WriteFile

WriteFile Function:

The WriteFile function is used to write data to a file or an output handle. The handle can represent a file or another output destination like the screen buffer.

The function writes data to the file starting at the position indicated by the file's internal position pointer.

After the write operation is completed, the file's position pointer is adjusted by the number of bytes actually written.

hFile: This is the handle to the file or output destination where the data should be written.

lpBuffer: It's a pointer to the buffer containing the data you want to write.

nNumberOfBytesToWrite: Specifies how many bytes should be written to the file.

lpNumberOfBytesWritten: A pointer to an integer that will hold the number of bytes actually written after the operation is completed.

lpOverlapped: This should be set to NULL for synchronous operation. It's used for asynchronous operations. The return value is zero if the function fails, and it's a non-zero value if the write operation is successful.

SetFilePointer Function:

The SetFilePointer function is used to move the position pointer of an open file. This function is handy for appending data to a file or for performing random-access record processing. It's often used to navigate within a file.

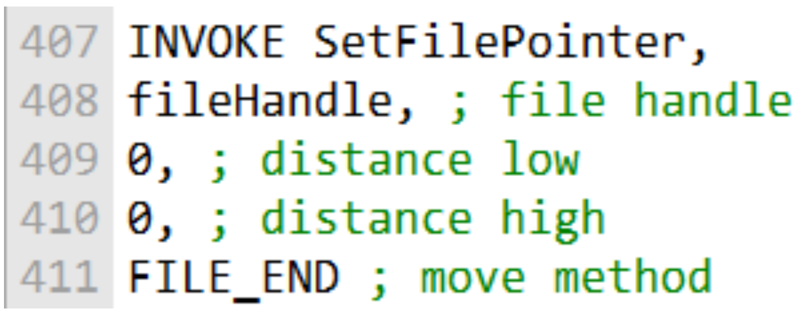
hFile: The file handle represents the file you want to move the pointer within.

lDistanceToMove: This is the number of bytes you want to move the pointer. It can be positive or negative, allowing you to move forward or backward within the file.

lpDistanceToMoveHigh: This is a pointer to a variable that contains the upper 32 bits of the distance. It's used for handling large file sizes, and if it's set to NULL, only the value in lDistanceToMove is considered.

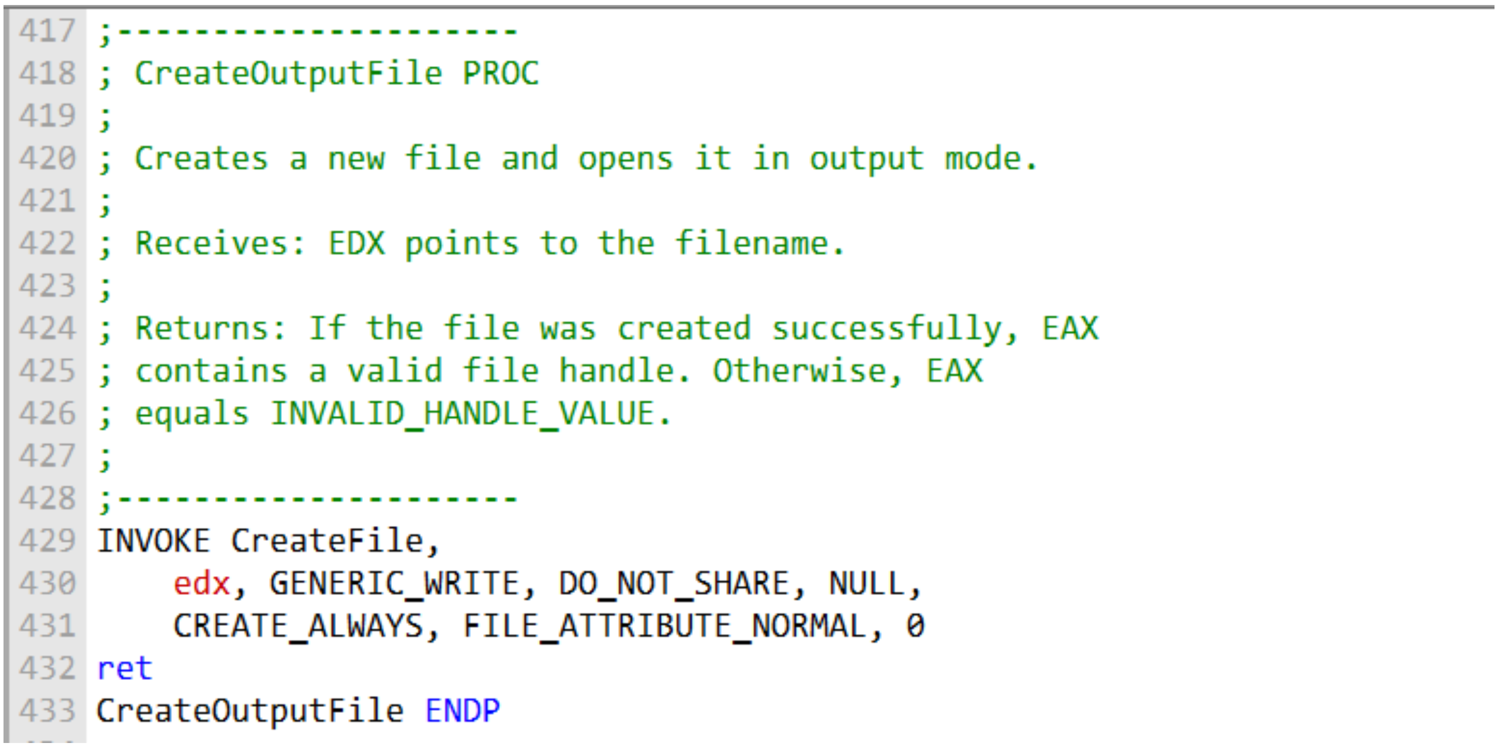
dwMoveMethod: Specifies the starting point for moving the file pointer and can take one of three values: FILE_BEGIN (absolute file positioning), FILE_CURRENT (relative to the current file position), and FILE_END (relative to the end of the file).

For example, to prepare to append data to the end of a file, you can use FILE_END as the move method:

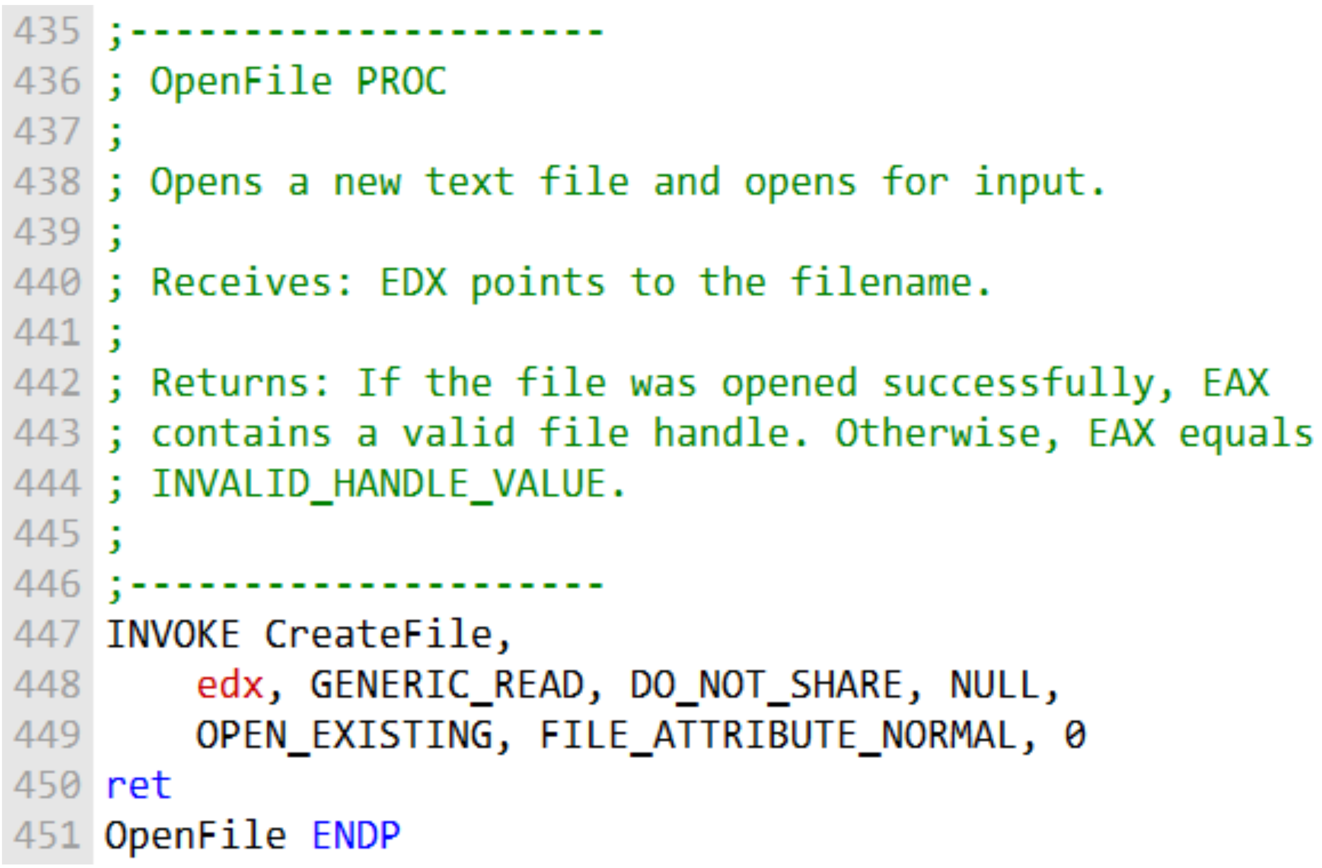


```
INVOKE SetFilePointer,
fileHandle, ; file handle
0, ; distance low
0, ; distance high
FILE_END ; move method
```

These functions are crucial for managing file access and data writing in Windows programming. They are often used in sequence, with SetFilePointer positioning the file pointer to the desired location, and WriteFile writing data to that location. Proper usage of these functions ensures efficient file manipulation in Windows applications.



```
;--------------------
; CreateOutputFile PROC
;
; Creates a new file and opens it in output mode.
;
; Receives: EDX points to the filename.
;
; Returns: If the file was created successfully, EAX
; contains a valid file handle. Otherwise, EAX
; equals INVALID_HANDLE_VALUE.
;
;--------------------
INVOKE CreateFile,
    edx, GENERIC_WRITE, DO_NOT_SHARE, NULL,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0
ret
CreateOutputFile ENDP
```



```
;--------------------
; OpenFile PROC
;
; Opens a new text file and opens for input.
;
; Receives: EDX points to the filename.
;
; Returns: If the file was opened successfully, EAX
; contains a valid file handle. Otherwise, EAX equals
; INVALID_HANDLE_VALUE.
;
;--------------------
INVOKE CreateFile,
    edx, GENERIC_READ, DO_NOT_SHARE, NULL,
    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0
ret
OpenFile ENDP
```

```
;---------------------
; WriteToFile PROC
;
; Writes a buffer to an output file.
;
; Receives: EAX = file handle, EDX = buffer offset,
; ECX = number of bytes to write
;
; Returns: EAX = number of bytes written to the file.
; If the value returned in EAX is less than the
; argument passed in ECX, an error likely occurred.
;
;---------------------
.data
WriteToFile_1 DWORD ?
; number of bytes written
.code
INVOKE WriteFile,
    eax,        ; file handle
    edx,        ; buffer pointer
    ecx,        ; number of bytes to write
    ADDR WriteToFile_1, ; number of bytes written
    0           ; overlapped execution flag
mov eax, WriteToFile_1 ; return value
ret
WriteToFile ENDP
```

```
;--------------------
; ReadFromFile PROC
;
; Reads an input file into a buffer.
;
; Receives: EAX = file handle, EDX = buffer offset,
; ECX = number of bytes to read
;
; Returns: If CF = 0, EAX = number of bytes read; if
; CF = 1, EAX contains the system error code returned
; by the GetLastError Win32 API function.
;
;--------------------
.data
ReadFromFile_1 DWORD ?
; number of bytes read
.code
INVOKE ReadFile,
    eax,        ; file handle
    edx,        ; buffer pointer
    ecx,        ; max bytes to read
    ADDR ReadFromFile_1, ; number of bytes read
    0           ; overlapped execution flag
mov eax, ReadFromFile_1
ret
ReadFromFile ENDP
```

```
;--------------------
; CloseFile PROC
;
; Closes a file using its handle as an identifier.
;
; Receives: EAX = file handle
;
; Returns: EAX = nonzero if the file is successfully closed.
;
;--------------------
INVOKE CloseHandle, eax
ret
CloseFile ENDP
```

That was the first program to test your knowledge, now let's do the second one:

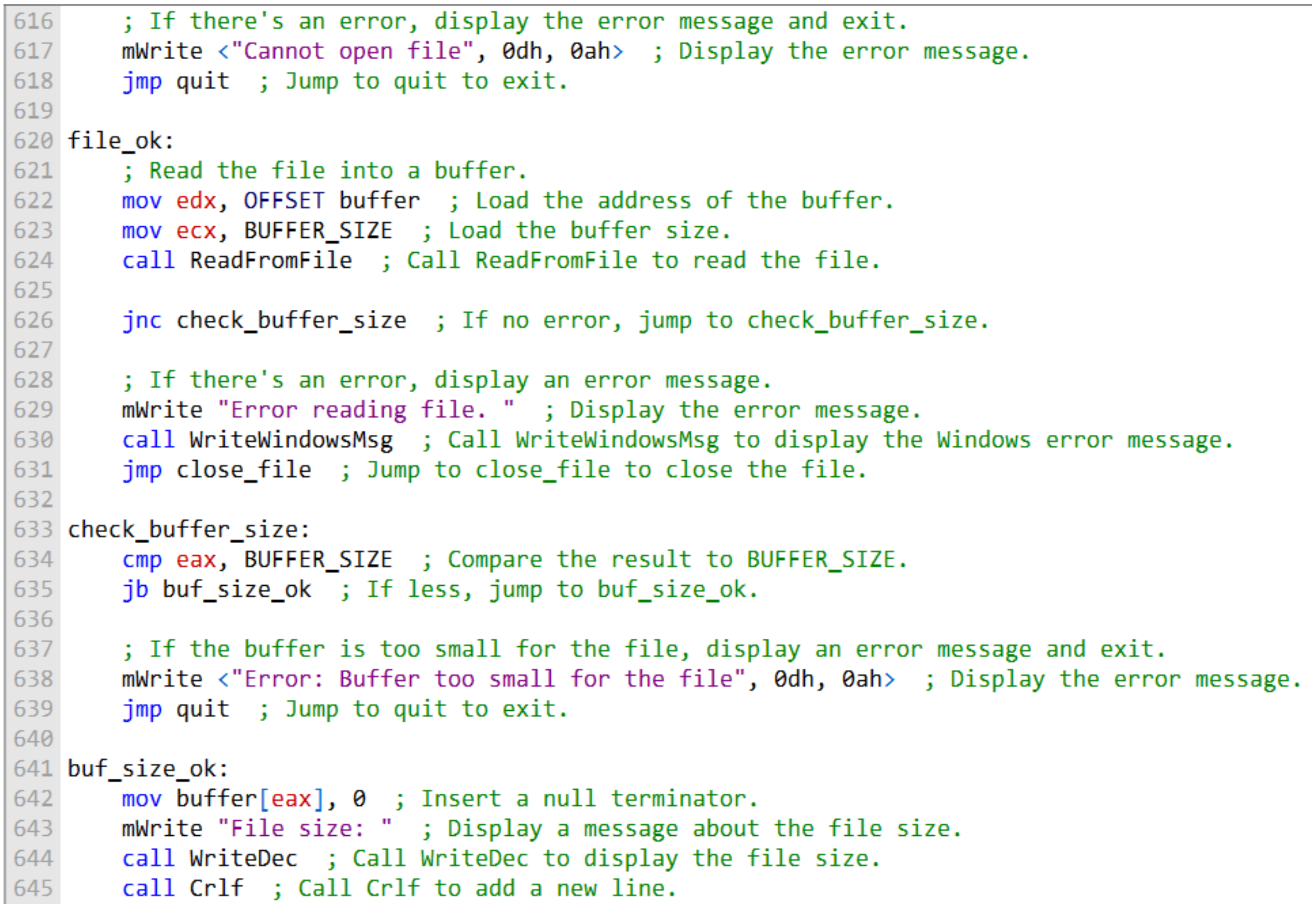
```
; Creating a File (CreateFile.asm)
INCLUDE Irvine32.inc
BUFFER_SIZE = 501

.data
buffer BYTE BUFFER_SIZE DUP(?)
filename BYTE "output.txt",0
fileHandle HANDLE ?
stringLength DWORD ?
bytesWritten DWORD ?
str1 BYTE "Cannot create file",0dh,0ah,0
str2 BYTE "Bytes written to file [output.txt]:",0
str3 BYTE "Enter up to 500 characters and press [Enter]: ",0dh,0ah,0

.code
main PROC
    ; Create a new text file.
    mov edx, OFFSET filename    ; Load the address of the filename.
    call CreateOutputFile       ; Call the CreateOutputFile procedure.
    mov fileHandle, eax         ; Store the file handle in fileHandle.

    ; Check for errors.
    cmp eax, INVALID_HANDLE_VALUE ; Compare the result to INVALID_HANDLE_VALUE.
    jne file_ok                  ; If not equal, jump to file_ok.

    ; If there's an error, display the error message and exit.
    mov edx, OFFSET str1        ; Load the address of the error message.
    call WriteString            ; Call WriteString to display the error message.
    jmp quit                    ; Jump to quit to exit.
```

```
file_ok:
    ; Ask the user to input a string.
    mov edx, OFFSET str3          ; Load the address of the input prompt.
    call WriteString              ; Call WriteString to display the input prompt.

    mov ecx, BUFFER_SIZE          ; Load the maximum buffer size.

    ; Input a string.
    mov edx, OFFSET buffer        ; Load the address of the buffer.
    call ReadString               ; Call ReadString to get user input.
    mov stringLength, eax         ; Store the length of the entered string.
    ; Write the buffer to the output file.
    mov eax, fileHandle           ; Load the file handle.
    mov edx, OFFSET buffer        ; Load the address of the buffer.
    mov ecx, stringLength        ; Load the length of the string.
    call WriteToFile              ; Call WriteToFile to write to the file.
    mov bytesWritten, eax         ; Store the number of bytes written.
    ; Close the file.
    call CloseFile                ; Call CloseFile to close the file.
    ; Display the return value.
    mov edx, OFFSET str2          ; Load the address of the output message.
    call WriteString              ; Call WriteString to display the message.
    mov eax, bytesWritten         ; Load the number of bytes written.
    call WriteDec                 ; Call WriteDec to display the value.
    call Crlf                     ; Call Crlf to add a new line.
quit:
    exit
main ENDP
END main
```

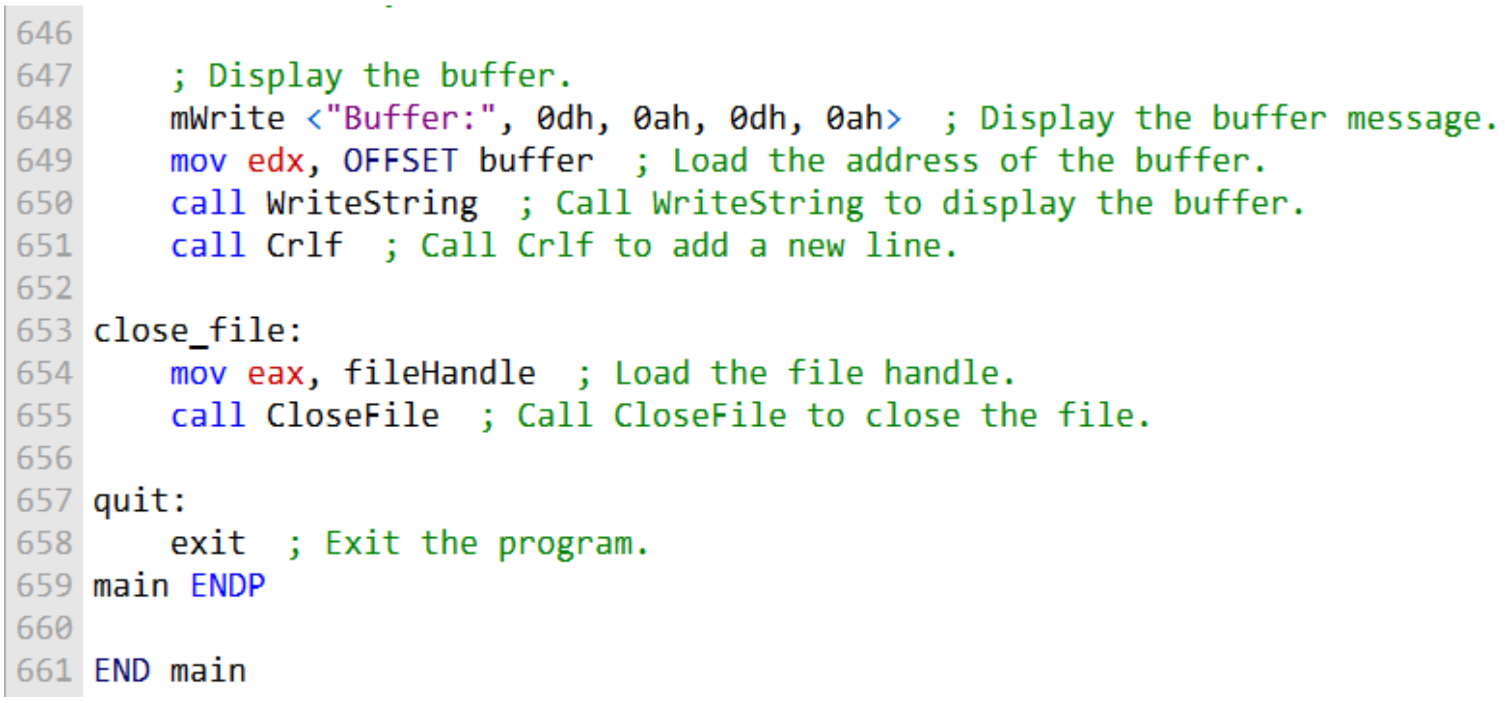
That's the second program.

Let's try another program:

```
; Reading a File (ReadFile.asm)
; Opens, reads, and displays a text file using
; procedures from Irvine32.lib.
INCLUDE Irvine32.inc
INCLUDE macros.inc
BUFFER_SIZE = 5000

.data
buffer BYTE BUFFER_SIZE DUP(?)
filename BYTE 80 DUP(0)
fileHandle HANDLE ?

.code
main PROC
    ; Let the user input a filename.
    mWrite "Enter an input filename: "  ; Display the input prompt.
    mov edx, OFFSET filename  ; Load the address of the filename.
    mov ecx, SIZEOF filename  ; Load the size of the filename.
    call ReadString  ; Call ReadString to get user input.

    ; Open the file for input.
    mov edx, OFFSET filename  ; Load the address of the filename.
    call OpenInputFile  ; Call OpenInputFile to open the file.
    mov fileHandle, eax  ; Store the file handle in fileHandle.

    ; Check for errors when opening the file.
    cmp eax, INVALID_HANDLE_VALUE  ; Compare the result to INVALID_HANDLE_VALUE.
    jne file_ok  ; If not equal, jump to file_ok.
```



```
    ; If there's an error, display the error message and exit.
    mWrite <"Cannot open file", 0dh, 0ah>  ; Display the error message.
    jmp quit  ; Jump to quit to exit.

file_ok:
    ; Read the file into a buffer.
    mov edx, OFFSET buffer  ; Load the address of the buffer.
    mov ecx, BUFFER_SIZE  ; Load the buffer size.
    call ReadFromFile  ; Call ReadFromFile to read the file.

    jnc check_buffer_size  ; If no error, jump to check_buffer_size.

    ; If there's an error, display an error message.
    mWrite "Error reading file. "  ; Display the error message.
    call WriteWindowsMsg  ; Call WriteWindowsMsg to display the Windows error message.
    jmp close_file  ; Jump to close_file to close the file.

check_buffer_size:
    cmp eax, BUFFER_SIZE  ; Compare the result to BUFFER_SIZE.
    jb buf_size_ok  ; If less, jump to buf_size_ok.

    ; If the buffer is too small for the file, display an error message and exit.
    mWrite <"Error: Buffer too small for the file", 0dh, 0ah>  ; Display the error message.
    jmp quit  ; Jump to quit to exit.

buf_size_ok:
    mov buffer[eax], 0  ; Insert a null terminator.
    mWrite "File size: "  ; Display a message about the file size.
    call WriteDec  ; Call WriteDec to display the file size.
    call Crlf  ; Call Crlf to add a new line.
```



```

    ; Display the buffer.
    mWrite <"Buffer:", 0dh, 0ah, 0dh, 0ah>  ; Display the buffer message.
    mov edx, OFFSET buffer  ; Load the address of the buffer.
    call WriteString  ; Call WriteString to display the buffer.
    call Crlf  ; Call Crlf to add a new line.

close_file:
    mov eax, fileHandle  ; Load the file handle.
    call CloseFile  ; Call CloseFile to close the file.

quit:
    exit  ; Exit the program.
main ENDP

END main
```

That's the 3rd program.