Lab #5: Simple Text Editor

Overview:

In this lab you will complete the development of a program that implements a simple text editor.

Requirements

The program should support the following features:

- allow the user to type in characters
- display typed in characters to the screen
- move the cursor to the position of the next character to be typed in
- exit if the user presses the ESC key
- allow a maximum of 10 lines
- allow a maximum of 40 characters per line
- automatically continue on new line after reaching maximum number of characters on a line
- erase the last entered character and update cursor position if the user presses BACKSPACE

Directions

1. Create an assembly language code file named c:\lab5.asm using Notepad++.

2. Type in (or copy & paste) the following code skeleton:

```assembly
.model small ;specify small memory model

;ASCII codes
BS equ 08h
ESCP equ 1bh
WS equ 20h

;Dimensions of editing area
COLUMNS equ 40
ROWS equ 10
MAX_X equ COLUMNS-1
MAX_Y equ ROWS-1
```
.stack 200h ;specify a stack size of 512 bytes
.data
    cursor_x db 0 ;cursor horizontal position
    cursor_y db 0 ;cursor vertical position
.code
start:
    mov ax,@data ;set-up ds to be able to access our data
    mov ds,ax

    ;Use BIOS interrupt 10h, Service 06h to scroll window up
    ;this creates a clear screen effect
    ;also set-up colors (blue background & red text)
    mov ax,0600h
    mov bh,1ch
    mov cx,0
    mov dx,184fh
    int 10h

set_cursor:
    ;Use BIOS interrupt 10h, Service 02h to position cursor
    mov ah,02h
    mov dl,cursor_x
    mov dh,cursor_y
    mov bh,0
    int 10h

read_key:
    ;Use BIOS interrupt 16h, Service 00h to read keyboard
    ;(returns ASCII code in al)
    mov ah,0
    int 16h

    ;TODO: Step #4 here

    ;TODO: Step #5 here

    ;Use BIOS interrupt 10h, service 0ah to print character
    ;at current cursor position
    mov ah,0ah
    mov cx,1
    mov bh,0
    int 10h

    ;TODO: Step #6 here

    ;TODO: Step #7 here

move_down:
    ;TODO: Step #8 here

    ;TODO: Step #9 here

backspace:
3. Take a close look at the above code and pay attention to the comments. Make sure that you understand what the given pieces of code do before proceeding any further.

4. Locate the `read_key` label. After reading the keyboard, add code to make a conditional branch to the `exit` label if the entered key is `ESCAPE`. 
   HINT: Use pre-defined constant ESCP

5. Immediately after the code you added on Step #4, add code to make a conditional branch to the `backspace` label if the entered key is `BACKSPACE`.  
   HINT: Use pre-defined constant BS

6. Notice that if other than ESCAPE or BACKSPACE is entered, the code will fall-thru to print the entered character (using BIOS interrupt 10h, service 0ah). After that, you should add a conditional branch to the `move_down` label if the
cursor position has reached the maximum allowed horizontally. HINT: check the `cursor_x` variable against the `MAX_X` constant.

7. If the maximum horizontal cursor position has not been reached yet, the code you added on Step #6 will fall-thru. You need to handle that case by adding code to update the cursor position and go back to read the next entered key. HINT: increment the `cursor_x` variable and do an unconditional branch to the `set_cursor` label.

8. Locate the `move_down` label. Notice that we get there from the code that you added on Step #6, when the maximum horizontal cursor position is reached. At this point we are getting ready to move the cursor to a new line (down). However, before doing that you need to add code to check if we already reached the maximum number of allowed line. If so, we will simply ignore the input key. HINT: check the `cursor_y` variable against the `MAX_Y` constant, if match do a branch to the `read_key` label.

9. If the maximum vertical cursor position has not been reached yet, the code you added on Step #8 will fall-thru. At this point you should add code to update the cursor position at the beginning of the next line and go back to read the next entered key. HINT: make `cursor_x` = 0, increment `cursor_y` and do an unconditional branch to the `set_cursor` label.

10. Locate the `backspace` label. Notice that we get there from the code that you added on Step #5, when the user press the BACKSPACE key. In such case you should erase the last entered character. But before doing that, you need to add code to check if the cursor is at the beginning of a line and, if so, handle appropriately. HINT: do a conditional branch to the `move_up` label if `cursor_x` = 0

11. If the cursor is not at the beginning of a line, the code you added on Step #10 will fall-thru. To handle that case you need to add code to move the cursor to the previous position (left) and erase the last character. HINT: decrement `cursor_x` and branch unconditionally to the `erase` label.

12. Locate the `move_up` label. Notice that we get here from the code that you added on Step #10 when the last character that needs to be erased is at the end of the previous line. At this point we are getting ready to move the cursor to the
previous line. However, before doing that, you need to add code to check if we are at the first line. If so, we will simply ignore the input key.
HINT: do conditional branch to the read_key label if cursor_y = 0

13. If the cursor is not at the first line, the code you added on Step #12 will fall-thru. At this point you should add code to set the cursor position at the end of the previous line and erase the key at that position.
HINT: make cursor_x = MAX_X, decrement cursor_y and fall-thru to the code at the erase label.

14. Assemble the code using TASM. At the DOS prompt type:
   tasm /l /zi lab5.asm
   
   The “/l” switch tells the assembler to generate an output listing file (named lab5.lst) while the “/zi” tells it to generate symbolic information needed for the linker.

   Note: MASM32 equivalent command is:
   ml /c /Fl lab5.asm

15. If you get any error on your code, take a look at lab5.lst (you can use notepad). It shows the generated machine code side-by-side with your assembly language. Always pay attention to the first error as subsequent errors may be caused by the first one. Correct your error by editing lab5.asm and then go back to step #14.

16. Link the program by using TLINK. At the DOS prompt type:
   tlink /v lab5.obj
   
   The “/v” switch tells the linker to generate full symbolic information needed for the debugger.

   Note: MASM32 equivalent command is:
   link16 lab5.obj,,,,,,,,

17. Run your program and verify that it operates correctly (as described on the Requirements section).

18. If the program doesn’t operate correctly, debug it using the Turbo Debugger tool. At the DOS prompt type:
td lab5.exe

Note: For MASM32 use DEBUG. The invocation command is:
debug lab5.exe

19. If necessary, use the Turbo Debugger help (from the menu) for reference on how to use it (or for DEBUG, use the ? command).

20. If necessary, make corrections to your program by editing lab5.asm and going back to step #14.

Lab Report

To complete this lab you should provide the instructor with the following files:

lab5.asm
lab5.lst
lab5.exe

The files should be archived on a zip file and send via e-mail. The zip file should be named as follows: lab5_<your name>.zip (e.g., lab5_WalterLara.zip).