Trabajo práctico N°0

Etcheverri Franco, *Padrón Nro. 95.812*
franverri@hotmail.com
Vicari Dario, *Padrón Nro. 86.559*
dariovicari@yahoo.com.ar
Gonzalez Esteban, *Padrón Nro. 54.476*
egonza@fi.uba.ar

1er. Cuatrimestre de 2016
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

1. Introducción

Este primer trabajo práctico consiste en desarrollar un programa en el lenguaje C, el cual va a recibir como entrada un archivo y luego se va a proceder a codificarlo a base 64 que, como bien explica el enunciado, se creó para poder transmitir archivos binarios en medios que sólo admitían texto (64 es la mayor potencia de 2 que se podía representar sólo con caracteres ASCII imprimibles). Adicionalmente, el programa también debe ofrecer la posibilidad de realizar el trabajo inverso, es decir, recibir un texto codificado y llevarlo a su estado inicial decodificándolo.

2. Desarrollo

Para llevar adelante el trabajo práctico se dividió la tarea principalmente en tres secciónes:

- **Ayudas brindadas**
- **Codificación**
- **Decodificación**

En los apartados que preceden a esta sección se explicaran con mayor detalle la implementación de cada una de ellas.

2.1. Ayudas brindadas

Se implementó un menu de ayuda para facilitarle la tarea al usuario que va a utilizar el programa. Al ejecutar el siguiente comando "./tp0 -h", se despliega un menu con los servicios y posibilidades de ejecución del programa:

- -V, –version (Print version and quit)
- -h, –help (Print this information)
- -o, –output (Path to output file)
- -i, –input (Path to input file)
- -d, –decode (Decode a base64-encoded file)

En términos de código, la forma en que se logro desarrollar esta sección fue a partir de la entrada de los paramétros del programa que en el lenguaje utilizado son de fácil acceso mediante las variables **argc** y **argv** provistas por el entorno.

2.2. Codificación

Para pasar de cualquier archivo a uno en base 64, es decir, codificar la información, utilizamos una funcion llamada b64 la cual recibe los siguientes parámetros:

"static int b64(char opt, char *infilename, char *outfilename, int linesize)"

- **opt:** indica si debe codificarse ('e') o decodificarse ('d') el archivo.
- **infilename:** contiene el nombre del archivo origen que va a ser procesado.
- **outfilename:** contiene el nombre del archivo destino donde se van a guardar los datos luego de realizar el codificado o decodificado.

Lo que realiza esta función básicamente es ir leyendo caracteres del archivo origen y guardandolos en un buffer múltiplo de 3 (el que usaremos va a ser exactamente de 3 chars es decir 24 bites) y cuando el buffer se encuentra lleno codificarlo a b64 dividiendolo en 4 elementos, cada uno de ellos de 6 bites.

Esto hace que cualquier combinacion de unos y ceros vaya a parar a un elemento dentro de la tabla de caracteres permitidos por la codificación en base 64. Por último, estos nuevos caracteres codificados van a ser escritos en el nuevo archivo destino.

Cabe destacar que, en el caso de que se llegue al fin de archivo antes de llenar el buffer nuevamente, se procedera a completarlo con caracteres '=' para lograr que sea múltiplo de 3 y poder codificarlo.

Un aspecto interesante de la implementación es el codificado del buffer de 24 bits. Para ellos se van a utilizar variables del tipo char y operadores logicos para acomodar la posición de los bits significativos. Van a ocurrir tres posibilidades diferentes en este buffer:

- **Primer elemento:** al obtener el primer elemento del buffer de caracteres obtendríamos 8 bits, de los cuales sólo nos interesan los 6 primeros. Por lo tanto al aplicar un desplazamiento lógico de dos lugares hacia la derecha de esa variable donde alojamos el caracter nos preveería el efecto deseado, ya que los dos bits mas significativos quedarían en cero, que es equivalente a darle importancia únicamente a los 6 bits restantes.
- **Segundo elemento:** para poder conformar el segundo elemento del archivo codificado en base 64 necesitaríamos los 2 bits que no se utilizaron del primer char para el elemento anterior y 4 bits más correspondientes al segundo char del buffer. Para ello se va a realizar operaciones logicas de and y de desplazamiento. La forma en que opera el programa se puede observar en la Figura 1 para no tener que explicar en palabras como se procesa cada elemento lo cual hace más dificil la comprensión.

2.3. Deodificación

Para el decodificado, es decir, a partir de un archivo en base 64 transformalo a un archivo que acepte todos los caracteres de la tabla ASCII, se utiliza la misma función pero pasandole como parámetro el caracter 'd' en la opcion .ºpt".

El procedimiento es similar al de la codificación pero ahora se utiliza un buffer con 4 elementos para decodificar, ya que para poder obtener 3 caracteres formados por 8 bits necesito 4 de 6 bits. Por lo que las operaciones para cada paso son las mostradas en la Figura 2

	Char								Operación	Char							
1	a7	a6	a5	a4	a3	a2	a1	a0	Desplazo 2 a derecha	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	X	X	X	X	X	X	-	-		0	0	a7	a6	a5	a4	a3	a2
2	a7	a6	a5	a4	a3	a2	a1	a0	and con 00000011 y	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	-	-	-	-	-	-	X	X	desplazo 4 lugares a la	0	0	b1	b0	0	0	0	0
	b7	b6	b5	b4	b3	b2	b1	b0	and con 11110000 y desplazo 4 lugares a la derecha	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	X	X	X	X	-	-	-	-		0	0	0	0	c7	c6	c5	c4
	or entre ambos resultados y me queda:									0	0	b1	b0	c7	c6	c5	c4
3	b7	b6	b5	b4	b3	b2	b1	b0	and con 00001111 y	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	-	-	-	-	X	X	X	X	desplazo 2 lugares a la	0	0	c3	c2	c1	c0	0	0
	c7	c6	c5	c4	c3	c2	c1	c0	and con 110000 y desplazo 6	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	X	X	-	-	-	-	-	-	lugares a la derecha	0	0	0	0	0	0	d7	d6
	or entre ambos resultados y me queda:									0	0	c3	c2	c1	c0	d7	d6
4	c7	c6	c5	c4	c3	c2	c1	c0	and con 00111111	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	-	-	X	X	X	X	X	X		0	0	c5	c4	c3	c2	c1	c0

Figura 1: Codificación de elementos

	Char								Operación	Char							
1	a7	a6	a5	a4	a3	a2	a1	a0	Desplazo 2 a izquierda	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	0	0	X	X	X	X	X	X		a5	a4	a3	a2	a1	a0	0	0
	b7	b6	b5	b4	b3	b2	b1	b0	Desplazo 4 a derecha	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	0	0	X	X	-	-	-	-		0	0	0	0	0	0	b5	b4
	or entre ambos resultados y me queda:									a5	a4	a3	a2	a1	a0	b5	b4
2	b7	b6	b5	b4	b3	b2	b1	b0	Desplazo 4 a izquierda	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	0	0	-	-	X	X	X	X		b3	b2	b1	b0	0	0	0	0
	c7	c6	c5	c4	c3	c2	c1	c0	Desplazo 2 a derecha	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	0	0	X	X	X	X	-	-		0	0	0	0	c5	c4	c3	c2
	or entre ambos resultados y me queda:									b3	b2	b1	b0	c5	c4	c3	c2
2	c7	c6	c5	c4	c3	c2	c1	c0	Desplazo 6 a izquierda	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	0	0	-	-	-	-	X	X		c1	c0	0	0	0	0	0	0
	d7	d6	d5	d4	d3	d2	d1	d0		[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
	0	0	X	X	X	X	X	X		0	0	d5	d4	d3	d2	d1	d0
	or entre ambos resultados y me queda:									c1	c0	d5	d4	d3	d2	d1	d0

Figura 2: Decodificación de elementos

3. Conclusiones

Dada la finalidad del trabajo práctico, que consistía en familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema, podemos concluir que se logró dicho objetivo al aprender a utilizar herramientas que hasta el momento no manejabamos.

Por otra parte, por el lado del programa escrito en C propiamente hablando, la dificultad fue acorde al propósito mencionado anteriormente, ya que si bien nos llevó su tiempo pensarlo y realizarlo, no fue excesivo y nos permitió en-

focarnos en le correcta utilización de las herramientas complementarías y la elaboración de un informe adaptándonos a las normas pedidas por el curso.