

ESCUELA DE INGENIERÍA INFORMÁTICA DE OVIEDO

Fundamentos de Computadores Y Redes

Curso 2024-2025

Trabajo Grupal - Fase I

Díaz Mendaña, Diego - UO301887

García Pernas, Pablo - UO300167

Gota Ortín, Jorge - UO301023

Suárez Fernández, Fernando - UO300028

Grupo de prácticas: PL.3 - A

Titulación: PCEO Informática y Matemáticas

21 de marzo de 2025

Índice

1. Introducción	2
2. ID	2
3. Desarrollo del programa	2
3.1. Descripción general	2
3.2. Funciones implementadas	2
3.2.1. ControlWithReversedStrings()	2
3.2.2. MaskControl()	3
3.2.3. ControlInAsm()	3
3.2.4. CheckArray()	4
4. Cuestiones	4
4.1. Cuestión 1	4
4.2. Cuestión 2	5
4.3. Cuestión 3	6
4.4. Cuestión 4	6
5. Distribución del trabajo	7
5.1. Estrategia de trabajo	7
5.2. Distribución de tareas	7
5.3. Tiempos de desarrollo	8

1. Introducción

La presente memoria da cuenta del desarrollo de la Fase I del trabajo práctico correspondiente a la asignatura Fundamentos de Computadores y Redes, con el propósito de profundizar en los conceptos generales tratados en los laboratorios de la citada asignatura. A través de la implementación de un sistema de control de licencias parametrizado, se integran conocimientos relativos al lenguaje C/C++ y al ensamblador sobre arquitectura Intel x86-64, abordando aspectos fundamentales como el manejo de bits, máscaras, desplazamientos y estructuras de control.

2. ID

Para el desarrollo del presente proyecto se ha seleccionado como identificador de referencia el número *UO300028*, al ser el más bajo entre los disponibles dentro del grupo, conforme a lo estipulado en la guía oficial de la práctica. No obstante, cabe señalar que, en determinados fragmentos del código —como se detallará en secciones posteriores de esta memoria—, se ha optado por la utilización de identificadores alternativos. Esta decisión ha respondido a la necesidad de preservar la coherencia lógica y funcional del programa, la cual se veía comprometida al emplear un único identificador de forma estricta en todos los contextos del desarrollo.

3. Desarrollo del programa

3.1. Descripción general

El programa desarrollado consiste en un sistema simplificado de control de licencias, parametrizado mediante un identificador numérico asociado al número de matrícula universitaria. Su implementación combina el lenguaje C++, utilizado para la lógica principal y la gestión de entrada/salida, con código en ensamblador x86-64 para una función específica de validación. Todo el desarrollo se ha realizado en el entorno *Visual Studio*, respetando la configuración proporcionada por la asignatura (*Teamwork*).

3.2. Funciones implementadas

3.2.1. `ControlWithReversedStrings()`

La función implementada tiene por objeto la verificación del formato de dos cadenas introducidas por el usuario a través de la consola, conforme a los criterios establecidos en el enunciado de la práctica. Para la inversión de la segunda cadena, se ha recurrido a la función nativa `strrev()`, propia del lenguaje C++, lo que permite una manipulación directa y eficiente de las estructuras de texto. A continuación, se muestran sendos ejemplos de entrada válida e inválida:

- **Entrada válida:** `aaaaaaaaaaaaaaaa` y `1df4g7n9`
- **Entrada inválida:** `a` y `a`

3.2.2. MaskControl()

Dado que el identificador asignado inicialmente (300028) conducía a una interpretación no operativa de la condición planteada —al implicar, en la práctica, la selección de los 32 bits menos significativos del segundo y los 0 bits más altos del primeros, resultando en el uso exclusivo del segundo operando—, se ha optado por emplear de forma justificada el identificador alternativo 299928, con el fin de preservar la lógica y coherencia funcional del ejercicio.

La función en cuestión verifica el formato de dos valores enteros introducidos por consola, conforme a los criterios específicos establecidos en el guión de la práctica. A continuación, se muestran sendos ejemplos de entrada válida e inválida:

- **Entrada válida:** 1024 y 32315
- **Entrada inválida:** 0 y 0

Alternativamente, se ha valorado la opción de realizar este ejercicio empleando máscaras para la creación del nuevo número, en lugar de emplear desplazamientos de bits. Sin embargo, la implementación final ha sido con esta última opción. No obstante, el código correspondiente a la implementación con máscaras para la generación del nuevo número se muestra a continuación:

```
unsigned int higherMask = 0xFF800000; // 1111 1111 1000 0000 0000 0000 0000 0000
unsigned int lowerMask = 0x7FFFFFFF; // 0000 0000 0111 1111 1111 1111 1111 1111

unsigned int higherNumber = a & higherMask; //Obtenemos los bits altos del primer numero
unsigned int lowerNumber = b & lowerMask; //Obtenemos los bits bajos del segundo numero
```

3.2.3. ControlInAsm()

Dado que el identificador asignado inicialmente (300028) conducía a una interpretación no operativa de la condición planteada —al implicar, en la práctica, que los 0 bits más bajos tenían que ser mayores a 108, lo cual es imposible—, se ha optado por emplear de forma justificada el identificador alternativo 299928, con el fin de preservar la lógica y coherencia funcional del ejercicio.

La función realiza una verificación inicial en **ensamblador** sobre los 9 bits menos significativos de un valor de 32 bits introducido por consola. En caso de superar dicha validación, se procede a efectuar una segunda comprobación lógica entre los otros dos valores también recibidos por consola. A continuación, se muestran sendos ejemplos de entrada válida e inválida:

- **Entrada válida:** 111, 0 y 0
- **Entrada inválida:** 0, 0 y 0

3.2.4. CheckArray()

La función toma tres valores de entrada, que se almacenan en un array de tres elementos de 8 bits. Se realiza una operación AND a nivel de bit entre los tres valores, y si el resultado —interpretado en base decimal— difiere de 200, se muestra un mensaje de error y se interrumpe la ejecución. A continuación, se muestran sendos ejemplos de entrada válida e inválida:

- **Entrada válida:** 1 1 1
- **Entrada inválida:** z z z

4. Cuestiones

A continuación se presentan las respuestas a las cuestiones planteadas en el guión de la práctica.

4.1. Cuestión 1

En la siguiente imagen se muestra el código en ensamblador correspondiente al paso de parámetros y llamada de la función `IsValidAssembly()` desde `ControlInAsm()`. Para llegar a él, se ha empleado la opción de depuración de *Visual Studio* llamada *desensamblado* activando la opción *mostrar bytes de código*:

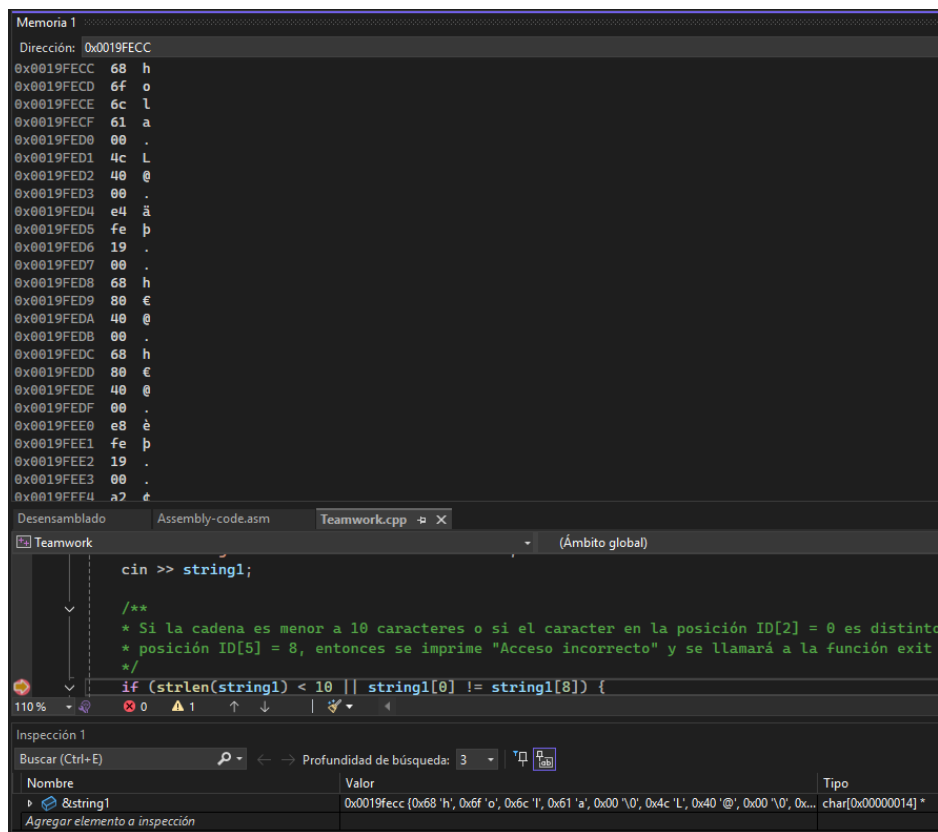
```
773E llama a la función IsValidAssembly
if (IsValidAssembly(a, b, c) == 0) {
00401337 8B 4D FC      mov     ecx,dword ptr [c] |
0040133A 51            push    ecx
0040133B 8B 55 F8      mov     edx,dword ptr [b]
0040133E 52            push    edx
0040133F 8B 45 F4      mov     eax,dword ptr [a]
00401342 50            push    eax
00401343 E8 20 15 00 00 call    IsValidAssembly (0402868h)
00401348 83 C4 0C      add     esp,0Ch
0040134B 0F B6 C8      movzx   ecx,al
0040134E 85 C9        test    ecx,ecx
00401350 75 2A        jne     ControlInAsm+0ECh (040137Ch)
```

A partir de la imagen, podemos ver que el paso de parámetros se realiza en la dirección de memoria 00401337 y la llamada al procedimiento en 00401343. Dado que el enunciado destila cierta ambigüedad en cuanto al código requerido, se ha optado por añadir también el código de la función `IsValidAssembly()`:

; -----Prologo-----		shl ecx, 9	; Id[2] = 9
push ebp	; Guarda el program counter	0040286E C1 E1 09	shr ecx, 9
0040286E 55	push ebp	and eax, ecx	; Dejo solo el bit 9 de eax
mov ebp, esp	; Mueve al program counter el counter de subprograma	00402891 23 C1	and eax, ecx
00402869 8B EC	mov ebp, esp	shr eax, 9	; Lo nuevo a la posicion 0
		00402893 C1 E8 09	shr eax, 9
push ebx	; Guarda el valor de ebx		
0040286B 53	push ebx	shr ecx, 7	; Id[2] - Id[0] = 9 - 2 = 7
push ecx	; Guarda el valor de ecx	00402896 C1 E9 07	shr ecx, 7
0040286C 51	push ecx	and ebx, ecx	; Dejo solo el bit 7
		00402899 23 D9	and ebx, ecx
		shr ebx, 7	; Lo nuevo a la posicion 0
		0040289B C1 E8 07	shr ebx, 7
; -----Primera condicion-----			
; Los 9 bits mas bajos de a tienen que ser mayores que 108		cmp eax, ebx	; eax - ebx != 0 implica falso
		0040289E 3B C3	cmp eax, ebx
mov eax, [ebp+8]	; eax = a		
0040286D 8B 45 08	mov eax, dword ptr [ebp+8]	JZ verdadero	; Si son iguales cumple las dos condiciones
		004028A0 74 02	je verdadero (004028A4h)
mov ebx, 0	; ebx = 0	JMP falso	; Si no, no las cumple
00402870 8B 00 00 00	mov ebx, 0	004028A2 EB 07	jmp verdadero+7h (004028A8h)
not ebx; Todo 1s	; ebx = 0xFFFFFFFF		
00402875 F7 D3	not ebx		
shr ebx, 23	; Deja tantos 1s como diga Id[1] = 9 (32 - 9 = 23),	verdadero:	
00402877 C1 EB 17	shr ebx, 17h	mov eax, 1	; Devuelve 1
		004028A4 8B 01 00 00	mov eax, 1
and eax, ebx	; Máscara de los 9 bits mas bajos de eax	jmp Epilogo	
0040287A 23 C3	and eax, ebx	004028A9 EB 07	jmp verdadero+8Eh (004028B2h)
cmp eax, 108	; eax-108 <= 0 implica falso		
0040287C 83 F8 6C	cmp eax, 6Ch	falso:	
		mov eax, 0	; Devuelve 0
JZ falso	; Comprueba = 0	004028AB 8B 00 00 00	mov eax, 0
0040287F 74 2A	je verdadero+7h (004028A8h)	jmp Epilogo	
JC falso	; Comprueba < 0	004028B0 EB 00	jmp verdadero+8Eh (004028B2h)
00402881 72 28	jb verdadero+7h (004028A8h)		
; -----Segunda condicion-----		Epilogo:	
; El bit 9 de ebx tiene que ser el mismo que el bit 2 de ecx		pop ecx	; Recupera ecx
		004028B2 59	pop ecx
mov eax, [ebp+12]	; eax = b	pop ebx	; Recupera ebx
00402883 8B 45 0C	mov eax, dword ptr [ebp+0Ch]	004028B3 5B	pop ebx
mov ebx, [ebp+16]	; ebx = c	mov esp, ebp	; Recupera el program counter
00402886 8B 5D 10	mov ebx, dword ptr [ebp+10h]	004028B4 8B E5	mov esp, ebp
		pop ebp	; Descarta el sub counter
mov ecx, 1	; ecx = 1	004028B5 5D	pop ebp
00402889 89 01 00 00	mov ecx, 1	ret	; Devuelve
		004028B7 C3	ret

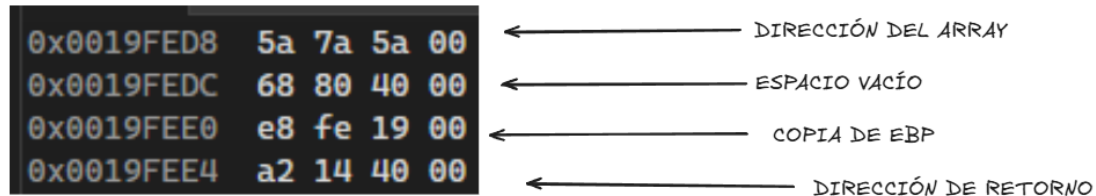
4.2. Cuestión 2

Empleando las herramientas de depuración de *Visual Studio* (depurador, ventana de memoria, etc.), se ha procedido con la ejecución del programa y, tras pasar la primera cadena de caracteres y llegar al **breakpoint** correspondiente, se ha analizado la dirección de memoria en la que se almacena el array de caracteres. Dicha posición de memoria es 0x0019FECC, como se muestra en la siguiente captura de pantalla:



4.3. Cuestión 3

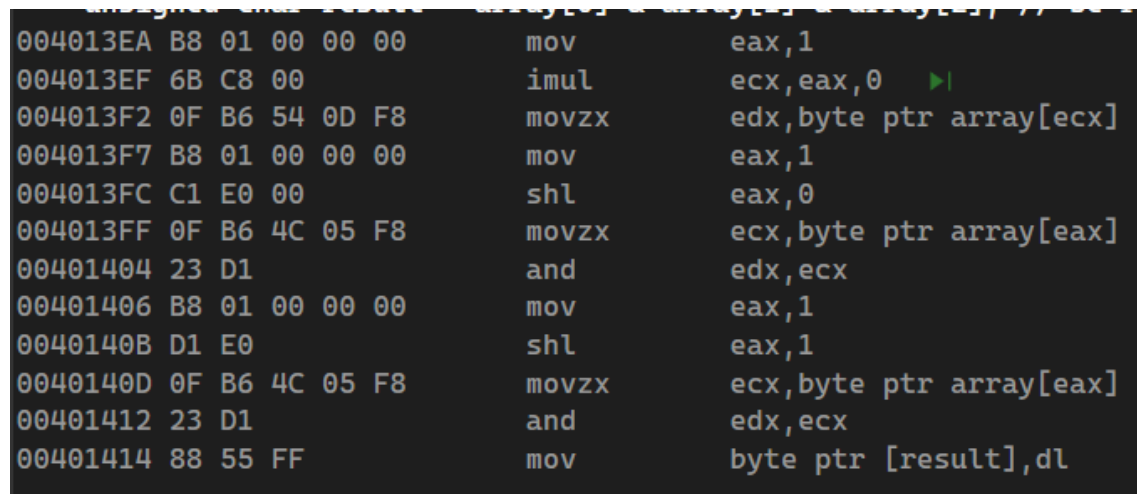
Se han empleado las herramientas de depuración de *Visual Studio* (depurador, ventana de memoria, ventana de registros, ventana de desensamblado, etc.) para analizar el contenido de los registros y la pila en el momento de la llamada a la función `CheckArray()`. Se ha comprobado el valor `EBP` en la ventana de registros y en la de memoria y la dirección de retorno con la ventana de desensamblado y la de memoria. A continuación, se muestra una captura de pantalla con los resultados obtenidos:



Notar que las direcciones de retorno se dan en formato *little-endian*, por lo que la dirección de retorno es 0x004014A2.

4.4. Cuestión 4

Para la realización de esta cuestión, se ha empleado la herramienta de desensamblado de *Visual Studio* para analizar el código ensamblador de la función `CheckArray()`. A continuación, se muestra una captura de pantalla con el código ensamblador de la función en cuestión:



A partir de la imagen, se puede ver el código correspondiente a la línea en la que se realiza la operación `AND` a nivel de bit entre los tres valores introducidos por el usuario:

- `MOV eax, 1`: Mueve el valor 1 al registro `EAX`.
- `IMUL ecx, eax, 0`: Multiplica el valor de `EAX` por 0 y lo almacena en `ECX`.
- `MOVZX edx, byte prt array[ecx]`: Mueve el valor de `array[0]` al registro `EDX`. Ya que se trata de un `byte`, se rellena con ceros a la izquierda.

5. Distribución del trabajo

5.1. Estrategia de trabajo

La implementación del proyecto se ha organizado siguiendo una estrategia de trabajo colaborativo, basada en la asignación individual de cada una de las funciones principales descritas en el enunciado. Cada integrante del grupo asumió la responsabilidad del desarrollo, depuración y validación de una función específica, lo que favoreció una distribución equilibrada de las tareas y una mayor especialización técnica en cada componente del sistema.

Como soporte para el trabajo en equipo, se ha utilizado el sistema de control de versiones distribuido *Git* en combinación con la plataforma *Github* como repositorio remoto centralizado. El grupo ha adoptado un flujo de trabajo basado en la estrategia de *feature branching*, en la cual cada funcionalidad fue desarrollada en una rama independiente derivada de la rama principal `main`. Una vez finalizada la implementación de cada funcionalidad, se procedía a la apertura de una `pull request` (PR), a través de la cual se solicitaba la revisión del código por parte de los demás integrantes.

Este enfoque ha permitido establecer un proceso sistemático de *code review*, mediante el cual cada `commit` propuesto era examinado antes de su integración definitiva en la rama principal del repositorio. Asimismo, se han empleado herramientas integradas de GitHub como la resolución de *merge conflicts*, en caso de ser necesarios. Las revisiones cruzadas entre miembros han servido tanto para detectar errores lógicos como para asegurar la consistencia del estilo y la correcta integración de las distintas partes del código.

Gracias a este flujo de trabajo, se ha logrado mantener una trazabilidad completa de los cambios, facilitar la colaboración asíncrona y reforzar la calidad del desarrollo mediante la incorporación de prácticas propias del desarrollo profesional de software.

El repositorio que ha servido como entorno de trabajo colaborativo se encuentra alojado en la plataforma *GitHub*, bajo el siguiente enlace: <https://github.com/PabloGarPe/fcrtrabajo>. Cabe señalar que, en el momento de redacción de esta memoria, el repositorio permanece configurado como privado, por lo que su acceso podría no estar disponible de forma pública. No obstante, se encuentra íntegramente documentado y estructurado conforme a las convenciones adoptadas durante el desarrollo, con un historial completo de *commits*, ramas y solicitudes de fusión que evidencian el flujo de trabajo seguido por el grupo.

5.2. Distribución de tareas

Tal como se ha expuesto anteriormente, la distribución de tareas se ha estructurado en torno a la asignación individual de cada una de las funciones del programa a los distintos integrantes del grupo. Finalizada la implementación inicial, se procedió a una revisión colectiva de todas las contribuciones, llevada a cabo tanto mediante sesiones de puesta en común como a través del sistema de control de versiones, concretamente mediante la supervisión cruzada de las correspondientes *pull requests* en el repositorio compartido.

A continuación, se detalla la asignación específica de tareas realizada por cada miembro del equipo:

Integrante	Función implementada
Diego Díaz Mendaña	<code>ControlWithReversedStrings()</code>
Pablo García Pernas	<code>MaskControl()</code>
Fernando Suárez Fernández	<code>ControlInAsm()</code>
Jorge Gota Ortín	<code>CheckArray()</code>

5.3. Tiempos de desarrollo

A continuación, se especifica el tiempo estimado invertido por cada miembro, considerando tanto las tareas individuales como las actividades colaborativas de revisión y coordinación:

Integrante	Tiempo dedicado
Diego Díaz Mendaña	5 horas
Pablo García Pernas	5 horas y 30 minutos
Fernando Suárez Fernández	4 horas
Jorge Gota Ortín	3 horas y 30 minutos