CM1205 Coursework

David Buchanan

March 17, 2017

1 Source listing:

```
.586                    ; Recognise 80x86 instructions that use 32-bit
.MODEL FLAT, STDCALL    ; Generate code for a flat memory model
.STACK 4096             ; Reserve 4096 bytes for stack operations
option casemap:none     ; Case sensitivity

include windows.inc     ; Configure your assembler to use the correct include directory


; Macros to simplify IO:

READ_INPUT      MACRO   buf
        invoke  ReadConsoleA,
                inHandle,
                OFFSET buf,
                SIZEOF buf,
                OFFSET bytesRead,
                NULL
        ENDM

WRITE_OUTPUT    MACRO   buf, len
        invoke  WriteConsoleA,
                outHandle,
                buf,
                len,
                OFFSET bytesWritten,
                NULL
        ENDM

; For when the length is constant:
WRITE_CONST     MACRO   buf
        WRITE_OUTPUT OFFSET buf, SIZEOF buf
        ENDM



.DATA

        promptMsg       BYTE    "Enter a temperature: "
        typeMsg         BYTE    "Would you like to convert to Celsius or Farenheit? [C/F]: "
        outputMsg       BYTE    0Dh, 0Ah, "Output: "
        exitMsg         BYTE    0Dh, 0Ah, "Press ENTER to do another conversion, or Ctrl+C to exit"
        CRLF            BYTE    0Dh, 0Ah
```

```
        CLS             BYTE    100 dup( 100 dup(' '), 0Dh, 0Ah )
        buf             BYTE    4096 dup(?) ; This length is slight overkill!
        bytesWritten    DWORD   ?
        bytesRead       DWORD   ?
        outHandle       DWORD   ?
        inHandle        DWORD   ?
        tmp             DWORD   ?
        homeCursor      COORD   {0, 0}
        boxCursor       COORD   {2, 19}

        ; these numeric constants are used for FPU instructions that don't
        ; support immediate agruments
        TEN             REAL8   10.0
        THOUSAND        REAL8   1000.0
        NINE            REAL8   9.0
        FIVE            REAL8   5.0
        THIRTY_TWO      REAL8   32.0
        HALF            REAL8   0.5

        bannerMsg \
        DB      "                                                                ", 0Dh, 0Ah
        DB      "David Buchanan presents:                                        ", 0Dh, 0Ah
        DB      " _______                                 _                      ", 0Dh, 0Ah
        DB      "|__   __|                               | |                     ", 0Dh, 0Ah
        DB      "   | | ___ _ __ ___  _ __   ___ _ __ __ _| |_ _   _ _ __ ___ ", 0Dh, 0Ah
        DB      "   | |/ _ \ '_ ` _ \| '_ \ / _ \ '__/ _` | __| | | | '__/ _ \", 0Dh, 0Ah
        DB      "   | |  __/ | | | | | |_) |  __/ | | (_| | |_| |_| | | |  __/", 0Dh, 0Ah
        DB      "   |_|\___|_| |_| |_| .__/ \___|_|  \__,_|\__|\__,_|_|  \___|", 0Dh, 0Ah
        DB      "           _____     | |                    _                 ", 0Dh, 0Ah
        DB      "          / ____|    |_|                   | |                ", 0Dh, 0Ah
        DB      "         | |     ___  _ ____   _____ _ __| |_ ___ _ __ TM     ", 0Dh, 0Ah
        DB      "         | |    / _ \| '_ \ \ / / _ \ '__| __/ _ \ '__|       ", 0Dh, 0Ah
        DB      "         | |___| (_) | | | \ V /  __/ |  | ||  __/ |          ", 0Dh, 0Ah
        DB      "          \_____\___/|_| |_|\_/ \___|_|   \__\___|_|          ", 0Dh, 0Ah
        DB      "                                                                ", 0Dh, 0Ah
        DB      "                                                                ", 0Dh, 0Ah
        DB      "                                                                ", 0Dh, 0Ah
        DB      " ____________________________________________________________ ", 0Dh, 0Ah
        DB      "|                                                            |", 0Dh, 0Ah
        DB      "|                                                            |", 0Dh, 0Ah
        msgBoxLower \ ; used when we need to redraw this section
        DB      "|____________________________________________________________|", 0Dh, 0Ah
        DB      "                                                                ", 0Dh, 0Ah
        ; used to calculate the length of the banner, assumes linear allocation
        endMsgBox       BYTE    NULL
.CODE


main PROC

        ; Acquire I/O handles
        INVOKE  GetStdHandle, STD_OUTPUT_HANDLE
        MOV     outHandle, eax
        INVOKE  GetStdHandle, STD_INPUT_HANDLE
        MOV     inHandle, eax

```

```
99          ; Ensure the FPU starts in a clean state, with round-to-nearest
100         ; mode enabled
101         FNINIT
102 prompt:
103         ; Prompt the user for temperature input
104         WRITE_CONST  CLS ; clear the screen (Why is this not part of the Windows API?!)
105         INVOKE       SetConsoleCursorPosition, outHandle, homeCursor
106         WRITE_OUTPUT OFFSET bannerMsg, OFFSET endMsgBox - OFFSET bannerMsg
107         INVOKE       SetConsoleCursorPosition, outHandle, boxCursor
108         WRITE_CONST  promptMsg
109
110         CALL readfloat
111
112         ; redraw lower portion of the box
113         WRITE_OUTPUT OFFSET msgBoxLower, OFFSET endMsgBox - OFFSET msgBoxLower
114
115 retry:
116         ; Prompt the user for conversion type
117         WRITE_CONST typeMsg
118
119         ; Read the user's choice
120         READ_INPUT buf
121
122         MOV     AL, buf ; Just check the first character
123
124         ; Convert to upper case, for case insensitivity
125         AND     AL, 11011111b
126
127         CMP     AL, 'C'
128         JNE     skipc
129         CALL    f2c
130         JMP     done
131 skipc:
132         CMP     AL, 'F'
133         JNE     retry ; Repeat the question on invalid input
134         CALL    c2f
135
136 done:
137         WRITE_CONST outputMsg
138         CALL printfloat
139         WRITE_CONST CRLF
140
141         WRITE_CONST exitMsg
142         READ_INPUT buf ; wait for the user to hit enter
143         JMP prompt ; Restart the program (Ctrl+C to exit)
144
145 main ENDP
146
147
148 f2c PROC ; Farenheit to Celsius
149
150         FSUB    THIRTY_TWO
151         FMUL    FIVE
152         FDIV    NINE
153         RET
154
```

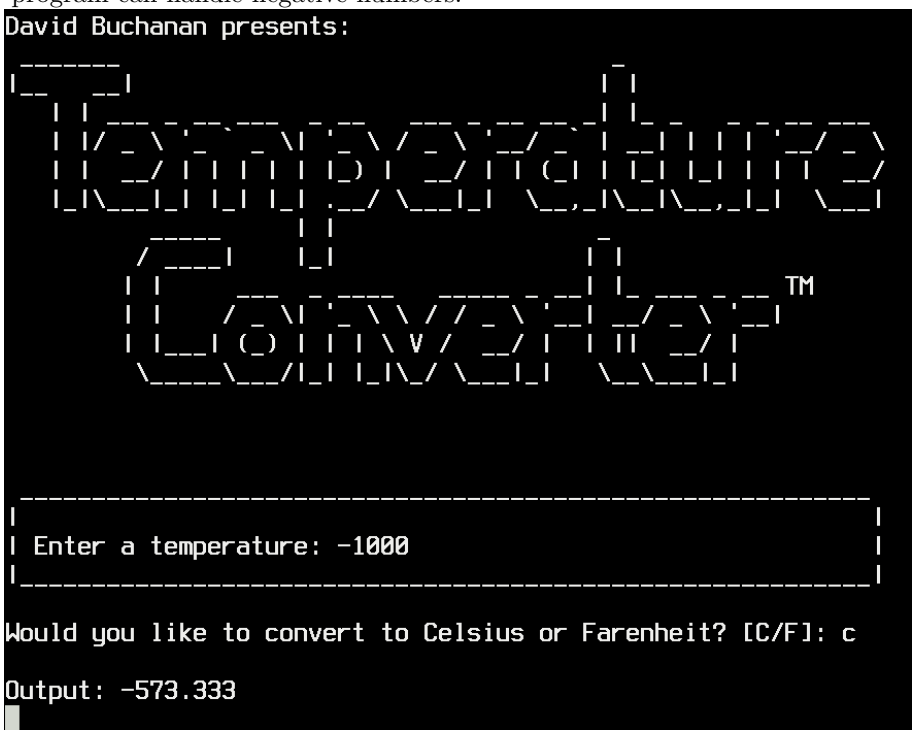
```
f2c ENDP


c2f PROC ; Celsius to Farenheit

        FMUL    NINE
        FDIV    FIVE
        FADD    THIRTY_TWO
        RET

c2f ENDP


; read a floating point number from stdin, and stores it in the ST(0)
; FPU register
readfloat PROC

        READ_INPUT buf

        MOV     ECX, bytesRead
        MOV     ESI, OFFSET buf
        XOR     EAX, EAX
        XOR     EBX, EBX ; EBX = 1, used to store the sign of the input
        INC     EBX
        FLDZ
        CLD

        parseloop:
                LODSB
                CMP     AL, 0Ah
                JE      parsedone ; end on carriage return

                CMP     AL, '-' ; if a '-' is encountered, flip the sign
                JNE     ispositive
                IMUL    EBX, -1

        ispositive:
                CMP     AL, '.'
                JE      decimal
                CMP     AL, '0'
                JL      parseloop ; skip over if less than '0'
                CMP     AL, '9'
                JG      parseloop ; skip over if greater than '9'

                FMUL    TEN
                SUB     AL, '0'
                MOV     tmp, EAX
                FIADD   tmp

                JMP parseloop

        decimal: ; works backwards from the end until the next decimal
                STD
                MOV     ESI, OFFSET buf
                ADD     ESI, bytesRead
                DEC     ESI
```

```
211                 FLDZ
212         decimalloop:
213                 LODSB
214                 CMP     AL, '.'
215                 JE      decimaldone
216                 CMP     AL, '0'
217                 JL      decimalloop ; skip over if less than '0'
218                 CMP     AL, '9'
219                 JG      decimalloop ; skip over if greater than '9'
220
221                 SUB     AL, '0'
222                 MOV     tmp, EAX
223                 FIADD   tmp
224                 FDIV    TEN
225
226                 JMP decimalloop
227
228 decimaldone:
229         FADDP ; add the integer and fractional components together
230         CLD
231
232 parsedone:
233         MOV     tmp, EBX ; apply the sign
234         FIMUL   tmp
235         RET
236
237 readfloat ENDP
238
239
240 printfloat PROC
241
242         LOCAL   isPositive :BYTE
243
244         MOV     EDI, (OFFSET buf)+(SIZEOF buf)-1 ; points to the end of the buffer
245         XOR     ECX, ECX ; keep a count of how many bytes written
246
247         FMUL    THOUSAND ; Gives us 3 decimal places of precision
248         FLDZ
249         FCOMP
250         FSTSW   AX
251         AND     AH, 1 ; isolate C0 status bit
252         MOV     isPositive, AH
253         FABS            ; Ensure the working value is always positive
254         FADD    HALF    ; the first digit should be rounded to the nearest integer
255         FLD     TEN
256
257         STD ; We will be working backwards from the end of the string
258
259         decodeloop:
260                 FLD     ST(1)
261                 FPREM
262                 FSUB    HALF    ; Hack to round down without fiddling with FPU regs
263                 FISTP   tmp     ; This instruction is likely very slow
264
265                 ; sometimes the FPU does not finish calculating the remainder,
266                 ; for some unknown reason, so in this case we assume the result
```

```
                ; should be 0.
                FSTSW   AX
                AND     AH, 4 ; C2 status bit
                JE noerror
                MOV     tmp, 0
        noerror:

                MOV     EAX, tmp        ; Convert to ascii
                ADD     AL, '0'
                STOSB                   ; Add to buffer

                FDIV    ST(1), ST(0)    ; Divide the working value by ten
                INC     ECX
                CMP     ECX, 3
                JNE     skipdp
                MOV     AL, '.'
                STOSB
                INC     ECX
        skipdp:
                CMP     ECX, 5
                ; This ensures leading zeroes are printed for values less than 1
                JL      decodeloop
                FLD1
                FCOMP   ST(2)
                FSTSW   AX
                AND     AH, 1
                JNE decodeloop ; Loop if working value >= 1?

        FINIT ; Put the FPU back in a clean state

        CMP     isPositive, 0
        JNE     skipneg
        MOV     AL, '-'
        STOSB
        INC     ECX
skipneg:

        CLD ; Without this, the Windows API will break, because it sucks.
        INC     EDI
        WRITE_OUTPUT EDI, ECX
        RET

printfloat ENDP


END main
```

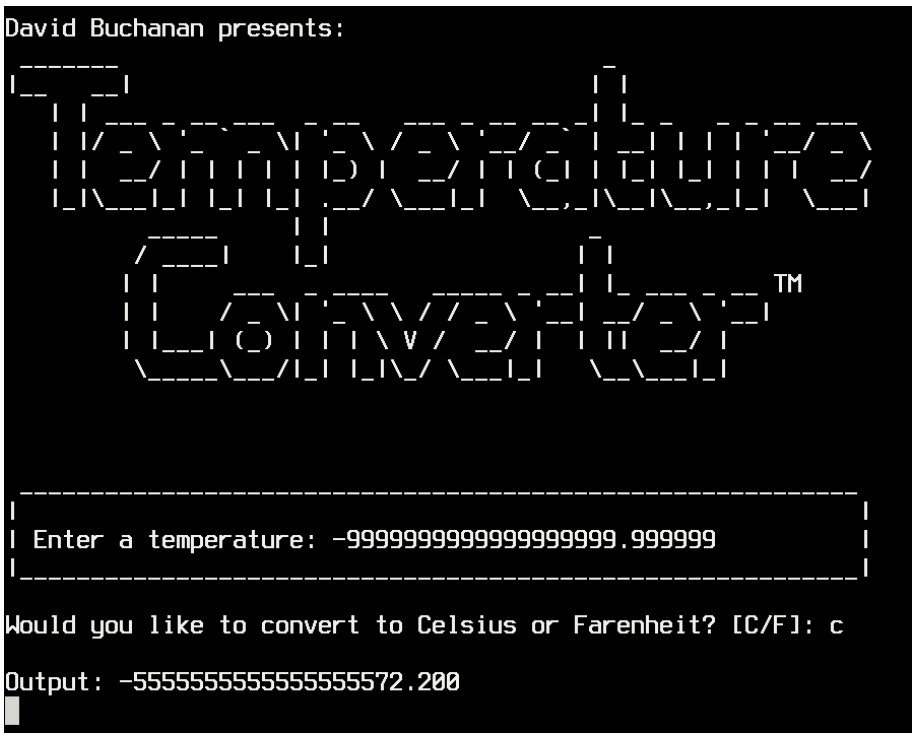
2 Screenshots:

My program can handle negative numbers:



...and decimals:

...and very large numbers:



And is resilient to invalid inputs: