



República Bolivariana de Venezuela  
Universidad Nacional Experimental Politécnica  
“Antonio José de Sucre”  
Vice Rectorado Barquisimeto  
Departamento de Ingeniería Electrónica



# Práctica 4 laboratorio de diseño de sistemas de computación

Integrantes:  
Gerardo Alfonzo Campos Fonseca  
V. 27085179  
José Andrés Cortez Teran  
V. 26540824

Barquisimeto, Julio del 2021

## Índice

Índice	II
Índice de figuras	1
1. Programa DOS	2
2. Programa consola	3
3. Programa ventana	6

## Índice de figuras

## 1. Programa DOS

El código comentado se adjunta en el archivo (dos/lectura.asm)

Las llamadas a sistema en **MS-DOS** funcionan mediante la interrupción **INT 21h** la cual recibe como parámetro el código de la función en el registro **AX**. Para la elaboración del programa se utilizaron 4 llamadas a sistema, una para abrir el archivo, otra para leer, una para cerrar y finalmente una para imprimir en pantalla.

La primera función fue **716Ch** la cual sirve para crear o abrir un archivo, toma 4 parámetros adicionales al código de función. En **BX** se pasa el tipo de acceso, en este caso se tomo 0 porque se va leer el archivo. En **CX** se pasan atributos como no se hará nada especial se paso 0 que significa normal. En **DX** se pasa la acción a tomar, se pasa 1 porque se desea abrir el archivo. Finalmente en **DS:SI** se pasa un puntero al nombre del archivo. La función retorna en **AX** el handle del archivo.

La segunda función que se utiliza es **3Fh** la cual lee un arreglo de bytes de un archivo o dispositivo. Toma tres parámetros, en **BX** se pasa el handle del archivo, en **CX** el numero máximo de bytes a leer y finalmente en **DS:DX** la dirección del buffer a escribir. En este programa se toman 4kb de salida, mas de ahí el programa truncara, pero es mas que suficiente para un programa que muestre en pantalla un archivo. El programa retorna la cantidad de bytes leídos del archivo y se guarda en una variable.

La tercera función que se utiliza es **3Eh** la cual cierra un archivo. Solo toma un parámetro en **BX** el handle del archivo a cerrar.

Finalmente la ultima función utilizada fue **40h** la cual permite escribir un arreglo de bytes en un archivo o dispositivo. Toma tres parámetros en **BX** el handle del archivo o dispositivo a escribir, en este caso pasamos 1 porque es el handle de la salida estándar. En **CX** se pasa la cantidad de bytes a escribir que es igual a la cantidad de bytes leídos del archivo. Finalmente pasamos en **DS:DX** el buffer que contiene la información leída con anterioridad.

```
1  .MODEL    small
2  .stack    100h
3
4  .data
5  Bufsize = 4096
6  input_file BYTE "input.txt",0
7  inHandle WORD ?
8  bytesRead WORD ?
9  buffer BYTE Bufsize DUP(?)
10
11 .code
12 main PROC
13     mov ax,@data
14     mov ds,ax
15
16     ; Abrir el archivo
17     mov ax,716ch           ; Funcion para crear o abrir un archivo
18     mov bx,0               ; Escogemos modo lectura
19     mov cx,0               ; Atributo normal
```

```

20     mov dx,1                ; Accion: abrir archivo
21     mov si, OFFSET input_file ; Pasamos el puntero al buffer
22     int 21h                ; llamamos a MS-DOS
23     jc quit                ; Salimos si ocurre un error
24     mov inHandle, ax
25
26     ; Leer el archivo
27     mov ah, 3Fh            ; Funcion para leer un archivo o un dispositivo
28     mov bx, inHandle       ; Pasamos el "handle" del archivo
29     mov cx, Bufsize        ; Numero maximo de bytes a leer
30     mov dx, OFFSET buffer  ; Puntero al buffer
31     int 21h                ; Llamamos a MS-DOS
32     jc quit                ; Salimo si ocurre un error
33     mov bytesRead, ax      ; Guardamos en bytesRead la cantidad bytes leidos
34
35     ; Cerrar el Archivo
36     mov ah,3Eh             ; Funcion para cerrar un archivo
37     mov bx, inHandle       ; Pasamos el handle del archivo que leimos
38     int 21h                ; Llamamos a MS-DOS
39     jc quit                ; quit if error
40
41     ; Imprimir contendio en pantalla
42     mov ah, 40h            ; Funcion para escribir en un archivo o dispositivo
43     mov bx, 1              ; Handle de la salida estandar
44     mov cx, bytesRead      ; Pasamos el numero de bytes a imprimir
45     mov dx, OFFSET buffer  ; buffer pointer
46     int 21h                ; Llamamos a MS-DOS
47     jc quit                ; Salimos si ocurre un error
48
49 quit:
50     .Exit
51 main ENDP
52 END main

```

## 2. Programa consola

El código comentado se adjunta en el archivo (consola/lectura.asm)

```

1
2 .586
3 .MODEL flat, stdcall
4 OPTION CASEMAP:NONE
5 Include windows.inc
6 Include kernel32.inc
7 Include masm32.inc
8 include user32.inc
9
10 IncludeLib kernel32.lib
11 IncludeLib masm32.lib

```

```

12 includelib user32.lib
13
14
15
16
17 Main PROTO
18     Print_Text Macro txt:REQ
19     Invoke StdOut,ADDR txt
20 EndM
21 Get_Input Macro prompt:REQ,buffer:REQ
22     Invoke StdOut,ADDR prompt
23     Invoke StdIn,ADDR buffer, LengthOf buffer
24 EndM
25
26 .DATA
27 Msg1 DB "Please Type the file is name or path: ",0AH,0DH,0
28 Msg4 DB "Press Enter to Exit",0AH,0DH,0
29 CRLF DB 0AH,0DH,0
30
31
32 .DATA?
33 inbuf DB 100 DUP (?)
34
35
36 hFile          dd ?
37 FileSize       dd ?
38 hMem          dd ?
39 BytesRead     dd ?
40
41
42 .CODE
43 Start:
44
45     Get_Input Msg1, inbuf
46     ;se usa la api de windows para abrir la fila
47     invoke CreateFile,ADDR inbuf,GENERIC_READ,0,0,\
48             OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
49
50     mov      hFile,eax
51
52     ;obtencion del tamano de la fila para pedir memoria dinamica
53     invoke GetFileSize,eax,0
54
55     mov      FileSize,eax
56     inc      eax
57
58     ;pedir memoria dinamica
59     invoke GlobalAlloc,GMEM_FIXED,eax
60     mov      hMem,eax

```

```

61
62 add     eax,FileSize
63
64 mov     BYTE PTR [eax],0    ; Set the last byte to NULL so that StdOut
65                               ; can safely display the text in memory.
66
67 ;finalmente se lee la fila
68 invoke  ReadFile,hFile,hMem,FileSize,ADDR BytesRead,0
69
70 ;se cierra la fila
71 invoke  CloseHandle,hFile
72
73 ;se escribe la fila
74 invoke  StdOut,hMem
75
76 ;se libera la memoria dinamica
77
78 invoke  GlobalFree,hMem
79
80 ;espera enter para salir y poder leer
81 Print_Text CRLF ;salto de linea
82 Get_Input Msg4,inbuf ;mensaje de salida
83
84
85 ;sale del programa
86 Invoke ExitProcess,0
87
88
89 End Start

```

Inicialmente es directivas e inclusión de librerías, luego viene el prototipo de funciones y el macro que permiten el controlar la entrada y salida de datos. Cabe resaltar que se utilizo como modelo la practica 2, posteriormente, vienen los segmentos de data y código:

En el segmento de data inicializada “**.DATA**” se encuentran:

- “Msg1” el cual es un mensaje que indica al usuario cuando debe ingresar un path o el nombre del archivo.
- “Msg4” mensaje que indica la salida del programa.
- “CRLF” string dedicada a dejar una linea en blanco.

En el segmento de data sin inicializar “**.DATA?**” se encuentran:

- “inbuf” buffer para la lectura de datos.
- “hfile” “puntero” a la dirección de la fila que se leerá.
- “FileSize” espacio para el tamaño, en bytes, del fichero.

- “hMem” “puntero” a la memoria creada con el contenido del fichero.
- “BytesRead” cantidad de bytes que se leyeron.

Posteriormente, comienza el código que básicamente es ejecutar “syscall” a través de la directiva “INVOKE”.

Primero se pide por consola y se lee por la misma el nombre o path de la fila que se abrirá. Una vez recibido se usa la api de windows con “CreateFile” para abrir la fila Microsoft, 2017, este devuelve la dirección en “eax” por lo tanto se mueve a hfile. Luego, se utiliza la api nuevamente con “GetFileSize” Microsoft, 2021c y se mueve el valor devuelto de “eax” a “FileSize”, se incrementa el valor de “eax” para al pedir memoria dinámicamente, necesario si la fila es muy grande, poder agregar el carácter de terminación y usar de forma segura la función de escritura.

Se pide la memoria dinámica con la api “GlobalAlloc” Microsoft, 2021d, se guarda en “hMem”, se le da el valor del tamaño de la fila de nuevo a “eax” para poder usarlo como puntero y darle el valor “0” (carácter de terminación) a “hMem” (espacio para el contenido de la fila + carácter terminación).

Finalmente se lee el contenido con la función de la api ReadFile Microsoft, 2021f se cierra la fila con “CloseHandle” Microsoft, 2021a luego se imprime por pantalla el contenido de la fila. Se libera la memoria con “GlobalFree” y para que se visualice correctamente el resultado, se imprime un salto de línea y se manda a leer esperando por un enter para finalizar.

El programa termina con “Invoke ExitProcess” para terminar la ejecución correctamente.

### 3. Programa ventana

El código comentado se adjunta en el archivo (ventana/DIALOG.asm y ventana/recursos.rc)

```

1  .386
2  .model flat,stdcall
3  option casemap:none
4
5  include \masm32\include\windows.inc
6  include \masm32\include\user32.inc
7  include \masm32\include\kernel32.inc
8  Include \masm32\include\masm32.inc
9
10 includelib \masm32\lib\user32.lib
11 includelib \masm32\lib\kernel32.lib
12 IncludeLib \masm32\lib\masm32.lib
13
14 WinMain proto :DWORD, :DWORD, :DWORD, :DWORD
15 DlgProc PROTO :HWND, :DWORD, :DWORD, :DWORD
16
17
18 Get_Input Macro prompt:REQ,buffer:REQ
19     Invoke StdOut,ADDR prompt

```



```

20  Invoke StdIn,ADDR buffer, LengthOf buffer
21 EndM
22
23
24 .data
25 ClassName db "SimpleWinClass",0
26 AppName db "Our Main Window",0
27 MenuName db "FirstMenu",0
28 DlgName db "MyDialog",0
29 TestString db "Hello, everybody",0
30 hwndDlg dd 0 ; Handle to the dialog box
31
32
33 .data?
34 hInstance HINSTANCE ?
35 CommandLine LPSTR ?
36
37 inbuf DB 100 DUP (?)
38 hFile dd ?
39 FileSize dd ?
40 hMem dd ?
41 BytesRead dd ?
42
43 .const
44 IDM_EXIT equ 1
45 IDM_ABOUT equ 2
46 IDC_EDIT equ 3000
47 IDC_BUTTON equ 3001
48 IDC_EXIT equ 3002
49
50 .code
51 start:
52  invoke GetModuleHandle, NULL;necesario para manejar el modulo
53  mov hInstance,eax
54  invoke GetCommandLine ;invocacion de terminal, no vienen por defecto
55  invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT;llamada al proceso que lo
    hace todo
56  invoke ExitProcess,eax
57
58 ;proceso que lo hace todo...
59 WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD;guardando
    valores iniciales, pasados en invoke
60 ;fijando variables locales
61 LOCAL wc:WNDCLASSEX;creacion de una clase
62 LOCAL msg:MSG
63 LOCAL hwnd:HWND
64 ;asignacion de los valores y direcciones a procedimientos necesarios en la clase
65 mov wc.cbSize,SIZEOF WNDCLASSEX
66 mov wc.style, CS_HREDRAW or CS_VREDRAW

```

```

67  mov     wc.lpfnWndProc, OFFSET WndProc;procedimiento asociado a la clase que se encarga
        del manejo, win process
68  mov     wc.cbClsExtra,NULL
69  mov     wc.cbWndExtra,NULL
70  push    hInst
71  pop     wc.hInstance
72  mov     wc.hbrBackground,COLOR_WINDOW+1
73  mov     wc.lpszMenuName,OFFSET MenuName
74  mov     wc.lpszClassName,OFFSET ClassName
75  invoke  LoadIcon,NULL,IDI_APPLICATION ;para el icono de la ventana
76  mov     wc.hIcon,eax
77  mov     wc.hIconSm,eax
78  invoke  LoadCursor,NULL,IDC_ARROW ;estilo del cursor
79  mov     wc.hCursor,eax
80  invoke  RegisterClassEx, addr wc ;fijacion de registros-datos con la direccion-
        informacion de la clase wc (windows class)
81
82  ;utilizacion de la api de windows para crear, sin mostrar ni cargar una ventana, donde
        la info viene de la clase llenada arriba
83  INVOKE  CreateWindowEx,WS_EX_CLIENTEDGE,ADDR ClassName,ADDR AppName,\
84          WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,\
85          CW_USEDEFAULT,300,200,NULL,NULL,\
86          hInst,NULL
87  mov     hwnd,eax ;direccion de la ventana
88  ;api de windows para
89  INVOKE  ShowWindow, hwnd,SW_SHOWNORMAL ;mostrar la ventana
90  INVOKE  UpdateWindow, hwnd ;actualizar la ventana
91  ;ciclo "infinito" usado para mantener el programa corriendo y hacer una especie de
        framework manual, donde dependiendo de los
92  ;eventos que ocurran se toman distintas medidas
93  .WHILE TRUE
94          INVOKE  GetMessage, ADDR msg,NULL,0,0 ;api de windows que permite "leer"
        mensajes-informacion de otras ventanas asociadas
95          .BREAK .IF (!eax) ;sale si eax es null (0)
96          .if hwndDlg!=0 ; si no se ha destruido la ventana
97              invoke  IsDialogMessage,hwndDlg,ADDR msg
98              .if eax==TRUE
99                  .continue
100             .endif
101         .endif
102         INVOKE  TranslateMessage, ADDR msg
103         INVOKE  DispatchMessage, ADDR msg
104     .ENDW ;fin del while
105     mov     eax,msg.wParam
106     ret
107 WinMain endp
108 WndProc proc hwnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
109     .IF uMsg==WM_DESTROY
110         invoke  PostQuitMessage,NULL;destruccion de la ventana

```

```

111 .ELSEIF uMsg==WM_COMMAND
112     mov eax,wParam
113     .if ax==IDM_ABOUT
114         invoke CreateDialogParam,hInstance, addr DlgName,hWnd,OFFSET DlgProc,NULL;llama al
           proceso de dialogo de ventana a traves de la api
115         mov hWndDlg,eax
116     .else
117         invoke DestroyWindow, hWnd
118     .endif
119 .ELSE
120     invoke DefWindowProc,hWnd,uMsg,wParam,lParam
121     ret
122 .ENDIF
123 xor     eax,eax
124 ret
125 WndProc endp
126
127 DlgProc PROC hWnd:HWND,iMsg:DWORD,wParam:WPARAM, lParam:LPARAM
128     .if iMsg==WM_INITDIALOG
129         invoke GetDlgItem,hWnd,IDC_EDIT
130         invoke SetFocus,eax
131     .elseif iMsg==WM_CLOSE
132         invoke EndDialog,hWnd,NULL
133         mov hWndDlg,0 ;se cerro la ventana, translada la info al while de arriba
134     .elseif iMsg==WM_COMMAND
135         mov eax,wParam
136         mov edx,eax
137         shr edx,16
138         .if dx==BN_CLICKED
139             .if eax==IDC_EXIT
140                 invoke SendMessage,hWnd,WM_CLOSE,NULL,NULL
141             .elseif eax==IDC_BUTTON
142
143                 invoke GetDlgItemText, hWnd,IDC_EDIT,addr inbuf,2000;lee el texto del input box,
           maximo de 2000 caracteres
144
145                 invoke CreateFile,ADDR inbuf,GENERIC_READ,0,0,\
           OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
146
147                 mov     hFile,eax
148                 ;obtencion del tamano de la fila para pedir memoria dinamica
149                 invoke GetFileSize,eax,0
150                 mov     FileSize,eax
151                 inc     eax
152                 ;pedir memoria dinamica
153                 invoke GlobalAlloc,GMEM_FIXED,eax
154                 mov     hMem,eax
155                 add     eax,FileSize
156                 mov     BYTE PTR [eax],0 ; Set the last byte to NULL so that StdOut
           ; can safely display the text in memory.
157

```

```

158     ;finalmente se lee la fila
159     invoke ReadFile,hFile,hMem,FileSize,ADDR BytesRead,0
160     ;se cierra la fila
161     invoke CloseHandle,hFile
162     ;se escribe la fila
163     invoke StdOut,hMem
164     ;MessageBox
165     invoke MessageBox, NULL, hMem, addr inbuf,
166     MB_OK
167     ;se libera la memoria dinamica
168     invoke GlobalFree,hMem
169
170     ;invoke SetDlgItemText,hWnd,IDC_EDIT,ADDR TestString
171     .endif
172     .endif
173     .else
174     mov eax,FALSE
175     ret
176     .endif
177     mov eax,TRUE
178     ret
179 DlgProc endp
180 end start

```

Para este programa se utilizó como base los tutoriales de “Iczelion” Iczelion, s.f. mas específicamente el 11-1, ya que en la carpeta 11-3 estaba prácticamente el código hecho y se quería intentar hacer a cuenta propia.

Esta estructura da acceso a una ventana principal con un submenú superior, desde el cual se puede cerrar el programa y se puede abrir una ventana emergente con un cuadro de texto y 2 botones. Aquí es en donde se basaron todas las modificaciones.

Para el funcionamiento del código, se incluyó la librería “masm32” y se declararon variables adicionales en la sección “.data?”

- “inbuf” buffer para la lectura de datos.
- “hfile” “puntero” a la dirección de la fila que se leerá.
- “FileSize” espacio para el tamaño, en bytes, del fichero.
- “hMem” “puntero” a la memoria creada con el contenido del fichero.
- “BytesRead” cantidad de bytes que se leyeron.

Cabe resaltar que el programa cuenta de estructuras que no se modificaron ya que parecían apropiadas, estas son constantes y el nombre de las ventanas. El segmento de código hace uso de 4 invoke, el primero es para poder manejar los módulos “GetModuleHandle”, el segundo “GetCommandLine” permite hacer uso de una “CMD”, el tercero, “WinMain” le da el control del programa a un subproceso descrito posteriormente y el final para cerrar los procesos.

“Invoke WinMain” le da el control del programa a un subproceso y le pasa como parámetro las variables `hInstance`, `NULL`, `CommandLine`, `SW_SHOWDEFAULT`, con las cuales se creará la ventana `main`, estas se inicializan y posteriormente se crea un objeto, `struct`, y a él se le asignan los valores, variables y procedimientos requeridos, el tamaño de la ventana, el color, los nombres de los objetos gráficos, estilo de cursor entre otros, el más importante es “`mov wc.lpfnWndProc, OFFSET WndProc`”, aquí se le da a la clase la dirección de otro subproceso definido y explicado posteriormente, en el cual se hará todo el proceso de búsqueda, lectura y escritura del fichero.

Después de estas definiciones se usan las directivas “`INVOKE ShowWindow`” y “`INVOKE UpdateWindow`” que crean una ventana y la actualizan con toda la información proveída desde la clase.

Posteriormente, se entra en un ciclo `while true` el cual hace la función de una especie de “framework” ya que mantiene el programa en ejecución a la espera de que un evento suceda y sea necesario realizar alguna acción, en este caso “`GetMessage`” se encarga de verificar si ocurre algún evento, “`eax`” guarda cualquier posible cambio y si es 0 indica la finalización del programa.

“`WndProc`” es otro proceso construido y es llamado automáticamente por la clase (ya que se le asigna su dirección, y es llamado cuando es requerido) y, básicamente, se encarga de revisar si algún evento sucedió, y si sucedió enviar los mensajes con “`PostQuitMessage`” si se destruyó la ventana, para finalizar el proceso mencionado anteriormente (asigna el valor 0 a “`eax`”), cuando este no es el caso, se trata de una serie de condicionales anidados para ir revisando todos los posibles eventos y tomar las medidas necesarias en cada caso. Estos son:

- Si “`uMsg==WM_DESTROY`”, condición de terminación, se envía el mensaje 0 (null) en “`eax`” al “framework” (“`.while true`”) para terminar su ejecución.
- Si no y “`uMsg==WM_COMMAND`”, hace la comparación de la entrada al submenú.
  - Si “`ax==IDM_ABOUT`” crea la nueva ventana y le da la dirección a otro proceso definido posteriormente donde se realizan todo lo relacionado a él el “`InputBox`” y los botones que permiten ingresar el nombre del fichero. Cabe resaltar que “`IDM_ABOUT`” no es más que el identificador de la ventana emergente creada que contendrá todos los elementos anteriormente nombrados, este nombre puede ser cambiado, requeriría modificar aquí y en el archivo “`recursos.rc`” ya que aquí se define el “`InputBox`”.
  - Si no, la única otra opción disponible implica cerrar, así que llama al proceso “`DestroyWindow`”
- En el caso que no se haya cumplido nada de esto, llama a “`DefWindowProc`” para definir la ventana y regresa el control.

El proceso “`DlgProc`” es igualmente una secuencia de condicionales anidados:

- “`.if iMsg==WM_INITDIALOG`” se inicializa la ventana de diálogos y se centra la atención en “`eax`”. “`inputbox`”
- “`.elseif iMsg==WM_CLOSE`” el mensaje es para cerrar la pestaña. se clickeo el salir de la ventana.

- “elseif iMsg==WM\_COMMAND” Ha ocurrido algún evento dentro de la ventana, particularmente se trabajan los botones:
  - “.if eax==IDC\_EXIT” se presiono el botón “salir” y se cierra la ventana, mandando un mensaje de terminación con “SendMessage”.
  - “.elseif eax==IDC\_BUTTON” se presiono el botón para la lectura del texto. entonces se hace:
    - “GetDlgItemText” para obtener el texto en el “InputBox” y guardarlo en “inbuf”.
    - “CreateFile” para abrir la fila especificada en “inbuf”.
    - “GetFileSize” tomar el tamaño de la fila.
    - “GlobalAlloc” reservar memoria dinámica. Se hacen las mismas consideraciones que en el programa consola.
    - “ReadFile” lee la fila.
    - “CloseHandle” cierra la fila.
    - “StdOut” escribe el contenido en “hMem”.
    - “MessageBox” Manda una ventana emergente con el contenido del fichero.
    - “GlobalFree” libera la memoria pedida.
- Si no se ejecuta ninguna de estas opciones, se devuelve “False” en “eax”.

Si algo de eso ocurrió, devuelve “True” en “eax”.

Con ese ultimo procedimiento finaliza todo el código.

## Referencias Bibliográficas

- Iczelion. (s.f.). <http://www.movsd.com/icz.htm>
- Irvine, K. R. (2002). Assembly Language for Intel Assembly Language for Intel-Based Computers, 4 Computers, 4th Edition Edition [fecha de consulta: 19/7/2021]. [https://www.csie.ntu.edu.tw/~acpang/course/asm\\_2004/slides/chapt\\_08Solve.pdf](https://www.csie.ntu.edu.tw/~acpang/course/asm_2004/slides/chapt_08Solve.pdf)
- Microsoft. (2017). CreateFileA function (fileapi.h) [fecha de consulta: 22/07/2021]. <https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea>
- Microsoft. (2021a). CloseHandle function (handleapi.h) [fecha de consulta: 22/7/2021]. <https://docs.microsoft.com/en-us/windows/win32/api/handleapi/nf-handleapi-closehandle>
- Microsoft. (2021b). GetDlgItemTextA function (winuser.h) [fecha de consulta: 24/7/2021]. <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getdlgitemtexta>
- Microsoft. (2021c). GetFileSize function (fileapi.h) [fecha de consulta: 22/7/2021]. <https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-getfilesize>
- Microsoft. (2021d). GlobalAlloc function (winbase.h) [fecha de consulta: 22/7/2021]. <https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-globalalloc>
- Microsoft. (2021e). How to get the string value of editbox in the variable in win32 application? [fecha de consulta: 24/7/2021]. <https://social.msdn.microsoft.com/Forums/vstudio/en-US/17c2d97a-011b-4fb1-9563-4f095d9321e4/how-to-get-the-string-value-of-editbox-in-the-variable-in-win32-application?forum=vcgeneral>
- Microsoft. (2021f). ReadFile function (fileapi.h) [fecha de consulta: 22/7/2021]. <https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile>
- OVERFLOW, S. (2015). Getting string input and displaying input with DOS interrupts MASM [fecha de consulta: 19/7/2021]. <https://stackoverflow.com/questions/29504516/getting-string-input-and-displaying-input-with-dos-interrupts-masm>
- to 2012, T. M. F. A. 2. (s.f.). how to read from file in masm [fecha de consulta: 22/7/2021]. <http://www.masmforum.com/board/index.php?topic=16266.0>