TABS

CSC 236

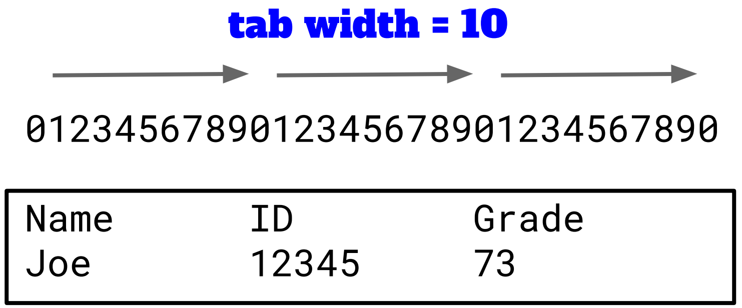
What

- Complete specification on web
- Type: individual

- Convert tabs into spaces
    - Read stdin
    - Write stdout
    - Expand tabs to spaces
    - Redirection for files: ie, `tabs < infile.txt > outfile.txt`

ASCII columns

- Want columns to line up
- But do not want to count spaces



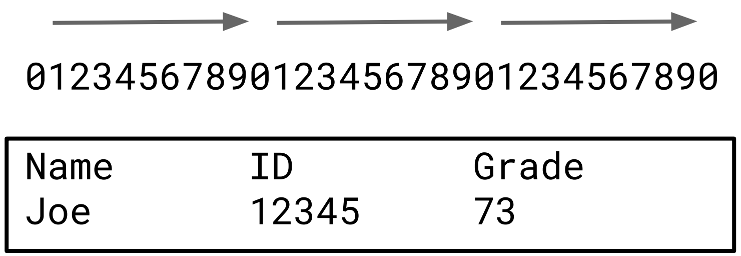
```
0123456789012345678901234567890

Name      ID        Grade
Joe       12345     73
```

ASCII columns

- Want columns to line up
- But do not want to count spaces

tab width = 10



```
012345678901234567890123456789
```

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

Input file:

```
Name\tID\tGrade
Joe\t12345\t73
```

ASCII columns

- Want columns to line up
- But do not want to count spaces

tab width = 10

```
0123456789012345678901234567890
```

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

Input file:

```
Name\tID\tGrade\r\nJoe\t12345\t73\r\n\1A
```

Details

- Program reads ASCII text file
    - Redirected to the standard input
- Output ASCII text file
    - Redirected from standard output
- ⇒ int 21h with ah=8 (read char) and ah=2 (write char)
- Action
    - Replace tab character (\t, 9h)
    - With 1 or more spaces (20h)
- Default tab stop is 10
    - First column is 0

Operations

- Process each line of input
    - Read each character
    - If not tab ⇒ write it out
    - If tab ⇒
        - write spaces (20h)
        - until next tab stop
    - Terminate on DOS EOF (1Ah)

Command line parameter

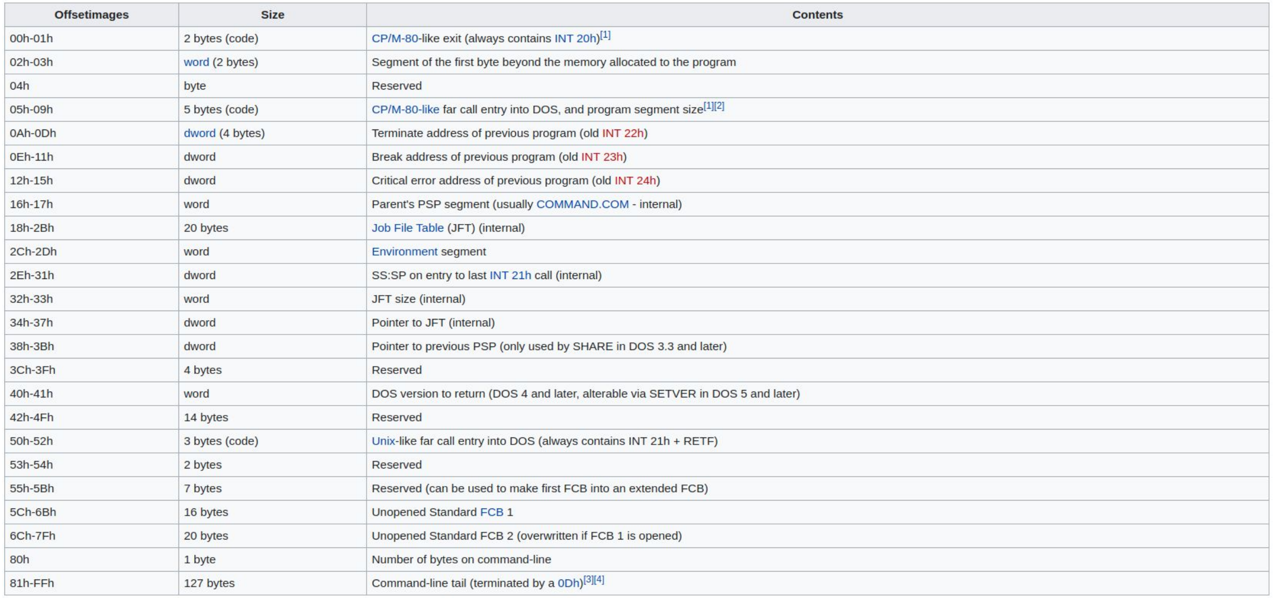
- Program has optional parameter
  - Tab stop
  - Valid values 1 to 9
- Example
  - `tabs` —— **use tabstop = 10**
  - `tabs 5` —— **use tabstop = 5**
  - `tabs 12` —— **invalid input -- will not test**

Find command line parameters

- In memory -- of course
  - Put there by DOS
  - Program Segment Prefix

Find command line parameters

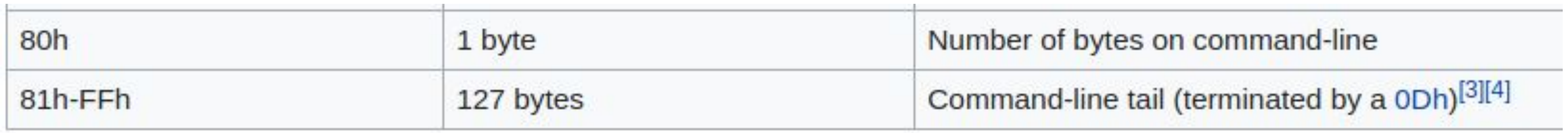
- In memory -- of course
  - Put there by DOS
  - Program Segment Prefix



Offsetimages	Size	Contents
00h-01h	2 bytes (code)	CP/M-80-like exit (always contains INT 20h)[1]
02h-03h	word (2 bytes)	Segment of the first byte beyond the memory allocated to the program
04h	byte	Reserved
05h-09h	5 bytes (code)	CP/M-80-like far call entry into DOS, and program segment size[1][2]
0Ah-0Dh	dword (4 bytes)	Terminate address of previous program (old INT 22h)
0Eh-11h	dword	Break address of previous program (old INT 23h)
12h-15h	dword	Critical error address of previous program (old INT 24h)
16h-17h	word	Parent's PSP segment (usually COMMAND.COM - internal)
18h-2Bh	20 bytes	Job File Table (JFT) (internal)
2Ch-2Dh	word	Environment segment
2Eh-31h	dword	SS:SP on entry to last INT 21h call (internal)
32h-33h	word	JFT size (internal)
34h-37h	dword	Pointer to JFT (internal)
38h-3Bh	dword	Pointer to previous PSP (only used by SHARE in DOS 3.3 and later)
3Ch-3Fh	4 bytes	Reserved
40h-41h	word	DOS version to return (DOS 4 and later, alterable via SETVER in DOS 5 and later)
42h-4Fh	14 bytes	Reserved
50h-52h	3 bytes (code)	Unix-like far call entry into DOS (always contains INT 21h + RETF)
53h-54h	2 bytes	Reserved
55h-5Bh	7 bytes	Reserved (can be used to make first FCB into an extended FCB)
5Ch-6Bh	16 bytes	Unopened Standard FCB 1
6Ch-7Fh	20 bytes	Unopened Standard FCB 2 (overwritten if FCB 1 is opened)
80h	1 byte	Number of bytes on command-line
81h-FFh	127 bytes	Command-line tail (terminated by a 0Dh)[3][4]

Find command line parameters

- In memory -- of course
  - Put there by DOS
  - Program Segment Prefix



80h	1 byte	Number of bytes on command-line
81h-FFh	127 bytes	Command-line tail (terminated by a 0Dh)[3][4]

80h is the offset into the PSP.

But where is the PSP?

Find command line parameters

- In memory -- of course
  - Put there by DOS
  - Program Segment Prefix
    - DS & ES point to the PSP
  - Problem:
    - You want to copy from PSP into DS
    - How do you address the PSP and data segments?

Segment override

- Default segments
  - Instruction — code (CS)
  - Stack — stack (SS)
  - Data — data (DS)
- How to access ES?
  - Segment override
  - `es:[offset]`
  - `es:[80h]` — access bytes in command line in the PSP
- TABS spec explains it in detail, including example code

Step 1 — design

Create design

- pseudocode
- flowchart

For simplicity

- No error checking on input required
- Data chars — 20h-7Fh
- Control chars — tab, cr, lf, eof
- All lines terminate with cr/lf
    - Neither cr nor lf never appear separately
    - CR — \r, $0D_{16}$, 13_{10}
    - LF — \n, $0A_{16}$, 10_{10}
- EOF ($1A_{16}$) always at start of a new line
- Tabstop parameter will be valid
    - An ASCII character '1' - '9'

Step2 — code

- Name source code file 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 & debug

- 4 test cases provided
  - tabin.1, ..., tabin.4
- Use testing program
  - `testtabs tabin.1`
  - `testtabs tabin.1 7`
  - Outputs file named “testout” (your output)
  - Outputs file named “okay” (correct output)
- makefile.exe
  - Simple editor
  - Create input for testing

Step 4 — grade

- Self grading
- gradtabs.exe
- File results will contains errors -- if any
- Grade
    - 60 pts for correctness
    - 20 pts executable instructions written
    - 20 pts documentation

Step 5 — submit

- Upload
    - File tabs.ans — no other file
    - To TABS locker

Hints

- If
  - All constants are immediate **and**
  - All variables are in register **then**
  - Do not initialize the DS register
- Counter
  - Need to count spaces to emit
  - Counting down is often better
- Loop command
  - Tests and jumps
  - Read pg 6-24

```
    mov  ax,0
L1: add  ax,1
    cmp ax,7
    jne L1
```

```
    mov ax,7
L1: sub ax,1
    jne L1
```