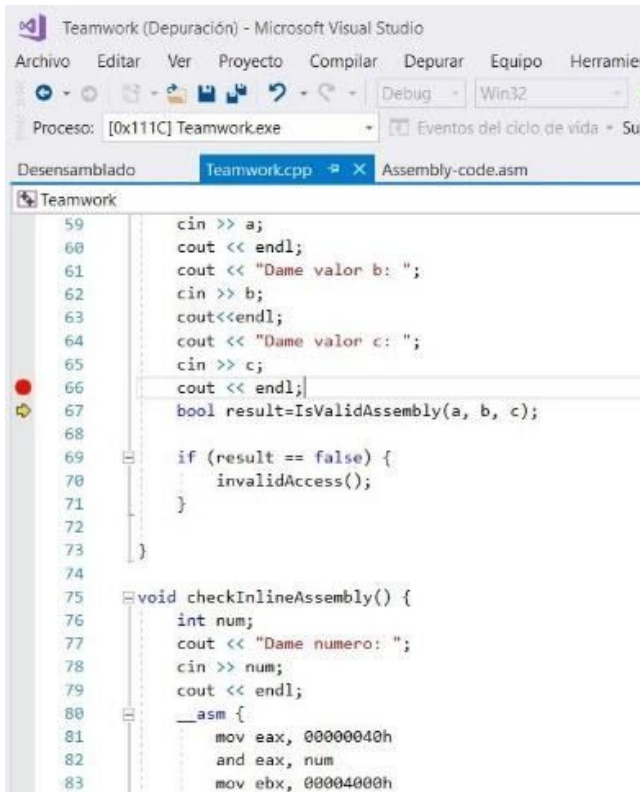


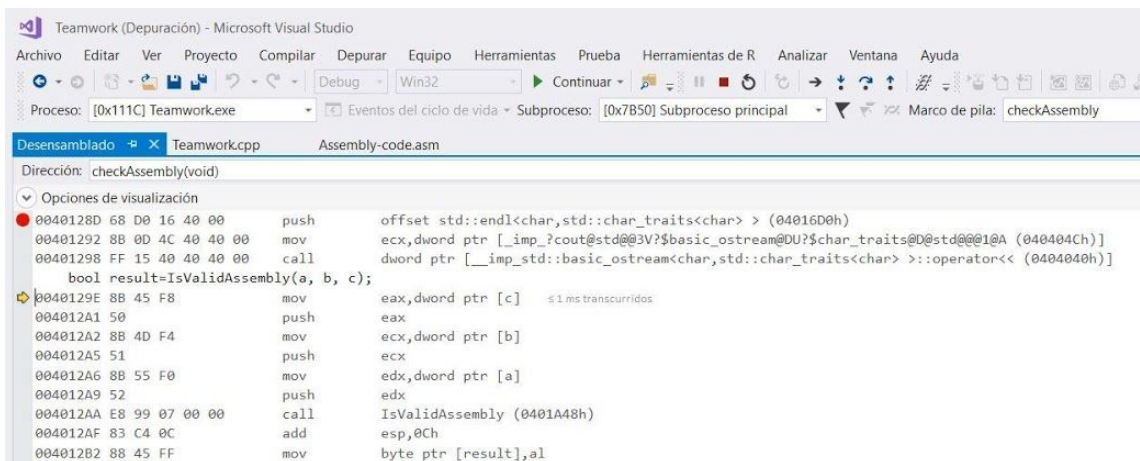
Ejercicio 1:

Primero hay que poner un punto de interrupción antes de la llamada al método (preferiblemente cerca de la llamada a `IsValidAssembly()`) :



```
59     cin >> a;
60     cout << endl;
61     cout << "Dame valor b: ";
62     cin >> b;
63     cout<<endl;
64     cout << "Dame valor c: ";
65     cin >> c;
66     cout << endl;
67     bool result=IsValidAssembly(a, b, c);
68
69     if (result == false) {
70         invalidAccess();
71     }
72
73 }
74
75 void checkInlineAssembly() {
76     int num;
77     cout << "Dame numero: ";
78     cin >> num;
79     cout << endl;
80     __asm {
81         mov eax, 00000040h
82         and eax, num
83         mov ebx, 00004000h
```

Después, mediante el botón derecho del ratón, seleccionamos la opción “ir a desensamblado”. Una vez estés en la ventana de desensamblado solo hay que buscar la llamada al método.



```
0040128D 68 D0 16 40 00 push offset std::endl<char, std::char_traits<char> > (04016D0h)
00401292 8B 00 4C 40 40 00 mov ecx, dword ptr [__imp_?cout@std@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A (040404Ch)]
00401298 FF 15 40 40 40 00 call dword ptr [__imp_std::basic_ostream<char, std::char_traits<char> >::operator<< (0404040h)]
bool result=IsValidAssembly(a, b, c);
0040129E 8B 45 F8 mov eax, dword ptr [c] <1 ms transcurridos>
004012A1 50 push eax
004012A2 8B 4D F4 mov ecx, dword ptr [b]
004012A5 51 push ecx
004012A6 8B 55 F0 mov edx, dword ptr [a]
004012A9 52 push edx
004012AA E8 99 07 00 00 call IsValidAssembly (0401A48h)
004012AF 83 C4 0C add esp, 0Ch
004012B2 8B 45 FF mov byte ptr [result], al
```

De ese bloque de código, la dirección de memoria en la que se empieza a pasar los parámetros al método es **0040128E h**; y las instrucciones y sus mnemónicos son las siguientes:

```
8B 45 F8      mov eax,dword ptr [c]
50           push eax
8B 4D F4      mov ecx,dword ptr [b]
51           push ecx
8B 55 F0      mov edx,dword ptr [a]
52           push edx
```

Ejercicio 2:

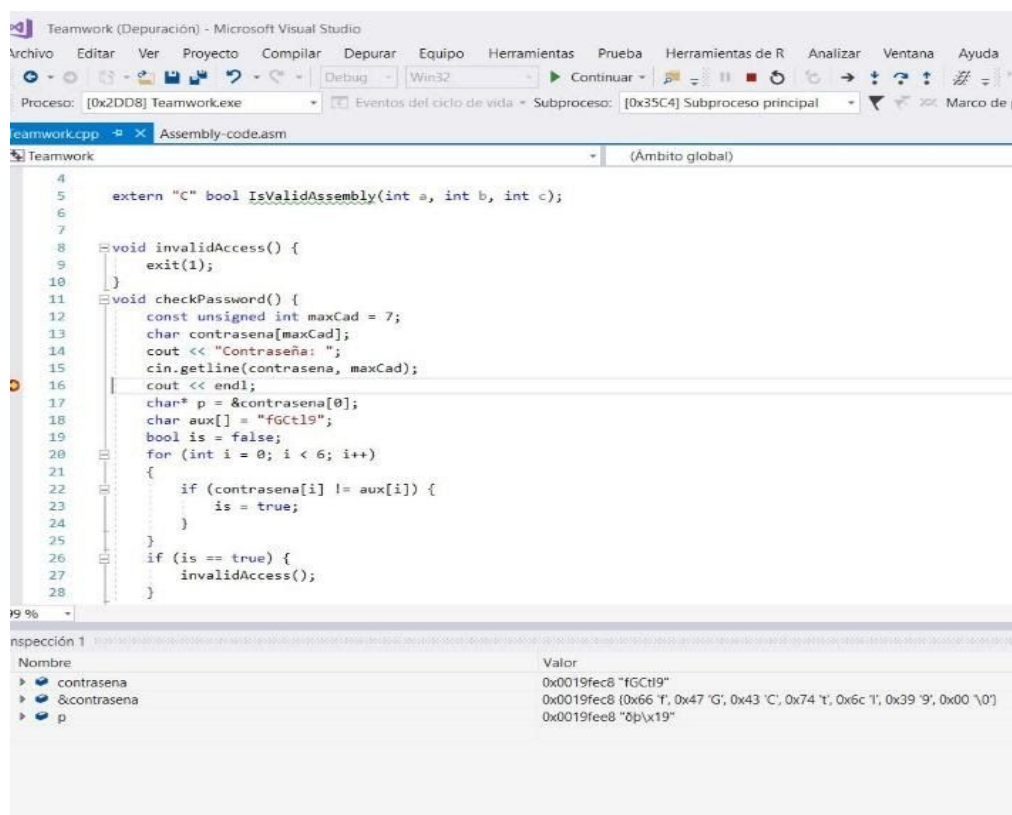
El primer paso, como en el ejercicio anterior, es poner un punto de ruptura después de haber leído la cadena por consola.

Una vez hecho esto, en la barra de herramientas, seleccionas

Depurar >> ventanas >> inspección > > inspección 1

Te saldrá una ventana emergente llamada “inspección 1”. En ella escribes “&” y el nombre de la variable (la & indica dirección de memoria de la variable). Una vez lo escribas, le das a “enter”, y te saldrá el valor de la dirección de memoria en la que empieza la cadena.

Si lo quieres en hexadecimal, con el botón derecho seleccionas “Presentación hexadecimal”.



Ejercicio 3:

Como en ejercicios anteriores hay que usar el desensamblado mediante un punto de interrupción. Una vez estés en el desensamblado, hay que buscar el epílogo, que consiste en todos los pop y la instrucción ret. De esta forma basta con identificar

Un bloque de código compuesto por estas dos instrucciones:

```
Dirección: checkPassword(void)
▼ Opciones de visualización
004010AF 8B 55 F8      mov     edx,dword ptr [ebp-8]
004010B2 0F BE 44 15 EC movsx   eax,byte ptr aux[edx]
004010B7 3B C8        cmp     ecx,eax
004010B9 74 04        je      checkPassword+9Fh (04010BFh)
        is = true;
004010BB C6 45 FF 01   mov     byte ptr [is],1
        }
    }
004010BF EB D7        jmp     checkPassword+78h (0401098h)
    if (is == true) {
004010C1 0F B6 4D FF   movzx   ecx,byte ptr [is]
004010C5 83 F9 01     cmp     ecx,1
004010C8 75 05        jne     checkPassword+0AFh (04010CFh)
        invalidAccess();
004010CA E8 31 FF FF FF call    invalidAccess (0401000h)
    }
}
004010CF 8B E5        mov     esp,ebp
    }
}
004010D1 5D          pop     ebp
004010D2 C3          ret
--- No hay archivo de origen -----
004010D3 CC          int     3
```

En este caso, el epílogo comienza en la dirección **004010D1**.

Ejercicio 4:

Primero hay que poner un punto de ruptura en el inicio del código del ensamblador en línea; e ir al ensamblador en línea

```
73
74 void checkInlineAssembly() {
75     int num;
76     cout << "Dame numero: ";
77     cin >> num;
78     cout << endl;
79     __asm {
80         mov eax, 00000040h
81         and eax, num
82         mov ebx, 00004000h
83         and ebx, num
84         shr eax, 6
85         shr ebx, 14
86         cmp eax, ebx
87         je consecuente
88         mov num, 0
89         jmp siguiente
90     consecuente :
91         mov num, 1
92     siguiente :
93     }
94     if (num == 0) {
95         invalidAccess();
96     }
97     else {
98         cout << "Valid access" << endl;
99     }
100 }
101
```

Después, buscar el bloque __asm en el desensamblador:

__asm {			
mov eax, 00000040h			
004012F9 B8 40 00 00 00	mov	eax, 40h	
and eax, num			
004012FE 23 45 FC	and	eax, dword ptr [num]	
mov ebx, 00004000h			
00401301 BB 00 40 00 00	mov	ebx, 4000h	
and ebx, num			
00401306 23 5D FC	and	ebx, dword ptr [num]	
shr eax, 6			
00401309 C1 E8 06	shr	eax, 6	
shr ebx, 14			
0040130C C1 EB 0E	shr	ebx, 0Eh	
cmp eax, ebx			
0040130F 3B C3	cmp	eax, ebx	
je consecuente			
00401311 74 09	je	checkInlineAssembly+5Ch (040131Ch)	
mov num, 0			
00401313 C7 45 FC 00 00 00 00	mov	dword ptr [num], 0	
jmp siguiente			
0040131A EB 07	jmp	siguiente (0401323h)	

En este caso la primera instrucción el código máquina de la primera instrucción es

B8 40 00 00 00 -> mov eax, 40h