que ya tenemos todo completo vemos el código fuente, los dos loops que si no hubiera **goto**, **i** valdría desde 0 hasta 6 incrementándose de a uno y que **j valdría** de 0 a 10 incrementándose de a dos, así que cuando **i** valga 0, la condición del **goto** no se cumplirá y **j** tomara todos los valores desde 0 a 20 como vimos en la salida en la imagen de abajo en rosado, ahora cuando **i** vale **1**, y **j** pasa a ser mayor o igual que 7 sale y imprime Fin del programa.

```
C:\ C:\Documents and Settings\ricnar\Esc
i vale 0 y j vale 0.
i vale 0 y j vale 2.
i vale 0 y j vale 4.
i vale 0 y j vale 6.
i vale 0 y j vale 8.
i vale 0 y j vale 10.
i vale 0 y j vale 12.
i vale 0 y j vale 14.
i vale 0 y j vale 16.
i vale 0 y j vale 18.
i vale 0 y j vale 20.
i vale 1 y j vale 0.
i vale 1 y j vale 2.
i vale 1 y j vale 4.
i vale 1 y j vale 6.
Fin del programa
```

Bueno hemos terminado la resolución de los tres ejemplos veremos si seguimos con una parte 4, pues me bajonee un poco.

Byes

Ricardo Narvaja