



Instructions For this programming assignment, you must write two **robust** procedures in **assembly language**. The two procedures will be included in ONE file inside ONE project and called by your main to test them. These procedures should be implemented and tested using the *Visual Studio* IDE. For all these exercises, you cannot use built-in functions that perform these operations. In case of a doubt, check with your instructor.

For this project, you will build simple strings functions. You will use basic **number** and **string** input/output functions provided by the author of the textbook.

Objectives of this assignment:

- To improve your programming skills in the assembly language
- To get familiar with *Visual Studio*
- To stress on the fact that the memory (variables) ultimately are 0s and 1s
- To use some basic **string** input/output functions
- To use some basic **byte** input/output functions

Warning:

The executable you will build may get **blocked** by an Antivirus scanner on your system. In this case, read this from the Textbook author: *“Antivirus scanner software has improved greatly in recent years, and so have the number of viruses (one website reports 50,000 at present). Because of this, your computer's antivirus scanner may report a false positive when you build your program, and refuse to let you run it. There are a few workarounds: (1) You can add your project's bin/debug folder into an exclusion list in your antivirus configuration. This is my preferred approach, and it nearly always works. (2) You can suspend your realtime antivirus scanner software, but this will leave you open to malware for a short time. If you choose this option, be sure to temporarily disconnect your computer from the Internet. (3) You can send a copy of your program's EXE file to the antivirus software vendor, labeling it as a false positive. The virus scanner I use automatically uploads any questionable EXE file to their website and either quarantines or releases the file within about 30 minutes.”* I used the first workaround, i.e., I added my project's /debug folder to the exclusion list in my antivirus configuration.

Input/Output Built-in Functions Allowed:

The textbook author wrote some helpful functions. For this project, you **can** use these functions:

- 1) **ReadString**: the `ReadString` procedure reads a string from the keyboard, stopping when the user presses the *Enter* key. You must pass two parameters before calling this function:
 - a. **Offset** of the buffer where to store the entered string: this offset must be passed in Register `EDX`
 - b. **Maximal** number of characters: this number is the maximal number of characters a user can enter. This number must be passed in Register `ECX`.

As a result, `ReadString` will store in the buffer the string entered by the user and will return in the register `EAX` the number of characters entered by the user. See excerpt from the textbook:



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            ; number of characters
```

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 "ABCDEFGH":

41	42	43	44	45	46	47	00	ABCDEFGH
----	----	----	----	----	----	----	----	----------

- 2) **WriteString**: (from the textbook) the WriteString procedure writes a null-terminated string to the console window. You must pass the string's offset in EDX. Below is a sample call from the textbook:

WriteString The WriteString procedure writes a null-terminated string to the console window. Pass the string's offset in EDX. Sample call:

```
.data
prompt BYTE "Enter your name: ",0
.code
mov     edx,OFFSET prompt
call    WriteString
```

- 3) **ReadChar**: this function will allow you to read a character from the keyboard and receive its ASCII code in the register AL. This function is described in the textbook Page 165 (beginning/page).
- 4) **WriteChar**: this function will allow you to display on the console window (command window) a character whose ASCII code is stored in the register AL. See in the textbook Page 169 (start/page).
- 5) **ReadHex**: this function reads a 32-bit hexadecimal integer from the keyboard and returns it in EAX. This function is described in the textbook Page 166 (middle/page).



Note: the two procedures must be included in ONE project using ONE assembly file. Try to create one procedure for each exercise. **Write a main program calling the three procedures.**

Programming Exercise 1 (15 points):

The objective of this exercise is to write a **procedure** that deletes n characters from a string S starting from Position P . The procedure:

- 1) Prompts the user with the message: "Please enter a sentence: "
- 2) Reads a string S from the keyboard
- 3) Prompts the user with the message: "Please enter the number of characters to delete (in hexadecimal: "
- 4) Reads the number n (use the *ReadHex* function)
- 5) Prompts the user with the message: "Please enter the position from where to start deleting (in hexadecimal: "
- 6) Reads the number P (use the *ReadHex* function)
- 7) Deletes n characters from a string S starting from Position P .
- 8) Displays the new string S (with deleted characters).

Note that your procedure must be well commented and **robust**: you must save and restore the appropriate registers.

Programming Exercise 2 (25 points):

The objective of this exercise is to write a **procedure** that inserts a string S_1 in String S_2 starting from Position P . The procedure:

- 1) Prompts the user with the message: "Please enter a sentence S_1 to insert: "
- 2) Reads a string S_1 from the keyboard
- 3) Prompts the user with the message: "Please enter a sentence S_2 in which to insert: "
- 4) Reads a string S_2 from the keyboard
- 5) Prompts the user with the message: "Please enter the position P where to insert"
- 6) Reads the number P (use the *ReadHex* function)
- 7) Inserts a string S_1 in String S_2 starting from Position P .
- 8) Displays the new string S_2 (with inserted string).

Note that your procedure must be well commented and **robust**: you must save and restore the appropriate registers.



What you need to turn in:

- Electronic copies of your report (standalone) and source code (zipped) of your program. The two exercises must be included in the same project. Clearly indicate the separation between them by using comments in main of the source program. Your project folder must be zipped folder named **m7-name**, where *name* is your lastname. Zip the project folder and post it with the standalone report on Canvas. Submit separately (not inside the zipped folder) the report as a Microsoft Word or PDF file.
- Your report must:
 - State whether your code works.
 - Clearly explain how to compile and execute your code.
 - If needed/applicable, report/analyze (as appropriate) the results. The quality of analysis and writing is critical to your grade.
- Good writing and presentation are expected.

How this assignment will be graded:

The program compiles and executes correctly without any apparent bugs. The code is well-designed and commented.	100% credit
The program compiles and executes with minor bugs. The code is well-designed and commented.	90% credit
The program compiles and executes with major bugs.	40% credit
The program compiles but does not produce any meaningful result.	5% credit

If the instructor needs additional communications/actions to compile and execute your code, then a 30% penalty will be applied. If the turn-in instructions are not correctly followed, 10 points will be deducted.