

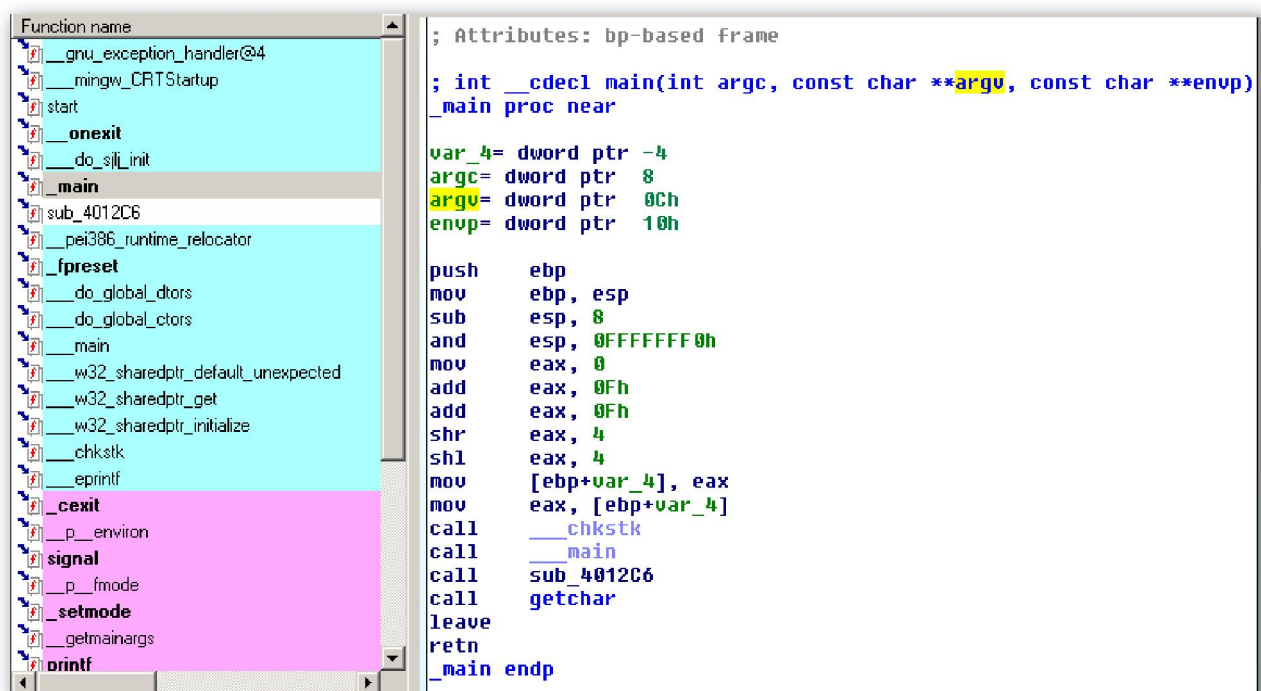
Aproximación al ejercicio 6 (Cast)

Lo primero que haremos será ejecutar el programa para tener una leve idea de lo que podremos encontrar cuando lo analicemos con IDA. Hacemos clic sobre él, y nos aparece en la ventana de comandos la siguiente vista.

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

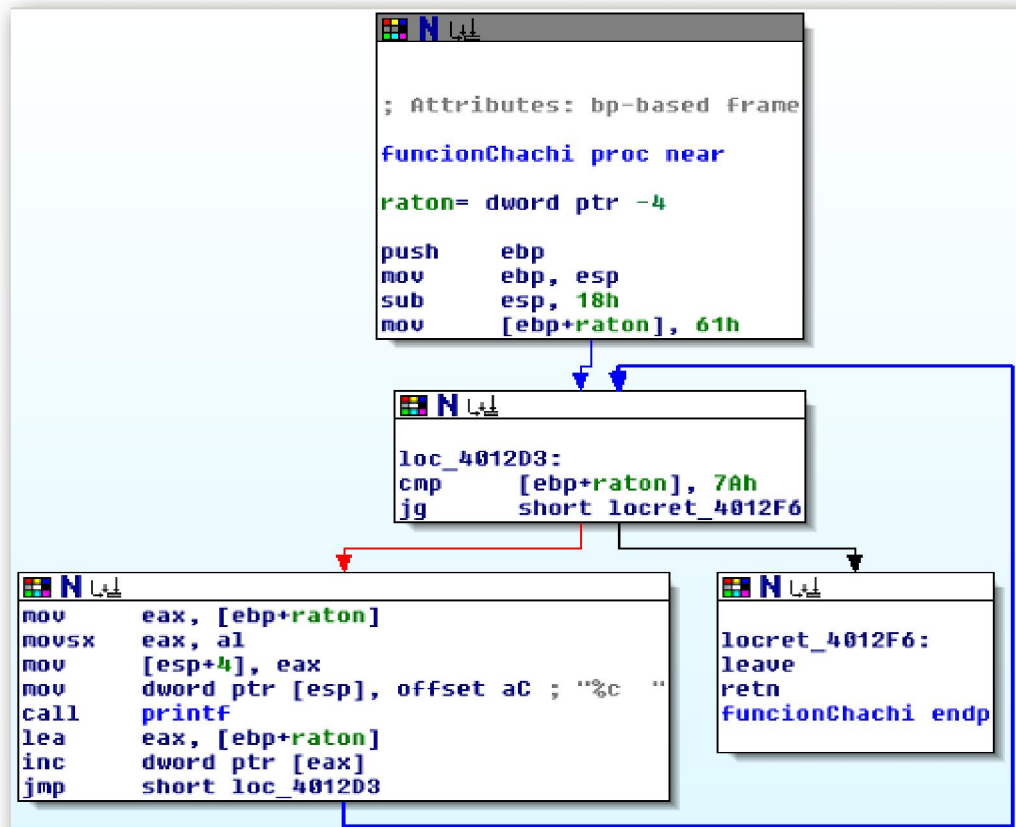
En principio podríamos confirmar de que se trata un programa para imprimir una serie de caracteres, en concreto 26.

Carguemos nuestro programa para analizarlo a fondo en IDA.



The screenshot shows the IDA Pro interface. On the left, the 'Function name' list is visible, with functions like `__gnu_exception_handler@4`, `__mingw_CRTStartup`, `start`, `__onexit`, `__do_sjl_init`, `__main`, `sub_4012C6`, `__pei386_runtime_relocator`, `__fpreset`, `__do_global_dtors`, `__do_global_ctors`, `__main`, `__w32_sharedptr_default_unexpected`, `__w32_sharedptr_get`, `__w32_sharedptr_initialize`, `__chkstk`, `__eprintf`, `__cexit`, `__p__environ`, `signal`, `__p__fmode`, `__setmode`, `__getmainargs`, and `printf`. The `__main` function is highlighted. On the right, the disassembly of the `__main` function is shown, starting with `__cdecl main(int argc, const char **argv, const char **envp)` and `__main proc near`. The disassembly includes variable declarations for `var_4`, `argc`, `argv`, and `envp`, followed by a series of instructions including `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`, `call __main`, `call sub_4012C6`, `call getchar`, `leave`, `retn`, and `__main endp`.

Se nos muestra la función “**main**”, si nos fijamos existen cuatro “**call**” de los cuales el nombrado con **sub_4012C6** será la función propiamente del código del programa la cual nos interesará analizar. Bueno observemos la función en cuestión, como yo ya la he mirado he visto que es la perteneciente al programa en cuestión. Con lo cual la he renombrado como “**funcionChachi**”. Veamosla físicamente.



A simple vista podemos observar una desviación de flujo realizada por un salto “**jg**”, por cierto si os fijáis las flechas de flujo son de color rojo y negro esto es debido a mi problema de daltonismo. Sigamos, vemos también una línea de flujo de un bucle “**for**” y asimismo la creación de una variable, **var_4**. Lo primero que haremos será renombrar dicha variable, como la iremos siguiendo como si de ello se tratara, le pondremos el nombre de “**raton**”.

```

; Attributes: bp-based frame

funcionChachi proc near

raton= dword ptr -4

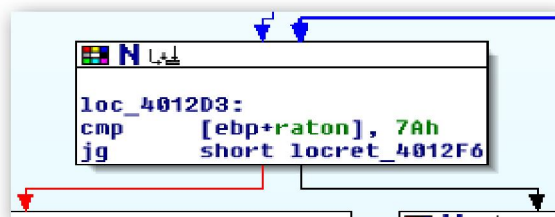
    push    ebp
    mov     ebp, esp
    sub     esp, 18h
    mov     [ebp+raton], 61h

```

Estudiemos nuestro primer módulo. En él podemos observar como se inicializa nuestra variable “**raton**” con un valor hexa de **61**, como somos muy listos recordamos que el valor 61h corresponde a la letra “**a**” minúscula, por lo tanto la cambiamos al valor de comprensión natural para nosotros.

```
mov     [ebp+raton], 'a'
```

Una vez inicializada con dicho valor, el flujo del programa sigue hacia el siguiente módulo de instrucciones, veamos.



Observamos que dicho módulo nos proporcionará una desviación del flujo de ejecución gracias al salto “**jg**” y que además recibe dicho flujo otra vez con la flecha del bucle “**for**”. Si lo analizamos vemos que “**raton**” es comparado con el valor “**7Ah**”, el cual también sabemos que corresponde al carácter “**z**”. Por consiguiente también lo renombramos

```
cmp     [ebp+raton], 'z'
```

En consecuencia podemos afirmar que se realiza una comparación del valor que en aquel momento tenga la variable “**ratón**” con el valor “**z**”. Después de dicha comparación nos encontramos con la desviación de flujo con el salto “**jb**” dicho salto comprueba si el valor es mayor, si éste lo es desviará el flujo hacia **locret_4012F6** con lo cual el programa finalizará. Si el valor no es mayor el flujo de ejecución seguirá hacia el siguiente módulo, el cual vamos a analizar a continuación.

```
mov     eax, [ebp+raton]
movsx   eax, al
mov     [esp+4], eax
mov     dword ptr [esp], offset aC ; "%c "
call    printf
lea     eax, [ebp+raton]
inc     dword ptr [eax]
jmp     short loc_4012D3
```

Vemos como toma a “**ratón**”, que en este momento alberga un valor numérico, y lo pasa a “**eax**”. A continuación toma la parte baja del registro “**eax**” y con **movsx** lo convierte en “**int**” pasándolo a “**eax**”. A continuación con “**%c**” se toma el valor entero de **eax** como **letra** y se le pasa a la función **printf** para que

ésta lo imprima en pantalla como carácter. Después de imprimir la letra, con la instrucción “**lea**” se toma el valor de “**ratón**” este se incrementa con **1** gracias a la instrucción “**inc**” y el flujo de ejecución se desvía hacia **loc_4012D3** gracias a la instrucción “**jmp**”.

Este bucle se irá ejecutando hasta que el valor de “**ratón**” exceda el valor de “**7Ah**”, y cuando esto se produzca el programa se parará pasando el flujo de ejecución a **locret_4012F6**. Habiendo cumplido el programa su misión que no es más que imprimir todos los caracteres desde la “**a**” hasta la “**z**”.

Si vemos el pseudocódigo de dicha función obtendremos lo siguiente:

```
int *__cdecl funcionChachi()
{
    int *result; // eax@3
    int raton; // [sp+14h] [bp-4h]@1

    for ( raton = 'a'; raton <= 'z'; ++raton )
    {
        printf("%c ", (char)raton);
        result = &raton;
    }
    return result;
}
```

Saludos.
Bigundill@