Parse a command line string into tokens (in C, C++, & Lisp)

Gene Michael Stover

created Monday, 2006 November 6
updated Wednesday, 2007 February 28

Contents

1 What is this?

This is how you obtain the values for your own `char *argv[]` when the run-time system doesn't do it for you. The run-time of unix[1] parse the command line into `argv` for you, but Microsloth Windows does not do so if your program's entry point is `WinMain`. When the run-time does not parse them for you, you can use the functions here to do it yourself. Then you can send the `argv` to `getopt` for standardized option processing.

[1] . . . and of unix-like systems, including Unix, POSIX, OpenBSD, NetBSD, FreeBSD, original BSD, AIX, HP/UX, A/UX, NeXT STEP. Oh, & Gnu/Linux.

I needed a C or C++ function which would parse a command line string into tokens which were suitable for `getopt` or some similar function. Normally, a command line processor such as `/bin/sh` does this work before your program's `main` function starts, but in this case, I was working in a limited environment in which there was a command line of sorts but no command line processor to parse the command line string into tokens. So I had to write one.

While I was at it, I also implemented it in Lisp. Actually, the algorithm in plain C was difficult enough that I implemented it in Lisp first so I could see how to do it. Then I used my Lisp code as a pattern for writing the C & C++ functions.

2 What the functions do

The functions take a single string as an argument & return a list[2] of the tokens in that string. The parsing algorithm resembles that used by the usual command line interpreters such as `/bin/sh` on unix[3] or `cmd` on Windows, but the algorithm is *not identical to those command line interpreters.*

The grammar rules for my parser are:

1. The functions are aware of a quote character & an escape character which can be defined at run-time. The quote character defaults to double-quote (`"`). The escape character defaults to back-slash (\).
2. In general, tokens are separated by one or more white-space characters. Multiple consecutive white-space characters are treated as a single separator.
3. White-space characters inside a pair of quotes are literals, not dividers.
4. A quote character preceeded by an escape character is a literal, not a token-encloser.
5. An escape character which precedes anything other than a quote is a literal, not an escape. In other words, an escape only escapes quotes.

I don't like that syntax, but it was a requirement imposed on me. It'd be simpler if an escape was always an escape, but the values which these functions will parse will themselves be embedded in another string & will need to be escaped or quoted. That, *and* it's a DOS-like enviroment where back-slashes separate subdirectory names in pathnames. So I'm stuck with this syntax.

[2] I'm using the term "list" loosely here. In Lisp, it really is a list. In C & C++, it's an array of pointers to strings, plus an integral length.

[3] . . . on unix or unix-like systems, which includes Leeeenux . . .

3 Examples

Here are some examples. The *input* column is literal, what you'd type on the command line. The *output* column is in Lisp notation, which I chose because C & C++ don't have a good notation for this type of thing. Also, I'm using dots (`.`) to show space characters in the examples which demonstrate how consecutive multiple spaces are parsed.

input	output
`word`	`("word")`
`blue shoes`	`("blue" "shoes")`
`"blue shoes"`	`("blue shoes")`
`"blue...shoes"`	`("blue...shoes")`
`"blue shoes" red rum`	`("blue shoes" red rum)`
`quoted \"quote\"`	`("quoted" "\"quote\"")`
`embedded q\"uot\"es`	`("embedded" "q\"uot\"es")`

4 License

All files, both source & compiled, are copyrighted by Gene Michael Stover & released under the terms of the GNU General Public License [1]. Here's a copy of the copyright notice & license agreement at the beginning of each source file:

> *Copyright (c) 2006 Gene Michael Stover. All rights reserved.*
>
> *This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.*
>
> *This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*
>
> *You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA*

5 The Lisp source code

There's only one Lisp source file. It's online at http://cybertiggyr.com/gene/pcm/src/parse.lisp. It's also in Appendix A.

6 The C source code

A parse.lisp

```
;;; -*- Mode: Lisp -*-
;;;
;;; $Header: /home/gene/library/website/docsrc/pcm/src/RCS/parse.lisp,v 395.1 2008/04/20 17:25:47 gene
;;;
;;; Copyright (c) 2006 Gene Michael Stover.  All rights reserved.
;;;
;;; This program is free software; you can redistribute it and/or modify
;;; it under the terms of the GNU General Public License as
;;; published by the Free Software Foundation; either version 2 of the
;;; License, or (at your option) any later version.
;;;
;;; This program is distributed in the hope that it will be useful,
;;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;;; GNU General Public License for more details.
;;;
;;; You should have received a copy of the GNU General Public
;;; License along with this program; if not, write to the Free Software
;;; Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
;;; USA
;;;

(defvar *quote* #\")
(defvar *escape* #\\)
(defvar *spaces* '(#\Space #\Tab #\Newline))

(defun is-space (x) (member x *spaces*))

(defun discard-space (lst)
  (do ()
      ((not (is-space (first lst))) lst)
      (pop lst)))

(defun lex0 (seq)
  "Perform level-0 lexical analysis on the sequence of characters.
Return list of tokens.  Most characters become themselves as tokens.
Quotes become :QUOTE unless escaped.  Contiguous spaces become
:SPACE.  Escape character is itself unless it was used to escape a
quote."
  (let ((x nil)
        (is-quoted nil))
    (do ((lst (coerce seq 'list) (rest lst)))
        ((endp lst) (nreverse x))
        (cond ((equal *quote* (first lst))
               ;; An un-escaped quote character becomes a :QUOTE token.
```

```
               (setq is-quoted (not is-quoted))
               (push :quote x))
              ((and (equal *escape* (first lst))
                    (equal *quote* (second lst)))
               ;; An escaped quote becomes a quote character.
               (push #\" x);
               (pop lst)) ; discard the escape
              ((and (not is-quoted) (is-space (first lst)))
               ;; The sequence begins with a white-space character.
               ;; So we push :SPACE onto the tokens, then discard all
               ;; of the spaces.
               (push :space x)
               (setq lst (cons :will-be-discarded (discard-space lst))))
              (t
               ;; Most characters are literals.
               (push (first lst) x))))))

(defun next-quoted-token (lst0)
  (assert (eq :quote (first lst0)))
  (pop lst0)                            ; discard :QUOTE
  (do ((token nil (cons (first lst) token))
       (lst lst0 (rest lst)))
      ((eq :quote (first lst)) (coerce (nreverse token) 'string))))

(defun after-quoted-token (lst)
  (assert (eq :quote (first lst)))
  (pop lst)                             ; discard beginning :QUOTE
  (loop until (member (first lst) '(:quote nil))
        do (pop lst))
  (pop lst)                             ; discard ending :QUOTE
  lst)

(defun next-raw-token (lst0)
  (do ((token nil (cons (first lst) token))
       (lst lst0 (rest lst)))
      ((member (first lst) '(:space nil))
       (coerce (nreverse token) 'string))))

(defun after-raw-token (lst)
  (loop until (member (first lst) '(:space nil))
        do (pop lst))
  (pop lst)                             ; discard the :SPACE
  lst)

(defun next-token (lst)
  "Return the next token as a string."
  (setq lst (discard-space lst))
  (cond ((endp lst) nil)
        ((eq :quote (first lst)) (next-quoted-token lst))
        (t (next-raw-token lst))))
```

```
(defun after-token (lst)
  (setq lst (discard-space lst))
  (cond ((endp lst) nil)
        ((eq :quote (first lst)) (after-quoted-token lst))
        (t (after-raw-token lst))))

(defun lex1 (seq0)
  "Given a sequence as returned by LEX0, produce a list of
tokens.  These tokens are strings."
  (do ((lst nil (cons (next-token seq) lst))
       (seq (coerce seq0 'list) (after-token seq)))
      ((endp seq) (nreverse lst))))

(defun parse (string)
  (lex1 (lex0 (coerce string 'list))))

;;;
;;;
;;;

(defun test0000 () 'test0000)

(defun test0002 ()
  "Test IS-SPACE on some hard-coded inputs."
  (and (is-space #\Space)
       (is-space #\Tab)
       (is-space #\Newline)
       (not (is-space #\a))
       (not (is-space nil))
       (not (is-space 32))))

(defun test0005 ()
  "Test LEX0 on a simple hard-coded input."
  (equal (lex0 "abc") '(#\a #\b #\c)))

(defun test0007 ()
  "Test LEX0 on a hard-coded input that contains a single space
between two non-space characters."
  (equal (lex0 "a c") '(#\a :space #\c)))

(defun test0008 ()
  "Test LEX0 on a hard-coded input that contains contiguous
spaces between two non-space characters."
  (equal (lex0 "a      c") '(#\a :space #\c)))

(defun test0009 ()
  "Test LEX0 on a hard-coded input that contains leading &
trailing spaces."
  (equal (lex0 "  a c ") '(:space #\a :space #\c :space)))
```

```
(defun test0013 ()
  "Test LEX0 on a hard-coded input that contains a quoted token.
That token contains an embedded space."
  (equal (lex0 "\"a c\"") '(:quote #\a #\Space #\c :quote)))

(defun test0015 ()
  "Test LEX0 on a hard-coded input that contains some spaced
which are quoted, some which are not, & some escaped quotes."
  (equal (lex0 " \"\\\" \"")
         '(:space :quote #\" #\Space :quote)))

(defun test0051 ()
  "Test LEX1 on a simple input.  This input has one token, no
quotes, no spaces, not escapes."
  (equal (lex1 '(#\w #\o #\r #\d)) '("word")))

(defun test0053 ()
  "Test LEX1 on a two-token input.  The tokens are plain, have no
quotes, escapes, or embedded spaces."
  (equal (lex1 '(#\b #\l #\u #\e :SPACE #\s #\h #\o #\e))
         '("blue" "shoe")))

(defun test0055 ()
  "Test LEX1 on a one-token input.  The token contains a space.
The token is quoted so it should include the space."
  (equal (lex1 (lex0 "\"a c\"")) '("a c")))

(defun test0056 ()
  "Test LEX1 on a one-token input which contains an escaped
quote."
  (equal (lex1 (lex0 "\\\"a")) '("\"a")))

;;; --- end of file ---
```

B Other File Formats

- This document is available in multi-file HTML format at http://cybertiggyr.com/gene/pcm/.
- This document is available in Pointless Document Format (PDF) at http://cybertiggyr.com/gene/pcm/pcm.pdf.

References

[1] Free Software Foundation. General public license. world wide web. http://www.gnu.org/licenses/licenses.html#GPL.