13.3.12 Accessing Command Line Parameters

Most programs like MASM and LINK allow you to specify command line parameters when the program is executed. For example, by typing

```
ML MYPGM.ASM
```

you can instruct MASM to assemble MYPGM without any further intervention from the keyboard. "MYPGM.ASM;" is a good example of a command line parameter.

When DOS' COMMAND.COM command interpreter parses your command line, it copies most of the text following the program name to location 80h in the PSP as described in the previous section. For example, the command line above will store the following at PSP:80h

```
11, " MYPGM.ASM", 0Dh
```

The text stored in the command line tail storage area in the PSP is usually an exact copy of the data appearing on the command line. There are, however, a couple of exceptions. First of all, I/O redirection parameters are not stored in the input buffer. Neither are command tails following the pipe operator ("|"). The other thing appearing on the command line which is absent from the data at PSP:80h is the program name. This is rather unfortunate, since having the program name available would allow you to determine the directory containing the program. Nevertheless, there is lots of useful information present on the command line.

The information on the command line can be used for almost any purpose you see fit. However, most programs expect two types of parameters in the command line parameter buffer-- filenames and switches. The purpose of a filename is rather obvious, it allows a program to access a file without having to prompt the user for the filename. Switches, on the other hand, are arbitrary parameters to the program. By convention, switches are preceded by a slash or hyphen on the command line.

Figuring out what to do with the information on the command line is called parsing the command line. Clearly, if your programs are to manipulate data on the command line, you've got to parse the command line within your code.

Before a command line can be parsed, each item on the command line has to be separated out apart from the others. That is, each word (or more properly, lexeme**[7]**) has to be identified in the command line. Separation of lexemes on a command line is relatively easy, all you've got to do is look for sequences of delimiters on the command line. Delimiters are special symbols used to separate tokens on the command line. DOS supports six different delimiter characters: space, comma, semicolon, equal sign, tab, or carriage return.

Generally, any number of delimiter characters may appear between two tokens on a command line. Therefore, all such occurrences must be skipped when scanning the command line. The following assembly language code scans the entire command line and prints all of the tokens that appear thereon:

```
                include         stdlib.a
                includelib      stdlib.lib

cseg            segment byte public 'CODE'
                assume  cs:cseg, ds:dseg, es:dseg, ss:sseg

; Equates into command line-

CmdLnLen        equ     byte ptr es:[80h] ;Command line length
CmdLn           equ     byte ptr es:[81h] ;Command line data

tab             equ     09h

MainPgm         proc    far

; Properly set up the segment registers:

                push    ds              ;Save PSP
                mov     ax, seg dseg
                mov     ds, ax
                pop     PSP

;-----------------------------------------------------------------

                print
                byte    cr,lf
                byte    'Items on this line:',cr,lf,lf,0

                mov     es, PSP         ;Point ES at PSP
                lea     bx, CmdLn       ;Point at command line
PrintLoop:      print
                byte    cr,lf,'Item: ',0
                call    SkipDelimiters  ;Skip over leading delimiters
PrtLoop2:       mov     al, es:[bx]     ;Get next character
                call    TestDelimiter   ;Is it a delimiter?
                jz      EndOfToken      ;Quit this loop if it is
                putc                    ;Print char if not.
                inc     bx              ;Move on to next character
                jmp     PrtLoop2

EndOfToken:     cmp     al, cr          ;Carriage return?
                jne     PrintLoop       ;Repeat if not end of line

                print
                byte    cr,lf,lf
                byte    'End of command line',cr,lf,lf,0
                ExitPgm
MainPgm         endp
```

```
; The following subroutine sets the zero flag if the character in
; the AL register is one of DOS' six delimiter characters,
; otherwise the zero flag is returned clear. This allows us to use
; the JE/JNE instructions afterwards to test for a delimiter.

TestDelimiter   proc    near
                cmp     al, ' '
                jz      ItsOne
                cmp     al,','
                jz      ItsOne
                cmp     al,Tab
                jz      ItsOne
                cmp     al,';'
                jz      ItsOne
                cmp     al,'='
                jz      ItsOne
                cmp     al, cr
ItsOne:         ret
TestDelimiter   endp

; SkipDelimiters skips over leading delimiters on the command
; line. It does not, however, skip the carriage return at the end
; of a line since this character is used as the terminator in the
; main program.

SkipDelimiters  proc    near
                dec     bx              ;To offset INC BX below
SDLoop:         inc     bx              ;Move on to next character.
                mov     al, es:[bx]     ;Get next character
                cmp     al, 0dh         ;Don't skip if CR.
                jz      QuitSD
                call    TestDelimiter   ;See if it's some other
                jz      SDLoop          ; delimiter and repeat.
QuitSD:         ret
SkipDelimiters  endp

cseg            ends

dseg            segment byte public 'data'

PSP             word    ?               ;Program segment prefix
dseg            ends

sseg            segment byte stack 'stack'
stk             word    0ffh dup (?)
sseg            ends

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       byte    16 dup (?)
zzzzzzseg       ends
                end     MainPgm
```

Once you can scan the command line (that is, separate out the lexemes), the next step is to parse it. For most programs, parsing the command line is an extremely trivial process. If the program accepts only a single filename, all you've got to do is grab the first lexeme on the command line, slap a zero byte onto the end of it (perhaps moving it into your data segment), and use it as a filename. The following assembly language example modifies the hex dump routine presented earlier so that it gets its filename from the command line rather than hard-coding the filename into the program:

```
                include         stdlib.a
                includelib      stdlib.lib

cseg            segment byte public 'CODE'
                assume  cs:cseg, ds:dseg, es:dseg, ss:sseg

; Note CR and LF are already defined in STDLIB.A

tab             equ     09h

MainPgm         proc    far

; Properly set up the segment registers:

                mov     ax, seg dseg
                mov     es, ax          ;Leave DS pointing at PSP

;-----------------------------------------------------------------
;
; First, parse the command line to get the filename:

                mov     si, 81h          ;Pointer to command line
                lea     di, FileName    ;Pointer to FileName buffer
SkipDelimiters:
                lodsb                   ;Get next character
                call    TestDelimiter
                je      SkipDelimiters

; Assume that what follows is an actual filename

                dec     si              ;Point at 1st char of name
GetFName:       lodsb
                cmp     al, 0dh
                je      GotName
                call    TestDelimiter
                je      GotName
                stosb                   ;Save character in file name
                jmp     GetFName
```

```
; We're at the end of the filename, so zero-terminate it as
; required by DOS.

GotName:        mov     byte ptr es:[di], 0
                mov     ax, es          ;Point DS at DSEG
                mov     ds, ax

; Now process the file

                mov     ah, 3dh
                mov     al, 0           ;Open file for reading
                lea     dx, Filename    ;File to open
                int     21h
                jnc     GoodOpen
                print
                byte    'Cannot open file, aborting program...',cr,0
                jmp     PgmExit

GoodOpen:       mov     FileHandle, ax  ;Save file handle
                mov     Position, 0     ;Initialize file position
ReadFileLp:     mov     al, byte ptr Position
                and     al, 0Fh         ;Compute (Position MOD 16)
                jnz     NotNewLn        ;Every 16 bytes start a line
                putcr
                mov     ax, Position    ;Print offset into file
                xchg    al, ah
                puth
                xchg    al, ah
                puth
                print
                byte    ': ',0

NotNewLn:       inc     Position        ;Increment character count
                mov     bx, FileHandle
                mov     cx, 1           ;Read one byte
                lea     dx, buffer      ;Place to store that byte
                mov     ah, 3Fh          ;Read operation
                int     21h
                jc      BadRead
                cmp     ax, 1           ;Reached EOF?
                jnz     AtEOF
                mov     al, Buffer      ;Get the character read and
                puth                    ; print it in hex
                mov     al, ' '         ;Print a space between values
                putc
                jmp     ReadFileLp

BadRead:        print
                byte    cr, lf
                byte    'Error reading data from file, aborting.'
                byte    cr,lf,0

AtEOF:          mov     bx, FileHandle  ;Close the file
                mov     ah, 3Eh
                int     21h

;-----------------------------------------------------------------

PgmExit:        ExitPgm
MainPgm         endp
```

```
TestDelimiter   proc    near
                cmp     al, ' '
                je      xit
                cmp     al, ','
                je      xit
                cmp     al, Tab
                je      xit
                cmp     al, ';'
                je      xit
                cmp     al, '='
xit:            ret
TestDelimiter   endp
cseg            ends

dseg            segment byte public 'data'

PSP             word    ?
Filename        byte    64 dup (0)      ;Filename to dump
FileHandle      word    ?
Buffer          byte    ?
Position        word    0

dseg            ends

sseg            segment byte stack 'stack'
stk             word    0ffh dup (?)
sseg            ends

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       byte    16 dup (?)
zzzzzzseg       ends
                end     MainPgm
```

The following example demonstrates several concepts dealing with command line parameters. This program copies one file to another. If the "/U" switch is supplied (somewhere) on the command line, all of the lower case characters in the file are converted to upper case before being written to the destination file. Another feature of this code is that it will prompt the user for any missing filenames, much like the MASM and LINK programs will prompt you for filename if you haven't supplied any.

```
                include         stdlib.a
                includelib      stdlib.lib

cseg            segment byte public 'CODE'
                assume  cs:cseg, ds:nothing, es:dseg, ss:sseg

; Note: The constants CR (0dh) and LF (0ah) appear within the
; stdlib.a include file.

tab             equ     09h

MainPgm         proc    far

; Properly set up the segment registers:

                mov     ax, seg dseg
                mov     es, ax                  ;Leave DS pointing at PSP

;-----------------------------------------------------------------
```

```
; First, parse the command line to get the filename:

                mov     es:GotName1, 0          ;Init flags that tell us if
                mov     es:GotName2, 0          ; we've parsed the
filenames
                mov     es:ConvertLC,0          ; and the "/U" switch.

; Okay, begin scanning and parsing the command line

                mov     si, 81h                 ;Pointer to command line
SkipDelimiters:
                lodsb                           ;Get next character
                call    TestDelimiter
                je      SkipDelimiters

; Determine if this is a filename or the /U switch

                cmp     al, '/'
                jnz     MustBeFN

; See if it's "/U" here-

                lodsb
                and     al, 5fh                 ;Convert "u" to "U"
                cmp     al, 'U'
                jnz     NotGoodSwitch
                lodsb                           ;Make sure next char is
                cmp     al, cr                  ; a delimiter of some sort
                jz      GoodSwitch
                call    TestDelimiter
                jne     NotGoodSwitch

; Okay, it's "/U" here.

GoodSwitch:     mov     es:ConvertLC, 1         ;Convert LC to UC
                dec     si                      ;Back up in case it's CR
                jmp     SkipDelimiters          ;Move on to next item.

; If a bad switch was found on the command line, print an error
; message and abort-

NotGoodSwitch:
                print
                byte    cr,lf
                byte    'Illegal switch, only "/U" is allowed!',cr,lf
                byte    'Aborting program execution.',cr,lf,0
                jmp     PgmExit

; If it's not a switch, assume that it's a valid filename and
; handle it down here-

MustBeFN:       cmp     al, cr          ;See if at end of cmd line
                je      EndOfCmdLn

; See if it's filename one, two, or if too many filenames have been
; specified-

                cmp     es:GotName1, 0
                jz      Is1stName
                cmp     es:GotName2, 0
                jz      Is2ndName
```

```
; More than two filenames have been entered, print an error message
; and abort.

                print
                byte    cr,lf
                byte    'Too many filenames specified.',cr,lf
                byte    'Program aborting...',cr,lf,lf,0
                jmp     PgmExit

; Jump down here if this is the first filename to be processed-

Is1stName:      lea     di, FileName1
                mov     es:GotName1, 1
                jmp     ProcessName

Is2ndName:      lea     di, FileName2
                mov     es:GotName2, 1
ProcessName:
                stosb                   ;Store away character in name
                lodsb                   ;Get next char from cmd line
                cmp     al, cr
                je      NameIsDone
                call    TestDelimiter
                jne     ProcessName

NameIsDone:     mov     al, 0           ;Zero terminate filename
                stosb
                dec     si              ;Point back at previous char
                jmp     SkipDelimiters  ;Try again.

; When the end of the command line is reached, come down here and
; see if both filenames were specified.

                assume  ds:dseg

EndOfCmdLn:     mov     ax, es          ;Point DS at DSEG
                mov     ds, ax

; We're at the end of the filename, so zero-terminate it as
;  required by DOS.

GotName:        mov     ax, es          ;Point DS at DSEG
                mov     ds, ax

; See if the names were supplied on the command line.
; If not, prompt the user and read them from the keyboard

                cmp     GotName1, 0     ;Was filename #1 supplied?
                jnz     HasName1
                mov     al, '1'         ;Filename #1
                lea     si, Filename1
                call    GetName         ;Get filename #1

HasName1:       cmp     GotName2, 0     ;Was filename #2 supplied?
                jnz     HasName2
                mov     al, '2'         ;If not, read it from kbd.
                lea     si, FileName2
                call    GetName
```

```
; Okay, we've got the filenames, now open the files and copy the
; source file to the destination file.

HasName2        mov     ah, 3dh
                mov     al, 0           ;Open file for reading
                lea     dx, Filename1   ;File to open
                int     21h
                jnc     GoodOpen1

                print
                byte    'Cannot open file, aborting program...',cr,lf,0
                jmp     PgmExit

; If the source file was opened successfully, save the file handle.

GoodOpen1:      mov     FileHandle1, ax ;Save file handle

; Open (CREATE, actually) the second file here.

                mov     ah, 3ch          ;Create file
                mov     cx, 0           ;Standard attributes
                lea     dx, Filename2   ;File to open
                int     21h
                jnc     GoodCreate

; Note: the following error code relies on the fact that DOS
; automatically closes any open source files when the program
; terminates.
                print
                byte    cr,lf
                byte    'Cannot create new file, aborting operation'
                byte    cr,lf,lf,0
                jmp     PgmExit

GoodCreate:     mov     FileHandle2, ax ;Save file handle

; Now process the files

CopyLoop:       mov     ah, 3Fh         ;DOS read opcode
                mov     bx, FileHandle1 ;Read from file #1
                mov     cx, 512         ;Read 512 bytes
                lea     dx, buffer      ;Buffer for storage
                int     21h
                jc      BadRead
                mov     bp, ax          ;Save # of bytes read

                cmp     ConvertLC,0     ;Conversion option active?
                jz      NoConversion

; Convert all LC in buffer to UC-

                mov     cx, 512
                lea     si, Buffer
                mov     di, si
ConvertLC2UC:
                lodsb
                cmp     al, 'a'
                jb      NoConv
                cmp     al, 'z'
                ja      NoConv
                and     al, 5fh
```

```
NoConv:         stosb
                loop    ConvertLC2UC

NoConversion:
                mov     ah, 40h         ;DOS write opcode
                mov     bx, FileHandle2 ;Write to file #2
                mov     cx, bp          ;Write however many bytes
                lea     dx, buffer      ;Buffer for storage
                int     21h
                jc      BadWrite
                cmp     ax, bp          ;Did we write all of the
                jnz     jDiskFull       ; bytes?
                cmp     bp, 512         ;Were there 512 bytes read?
                jz      CopyLoop
                jmp     AtEOF
jDiskFull:      jmp     DiskFull

; Various error messages:

BadRead:        print
                byte    cr,lf
                byte    'Error while reading source file, aborting '
                byte    'operation.',cr,lf,0
                jmp     AtEOF

BadWrite:       print
                byte    cr,lf
                byte    'Error while writing destination file, aborting'
                byte    ' operation.',cr,lf,0
                jmp     AtEOF

DiskFull:               print
                byte    cr,lf
                byte    'Error, disk full.  Aborting operation.',cr,lf,0

AtEOF:          mov     bx, FileHandle1         ;Close the first file
                mov     ah, 3Eh
                int     21h
                mov     bx, FileHandle2         ;Close the second file
                mov     ah, 3Eh
                int     21h

PgmExit:        ExitPgm
MainPgm         endp

TestDelimiter   proc    near
                cmp     al, ' '
                je      xit
                cmp     al, ','
                je      xit
                cmp     al, Tab
                je      xit
                cmp     al, ';'
                je      xit
                cmp     al, '='
xit:            ret
TestDelimiter   endp

; GetName- Reads a filename from the keyboard.  On entry, AL
; contains the filename number and DI points at the buffer in ES
; where the zero-terminated filename must be stored.
```

```
GetName         proc    near
                print
                byte    'Enter filename #',0
                putc
                mov     al, ':'
                putc
                gets
                ret
GetName         endp
cseg            ends

dseg            segment byte public 'data'

PSP             word    ?
Filename1       byte    128 dup (?)     ;Source filename
Filename2       byte    128 dup (?)     ;Destination filename
FileHandle1     word    ?
FileHandle2     word    ?
GotName1        byte    ?
GotName2        byte    ?
ConvertLC       byte    ?
Buffer          byte    512 dup (?)

dseg            ends

sseg            segment byte stack 'stack'
stk             word    0ffh dup (?)
sseg            ends

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       byte    16 dup (?)
zzzzzzseg       ends
                end     MainPgm
```

As you can see, there is more effort expended processing the command line parameters than actually copying the files!

13.3.13 ARGC and ARGV

The UCR Standard Library provides two routines, `argc` and `argv`, which provide easy access to command line parameters. `Argc` (argument count) returns the number of items on the command line. `Argv` (argument vector) returns a pointer to a specific item in the command line.

These routines break up the command line into lexemes using the standard delimiters. As per MS-DOS convention, argc and argv treat any string surrounded by quotation marks on the command line as a single command line item.

`Argc` will return in `cx` the number of command line items. Since MS-DOS does not include the program name on the command line, this count does not include the program name either. Furthermore, redirection operands (">filename" and "<filename") and items to the right of a pipe ("| command") do not appear on the command line either. As such, argc does not count these, either.

`Argv` returns a pointer to a string (allocated on the heap) of a specified command line item. To use `argv` you simply load `ax` with a value between one and the number returned by `argc` and execute the `argv` routine. On return, `es:di` points at a string containing the specified command line option. If the number in `ax` is greater than the number of command line arguments, then `argv` returns a pointer to an empty string (i.e., a zero byte). Since `argv` calls `malloc` to allocate storage on the heap, there is the possibility that a memory allocation error will occur. `Argv` returns the carry set if a memory allocation error occurs. Remember to free the storage allocated to a command line parameter after you are through with it.

Example: The following code echoes the command line parameters to the screen.

```
                include         stdlib.a
                includelib      stdlib.lib

dseg            segment para public 'data'

ArgCnt          word    0

dseg            ends

cseg            segment para public 'code'
                assume  cs:cseg, ds:dseg

Main            proc
                mov     ax, dseg
                mov     ds, ax
                mov     es, ax

; Must call the memory manager initialization routine if you use any routine
; which calls malloc!  ARGV is a good example of a routine which calls malloc.

                meminit

                argc                    ;Get the command line arg count.
                jcxz    Quit            ;Quit if no cmd ln args.
                mov     ArgCnt, 1       ;Init Cmd Ln count.
PrintCmds:      printf                  ;Print the item.
                byte    "\n%2d: ",0
                dword   ArgCnt

                mov     ax, ArgCnt      ;Get the next command line guy.
                argv
                puts
                inc     ArgCnt          ;Move on to next arg.
                loop    PrintCmds       ;Repeat for each arg.
                putcr

Quit:           ExitPgm                 ;DOS macro to quit program.
Main            endp
cseg            ends

sseg            segment para stack 'stack'
stk             byte    1024 dup ("stack   ")
sseg            ends

;zzzzzzseg is required by the standard library routines.

zzzzzzseg       segment para public 'zzzzzz'
LastBytes       byte    16 dup (?)
zzzzzzseg       ends
                end     Main
```

---

[7] Many programmers use the term "token" rather than lexeme. Technically, a token is a different entity.