TABS

Print and use
the specification document
on the class web site.

See syllabus for assignment type
individual

Wish to create a simple text file
where the data lines up in columns

```
Name       ID        Grade
Joe        12345     73
```

Wish to create a simple text file
where the data lines up in columns

Each Column is 10 characters

```
01234567890123456789012345678 9
Name       ID        Grade
Joe        12345     73
```

Wish to create a simple text file
where the data lines up in columns

Each Column is 10 characters

```
01234567890123456789012345678 9
Name␣␣␣␣␣␣␣ID␣␣␣␣␣␣␣␣␣Grade
Joe␣␣␣␣␣␣␣12345␣␣␣␣␣73
```

But I do not want to have to count
the number of spaces ␣ needed
to reach the next column

Tabs can be used to create columns of text

Given *tabstop* = 10

tab ≡ go to next col that is multiple of 10

Each Column is 10 characters

```
01234567890123456789012345678 9
Name      ID        Grade
Joe       12345     73
```

ASCII text file

Go to next tab stop position
Column that is multiple of 10

N	a	m	e	09h	I	D	09h	G	r	a	d	e	0Dh	0Ah

J	o	e	09h	1	2	3	4	5	09h	7	3	0Dh	0Ah

Each Column is 10 characters

```
01234567890123456789012345678 9
Name      ID        Grade
Joe       12345     73
```

ASCII text file

Go to next tab stop position
Column that is multiple of 10

N	a	m	e	09h	I	D	09h	G	r	a	d	e	0Dh	0Ah

J	o	e	09h	1	2	3	4	5	09h	7	3	0Dh	0Ah

When displaying or printing the file
your editor must expand the tabs by
replacing them with the correct number of
spaces to create columns

TABS
(program processing and output)

The TABS program will read
an ASCII text file
redirected to the standard input
so you can use int 21h with ah=8 and ah=2
and it will expand any tabs with spaces
to create columns

TABS
(program processing and output)

- The default tabstop is 10
  - Stop at cols that are multiples of 10
    (1st col on a line is 0)

- Processing for each input line
  - Read each character
  - If not a tab then write it to the output

All characters except tabs
are written to the output file

TABS
(program processing and output)

- The default tabstop is 10
  - Stop at cols that are multiples of 10
    (1st col on a line is 0)

- Processing for each input line
  - Read each character
  - If not a tab then write it to the output
  - If a tab then write spaces until the
    next tabstop position is reached

- Terminate the program after the
  DOS end of file char (1Ah) is read
  and written

Reading a user specified tabstop 1-9
(Optionally specified when program started)

Use default value of 10

Reading a user specified tabstop 1-9
(Optionally specified when program started)

Use default value of 10

Use value entered 1-9
specified by the
Command Line Parameter

Reading a user specified tabstop 1-9
(Optionally specified when program started)

The Command Line Parameter
is passed to the program
in a special control block

Use value entered 1-9
specified by the
Command Line Parameter

Reading a user specified tabstop 1-9
(Optionally specified when program started)

The code to read the
Command Line Parameter
is provided in the specification

Use value entered 1-9
specified by the
Command Line Parameter

Step 1. Create a design

Software design represents the methodology used to create a working program from a specification document.

It assures the problem is understood well enough to be solved.

Step 1. Create a design

An option ... write TABS in Java or C

Data in → TABS → Check output

Only use basic HLL statements

Model Java and C code
in the Readfile course locker

To simplify your program the grading system will adhere to these rules

- only data chars 20h-7Fh
- only control chars . . . . .tab, cr, lf, eof
- all lines terminate with . . . cr/lf
- cr and lf never appear separately
- file termination will only occur at the start of a new line

- Any tabstop specified in the CLP will be valid ... an ASCII character '1' - '9'

No error checking needed

Step 2. Code your assembler solution

Your source code named *tabs.asm*

Retrieve *unpack.exe* from *tabs* locker

Put it in the \P23X\TABS directory

In DOSBox type *unpack* to build the grading system

Step 3. Test and debug your solution

We provide 4 test cases for your use
tabin.1 *tabin.2* *tabin.3* *tabin.4*

All in the correct format

Step 3. Test and debug your solution

We provide 4 test cases for your use
tabin.1 *tabin.2* *tabin.3* *tabin.4*

Run test against your program

testtabs tabin.1 — *default tabstop*

testtabs tabin.1 7 — *specify tabstop*

The output will be a file named *testout*

The correct output is in a file *okay*

Step 3. Test and debug your solution

We also provide a simple editor for creating files with tabs

makefile *output_file_name*

A→B$_{enter}$
C→→D.

41	09	42	0D	0A	43	09	09	44	0D	0A	1A

Step 3. Test and debug your solution

We also provide a simple editor for creating files with tabs

makefile *output_file_name*

A→B$_{enter}$
C→→D.

Period terminates program

41	09	42	0D	0A	43	09	09	44	0D	0A	1A

Step 3. Test and debug your solution

3.3 Correct output for any valid input

gradokay tabin.#

gradokay tabin.# n

input_file → TABS → okay

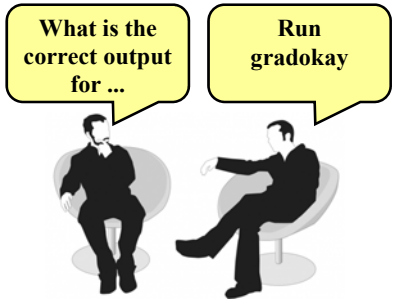
The output will be a file named *okay*

Step 3. Test and debug your solution

3.3 Correct output for any valid input

gradokay tabin.#

gradokay tabin.# n



Step 4. Grading

gradtabs

If errors look in the file named *results*

The final grade will be based on:

- **60 points for the correct answers**
- **20 points for executable instrs written**
- **20 points for documentation**

Step 5. Submit your assignment

tabs.ans

(This is the only acceptable file)

Additional Thoughts

- **If you limit all constants to immediate data and all variables to registers, you will not need to initialize the DS register**
- **You need a counter to control writing spaces to expand tabs**

  Counting up using *add* requires a *compare* to test for completion

  Counting down using *sub* does not need a compare against zero

- **Check out the *loop* instruction**

loop can help control how many times code executes a loop
Class Notes ... Chapters 6