CreateOutputFile: *(Irvine32 only)* The CreateOutputFile procedure creates a new disk file and opens it for writing. When you call the procedure, place the offset of a filename in EDX. When the procedure returns, EAX will contain a valid file handle (32-bit integer) if the file was created successfully. Otherwise, EAX equals INVALID_HANDLE_VALUE (a predefined constant).
Sample call:

```
.data
filename BYTE "newfile.txt",0
.code
mov edx,OFFSET filename
call CreateOutputFile
```

OpenInputFile: *(Irvine32 only)* The OpenInputFile procedure opens an existing file for input. Pass it the offset of a filename in EDX. When it returns, if the file was opened successfully, EAX will contain a valid file handle. Otherwise, EAX will equal INVALID_HANDLE_VALUE (a predefined constant).
Sample call:

```
.data
filename BYTE "myfile.txt",0
.code
mov edx,OFFSET filename
call OpenInputFile
```

ReadFromFile (*Irvine32 only)* The ReadFromFile procedure reads an input disk file into a memory buffer. When you call ReadFromFile, pass it an open file handle in EAX, the offset of a buffer in EDX, and the maximum number of bytes to read in ECX. When ReadFromFile returns, check the value of the Carry flag: If CF is clear, EAX contains a count of the number of bytes read from the file. But if CF is set, EAX contains a numeric system error code. You can call the WriteWindowsMsg procedure to get a text representation of the error. In the following example, as many as 5000 bytes are copied from the file into the buffer variable:

```
.data
BUFFER_SIZE = 5000
buffer BYTE BUFFER_SIZE DUP(?)
bytesRead DWORD ?
.code

mov edx, OFFSET buffer    ; points to buffer
mov ecx, BUFFER_SIZE      ; max bytes to read
call ReadFromFile         ; read the file
```

If the Carry flag were clear at this point, you could execute the following instruction:

```
 mov bytesRead, eax       ; count of bytes actually read
```

But if the Carry flag were set, you would call <u>WriteWindowsMsg</u> procedure, which displays a string that contains the error code and description of the most recent error generated by the application:

```
    call WriteWindowsMsg
```

WriteToFile *(Irvine32 only)* The WriteToFile procedure writes the contents of a buffer to an output file. Pass it a valid file handle in EAX, the offset of the buffer in EDX, and the number of bytes to write in ECX. When the procedure returns, if EAX is greater than zero, it contains a count of the number of bytes written; otherwise, an error occurred. The following code calls <u>WriteToFile</u>:

```
    .data
    BUFFER_SIZE = 5000
    fileHandle DWORD ?
    buffer BYTE BUFFER_SIZE DUP(?)
    .code
    mov eax, fileHandle
    mov edx, OFFSET buffer
    mov ecx, BUFFER_SIZE
    call WriteToFile
```

CloseFile (*Irvine32 only)* The CloseFile procedure closes a file that was previously created or opened (see CreateOutputFile and OpenInputFile). The file is identified by a 32-bit integer *handle*, which is passed in EAX. If the file is closed successfully, the value returned in EAX will be nonzero. Sample call:

```
     mov     eax,fileHandle
     call    CloseFile
```

WriteWindowsMsg (*Irvine32 only*) The WriteWindowsMsg procedure displays a string containing the most recent error generated by your application when executing a call to a system function. Sample call:

```
    call WriteWindowsMsg
```

The following is an example of a message string:

```
Error 2: The system cannot find the file specified.
```

MsgBox *(Irvine32 only)* The MsgBox procedure displays a graphical popup message box with an optional caption. (This works when the program is running in a console window.) Pass it the offset of a string in EDX, which will appear in the inside the box. Optionally, pass the offset of a string for the box's title in EBX. To leave the title blank, set EBX to zero. Sample call:

```
.data
caption db "Dialog Title", 0
HelloMsg BYTE "This is a pop-up message box.", 0dh,0ah
BYTE "Click OK to continue...", 0
.code
mov ebx,OFFSET caption
mov edsx,OFFSET HelloMsg
call MsgBox
```

Sample output:

MsgBoxAsk *(Irvine32 only)* The MsgBoxAsk procedure displays a graphical popup message box with Yes and No buttons. (This works when the program is running in a console window.) Pass it the offset of a question string in EDX, which will appear in the inside the box. Optionally, pass the offset of a string for the box's title in EBX. To leave the title blank, set EBX to zero. MsgBoxAsk returns an integer in EAX that tells you which button was selected by the user. The value will be one of two predefined Windows constants: IDYES (equal to 6) or IDNO (equal to 7). Sample call:

```
.data
caption BYTE "Survey Completed",0
question BYTE "Thank you for completing the survey."
BYTE 0dh,0ah
BYTE "Would you like to receive the results?",0
.code
mov ebx,OFFSET caption
mov edx,OFFSET question
call MsgBoxAsk
;(check return value in EAX)
```

ReadString*:* The ReadString procedure reads a string from the keyboard, stopping when the user presses the Enter key. Pass the offset of a buffer in EDX and set ECX to the maximum number of characters the user can enter, plus 1 (to save space for the terminating null byte). The procedure returns the count of the number of characters typed by the user in EAX. Sample call:

```
.data
 buffer BYTE 21 DUP (0)             ; input buffer
 byteCount DWORD ?                  ; holds counter
 .code
 mov edx,OFFSET buffer              ; point to the buffer
 mov ecx,SIZEOF buffer              ; specify max characters
 call ReadString                    ; input the string
 mov byteCount,eax
```

ReadString automatically inserts a null terminator in memory at the end of the string. The following is a hexadecimal and ASCII dump of the first 8 bytes of **buffer** after the user has entered the string "ABCDEFG": The variable **byteCount** equals 7.

RandomRange The RandomRange procedure produces a random integer within the range of 0 to *n*, where *n* is an input parameter passed in the EAX register. The random integer is returned in EAX. The following example generates a single random integer between 0 and 4999 and places it in a variable named *randVal*.

```
.data
randVal DWORD ?
.code
mov eax, 5000
call RandomRange
mov ranVal, eax
```