CSC 236 Programming Assignment

TABS

Summary

The TABS assignment is worth 8% of your grade. For this assignment, you will be submitting a file named **tabs.ans**. This file is created by the grading system when you grade your program. The grade is based on correct answers, documentation and the number of executable instructions used.

If you have questions about the assignment, feel free to use our EdStem forum, but be sure not to post any code from your solution in a public EdStem post (even code that's not working). You can also stop by during office hours to get help from anyone on the teaching staff. Remember, if you need help, you need to make sure your code is documented, including block comments above sections of your code and line comments saying what you are trying to do with each instruction. We'll also expect you to have a design for the program that you're trying to follow.

Specifications

You will write a program named TABS. It will read lines from an ASCII input text file which is redirected to standard input. It will perform some simple text editing on the lines, and write the updated lines to an ASCII output text file which is redirected from the standard output. The specific editing function is to expand tab characters (replace each tab with the correct number of spaces) to arrange the output into columns at particular tab stops.

The column width (also referred to as the tab stop position) for tab expansion will default to 10. However, the user can specify a different value in the range of 1 to 9 through the use of a Command Line Parameter.

The way to expand tabs for the TABS program is given in the section titled **Tabs Explained**.
The way to read the Command Line Parameter is given in the section **Accessing The Command Line Parameter**.

Program Input

Input to the program is the ASCII text from a file redirected to the standard input. The grading system will adhere to the following rules when creating input to validate your program.

- Files will contain 0 or more lines. There is no limit to the number of characters per line or lines per file.
- The first character position on a line is called position 0 (i.e., we start counting columns from column zero on the left).
- The characters in the input can be any ASCII characters in the range of 20h-7Fh or a few control characters. We will not use characters above 7Fh.

- Characters in the file can also include some control characters (those with codes in the range 00h-1Fh). The only control characters that will be used are:
  tab = 09h, line feed (LF) = 0Ah, carriage return (CR) = 0Dh and DOS End Of File (EOF) = 1Ah
- All lines will terminate with the DOS carriage return and line feed pair (0D0Ah). The characters 0Dh and 0Ah will never appear by themselves in the file; they will only appear as part of a two-byte line termination sequence.
- Files will end with a DOS text file End Of File (EOF) character (1Ah). The EOF will only appear immediately after a DOS carriage return and line feed pair (0D0Ah) or as the only character in an otherwise empty file.

A second optional input is that user may specify a tab stop position as part of a Command Line Parameter (CLP) passed to the program when the program is started. If the user specifies the tab stop position in the CLP, it will be valid digit in the range '1' .. '9' and it will consist of only one digit. No error checking is needed for this parameter. If it is provided, it will adhere to this format.

Program Output For Each Input Line

Write all characters in the input file to the output file except tab characters (09h). Tabs need to be expanded by writing spaces (20h) until you reach the next tab stop position. If the input file does not contain any tabs then the output file will be byte-for-byte identical to the input file, including the carriage return, line feed and end of file characters.

Creating your Program

Step 1 : Create a design

TABS is complex enough that most developers will require a design in order to successfully implement the program in assembly language. You may wish to code TABS in C or Java as your design. There is sample code for reading the keyboard and writing to the display in the Readfile example from the KEY assignment.

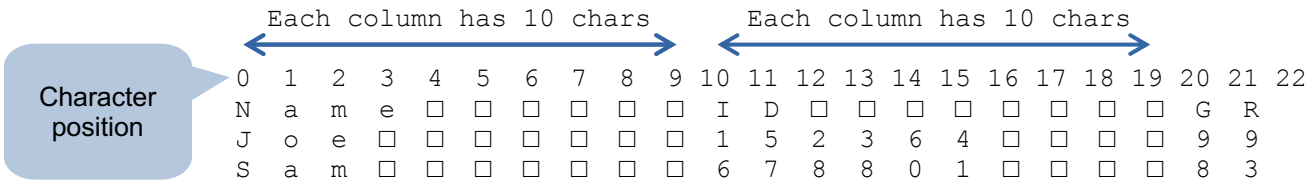
Expanding Tabs

The function of TABS is to help the user to organize data into columns, each column containing the same number of characters.

Normally, when you press the tab key, it does not mean put out a fixed number of spaces. That would not really help to maintain fixed-width columns. Instead, the tab key typically means to add whatever number of spaces are needed to get to the next tab stop position. For the TABS program, 10 will be the default interval between tab stops. Every time you press the tab key you go to the next column that is a multiple of 10 (10, 20, 30, 40 ...). The following is the basic algorithm for replacing a tab with spaces for the tabs program. The first input and output positions in a line are position number 0.

```
If the current input character is a tab then
While (true)
{ Output a space to the current output position.
  Advance to the next output position.
  If the new current output position is a tab stop position then break. }
```

Here is an example using 10 as the tab stop interval. Assume the user presses the tab key after a typing a person's name and the user presses the tab key after typing a person's ID. Here, the □ symbol is used to indicate a space character.



A tab stop interval of 1 is valid. It just just means you advance to the next character position.

Accessing The Command Line Parameter (CLP)

When you invoke an assembly language program, you can pass a set of command-line parameters to the program. These command line-parameters are normally used to provide information to the program. For example, in the command:

```
tab 5
```

we are indicating that we want the tab stop value to be 5 instead of the default value of 10.

The CLPs are passed to the program in a control block called the Program Segment Prefix (PSP). When a program is invoked, both the DS and ES register point to the start of the PSP. This is the reason that you need to initialize the DS register to point to your data. Even after changing DS, the ES register should still point the PSP. Using a technique called *segment override*, you can access the data in the PSP pointed to by the ES register.

The command line parameters are located 128 (80H) bytes from the start of the PSP.

- The byte 80H from the start of the PSP is the count of the number of characters that were entered as command-line parameters. This includes everything typed after the name of the program the user entered.
- There will be a space at 81H and the next ASCII character typed by the user will be in hex 82. This makes sense. Normally, the user would type in a command, then a space before they start giving command-line parameters for the command. For an assembly language run from DOS, we get to see exactly the sequence of characters the user typed after the command name. It's not like C, where the command-line parameters are broken up into individual words by the shell.

Example 1, the user does not pass a command-line parameter; they just enter the program name: **tab**

PSP

Character	00	0D	?	?
Offset	80h	81h	82h	83h

Example 2, the user passes 5 as the tab stop interval: **tab 5**

PSP

Character	02	20	35	0D
Offset	80h	81h	82h	83h

The following code can be used to test whether a Command Line Parameter was typed and to retrieve it:

```
cmp byte ptr es:[80h], 0     ; access the CLP count 80h bytes into the extra segment
je noclp                     ; no parameter ... continue with the program
mov al, byte ptr es:[82h]    ; parameter entered ... load al with the character
                             ; after the space
```

Remember this is an ASCII character, so the code for the digit five would be 35h. It must be converted to a numeric value (e.g., 5) before it can be used as the tab interval. If the user specifies the tab stop position, you are guaranteed it will be valid and in the range of '1' to '9'. No error checking is expected for this part of the program.

Step 2 : Code your solutions

Next, you can code your design into assembly language instructions. Your source file should be named **tabs.asm** and the executable will be named **tabs.exe**. Use the following commands to assemble and link your program using MASM:

Compile	`ml /c /Zi /Fl tabs.asm`
Link	`link /CO tabs.obj`

Retrieve the testing and grading files. All files are packed together in one self-extracting file named **unpack.exe**. It's with the rest of the materials for the TABS assignment on our Moodle page. You can click on the unpack.exe link to download this executable file and save it on your hard drive in the TABS subdirectory of your shared P23X directory.

Once you have a copy of the unpack.exe archive in the right place, you can run the usual commands to get ready to work. Then, change to the TABS directory and unpack the archive you just downloaded. Depending on where you put your shared directory, it will probably be something like the following:

```
mount e ~/P23X
E:
dbset
cd TABS
unpack
```

Step 3 : Test and debug

Develop test cases

We provide 4 test cases for your use. They are named **tabin.1** to **tabin.4**

To use the default tab stop position try out these tests with the following command, where *t* is one of the test numbers, 1 .. 4:

```
testtabs tabin.t
```

To specify a tab stop interval, *n*, you can run the command as follows:

```
testtabs tabin.t n
```

When you run your program via testtabs like this, your program's output will go into a file named **testout** and the correct output will go into a file named **okay**.

You may use any editor you wish to create your own test cases. However, an ordinary text editor may not store tabs, it may expand the tabs as you type them. So, we provide a program named **makefile.exe** (no relation to the make command or makefiles used in CSC 230). If you run this program in DOSBox, it will let you type int text for a test case. To run this simple editor, enter:

```
makefile output_file_name
```

The makefile program will allow you to type data into the file. This includes tabs, which will be entered into the file as bytes with the value hex 09.

The makefile program will terminate when you type a period. It will automatically insert the correct end of line marker for the last line in the file and it will put a DOS text file end-of-file character at the end.

The makefile program doesn't really support file editing. It just records characters as you type them and saves them at the end. If you make a mistake typing in the text you want to test with, you'll need to re-run makefile and try again.

After you create a file, you should use the testfile program to view the contents of the file in hexadecimal. This will let you see exactly what your TABS program will see when it reads the input.

To test your solution efficiently, you will want to start with the simplest test cases first. It is much easier to isolate the defect with a simple test case than with a complicated test case.

View your Output

As a quick check, you can look at your program's output in the DOSBox window. This should reveal any obvious errors in your program.

Of course, even if your programs output looks right, there may still be errors in the control characters (CR, LF, EOF). You'll want to verify that your output is really correct; it must match the expected output exactly. If it varies at all from the expected output then it will not be considered correct by the grading program. See the section below to about how to make sure your program's output is exactly right.

Checking Expected Output

We provide a batch file that will show you the correct output for any valid input file. To run it type:

```
gradokay tabin.t
     or
gradokay tabin.t n
```

The output will go to a file named **okay**. You can look at that file with the **testfile** program to see the exact hex output that should be generated for the input. Use this to check on any details about what should be in a correct output file for a particular test case.

Checking Documentation

You can tell if the grading system likes your documentation. Just enter the following command:

```
graddoc tabs.asm
```

The system will check your documenation and tell you how many out of the 20 points it would earn.

Debugging your Program

You can use the debugger to isolate the cause of any defect uncovered by your tests. You can run CodeView Debug file input and output and with command-line parameters. The following will run it with input from tabin.1 and with output going to a file named testout:

```
cv tabs < tabin.# > testout
```

If you need to try command-line parameters, you can give those before the I/O redirection syntax:

```
cv tabs 7 < tabin.1 > testout
```

Checking if Output is Exactly Right

How can I verify the content and control characters in my output file for my tests or the grading system's tests? Remember there are two programs introduced in the TOOLS assignment that can help. The **testfile** program will show you all the characters in a file, including all the control characters. The **compfile** program will compare two files.

You can run a test of your program by typing:

```
testtabs tabin.n
```

Then, you can compare the files okay and testout:

```
compfile okay testout
```

If the files have different content, then the compfile will show you those differences.

During a grading run, your program's output is placed in a file named **testout** and the expected output for the same test case is placed in the file named **okay**. The compfile program will let you compare these two files for visible characters and control characters.

Getting Help

If you need help while developing your solution, you can visit the teaching staff during office hours, or you can post a question on EdStem. Remember, never post part of your solution (even a non-working solution) in a public EdStem post.

To get help, you will need to make sure you have a coherent design for your project that you're trying to follow. You'll also need to make sure your partial solution is documented, to help describe what you're trying to do in each part of your program.

If your problem is with the grading system then make sure you have the **testout** and **okay** files generated by the grading system. This will make it easier to look into what's going wrong.

Step 4 : Grading

Run the grading system to determine how many points your program has earned. Go to the TABS subdirectory. If you successfully unpacked the archive for the TABS assignment, you should be able to run the grading program by entering:

```
gradtabs
```

To pass the grading tests, your programs' output must match the expected, correct output exactly. Since some of the output characters (e.g., control characters) won't really be visible in the DOSBox window, don't rely on just a visual comparison to know if your output is correct. Instead, use the **compfile** or **testfile** programs to see where your program's output might not match the expected output, or to see exactly what's in your output.

Even after your program is working, you may need to test it multiple times, to try to to improve efficiency or documentation. You can make more than one grading run. To mark these as grading runs that were only made to improve efficiency or documentation, execute this command after your program has earned the 60 points for getting the correct answers.

```
testtabs mark
```

All subsequent grading runs will be marked as only being done to improve efficiency or documentation.

The final grade for your solution will be based on:

- 60 points for getting the correct answers.
- 20 points for the number of executable instructions written to correctly complete this assignment. The goal is an efficient program. Your source code will be analyzed and you will be awarded 0 to 20 points, with the most points awarded to the programs with the fewest instructions. You can find the target instruction count on our Moodle page, in the section about he TABS program.
- 20 points for documentation of a program that functions correctly.

Efficiency and documentation are only a grading concern after the code works totally correctly. So, Therefore, efficiency and documentation grading will only occur after your code passes all the functional tests.

Step 5 : Submit your Assignments

Electronically submit the file **tabs.ans** created by the grading system. This file is submitted to the TABS assignment in Moodle. This is the file we're expecting when we're assigning you a grade for your solution.

It is your responsibility to make sure:

- Your **tabs.ans** file contains two concatenated sections. First the results file and then a copy of your programs source from **tabs.asm**.

- Your **tabs.ans** contains the line:
  `++ Grade ++ nnn = Total grade generated by the Grading System.`

Incorrect electronic submissions will result in your program not being graded and considered late.

Do not edit or modify any of the grading system files. This is especially important for the files named **results** and **tabs.ans**, which contain the grading results. Any modification of grading system files will look like an academic integrity violation.