```c
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
char strArgc[256]={0};

/**************************************************************************
 * CommandLineToArgvA            [SHELL32.@]
 *
 * We must interpret the quotes in the command line to rebuild the argv
 * array correctly:
 * - arguments are separated by spaces or tabs
 * - quotes serve as optional argument delimiters
 *   '"a b"'   -> 'a b'
 * - escaped quotes must be converted back to '"'
 *   '\"'      -> '"'
 * - consecutive backslashes preceding a quote see their number halved with
 *   the remainder escaping the quote:
 *   2n   backslashes + quote -> n backslashes + quote as an argument delimiter
 *   2n+1 backslashes + quote -> n backslashes + literal quote
 * - backslashes that are not followed by a quote are copied literally:
 *   'a\b'     -> 'a\b'
 *   'a\\b'    -> 'a\\b'
 * - in quoted strings, consecutive quotes see their number divided by three
 *   with the remainder modulo 3 deciding whether to close the string or not.
 *   Note that the opening quote must be counted in the consecutive quotes,
 *   that's the (1+) below:
 *   (1+) 3n   quotes -> n quotes
 *   (1+) 3n+1 quotes -> n quotes plus closes the quoted string
 *   (1+) 3n+2 quotes -> n+1 quotes plus closes the quoted string
 * - in unquoted strings, the first quote opens the quoted string and the
 *   remaining consecutive quotes follow the above rule.
 */
LPSTR* WINAPI CommandLineToArgvA_wine(LPSTR lpCmdline, int* numargs)
{
  DWORD argc;
  LPSTR  *argv;
```

```
37      LPSTR s;
38      LPSTR d;
39      LPSTR cmdline;
40      int qcount,bcount;
41
42      if(!numargs || *lpCmdline==0)
43        {
44          SetLastError(ERROR_INVALID_PARAMETER);
45          return NULL;
46        }
47
48      /* --- First count the arguments */
49      argc=1;
50      s=lpCmdline;
51      /* The first argument, the executable path, follows special rules */
52      if (*s=='"')
53        {
54          /* The executable path ends at the next quote, no matter what */
55          s++;
56          while (*s)
57            if (*s++=='"')
58              break;
59        }
60      else
61        {
62          /* The executable path ends at the next space, no matter what */
63          while (*s && *s!=' ' && *s!='\t')
64            s++;
65        }
66      /* skip to the first argument, if any */
67      while (*s==' ' || *s=='\t')
68        s++;
69      if (*s)
70        argc++;
71
72      /* Analyze the remaining arguments */
73      qcount=bcount=0;
74      while (*s)
75        {
76          if ((*s==' ' || *s=='\t') && qcount==0)
77            {
78              /* skip to the next argument and count it if any */
79              while (*s==' ' || *s=='\t')
80                s++;
81              if (*s)
82                argc++;
83              bcount=0;
84            }
85          else if (*s=='\\')
86            {
87              /* '\', count them */
88              bcount++;
89              s++;
```

```
      }
    else if (*s=='"')
      {
        /* '"' */
        if ((bcount & 1)==0)
          qcount++; /* unescaped '"' */
        s++;
        bcount=0;
        /* consecutive quotes, see comment in copying code below */
        while (*s=='"')
          {
            qcount++;
            s++;
          }
        qcount=qcount % 3;
        if (qcount==2)
          qcount=0;
      }
    else
      {
        /* a regular character */
        bcount=0;
        s++;
      }
  }

/* Allocate in a single lump, the string array, and the strings that go
 * with it. This way the caller can make a single LocalFree() call to free
 * both, as per MSDN.
 */
argv=LocalAlloc(LMEM_FIXED, (argc+1)*sizeof(LPSTR)+(strlen(lpCmdline)+1)*sizeof(char))
if (!argv)
  return NULL;
cmdline=(LPSTR)(argv+argc+1);
strcpy(cmdline, lpCmdline);

/* --- Then split and copy the arguments */
argv[0]=d=cmdline;
argc=1;
/* The first argument, the executable path, follows special rules */
if (*d=='"')
  {
    /* The executable path ends at the next quote, no matter what */
    s=d+1;
    while (*s)
      {
        if (*s=='"')
          {
            s++;
            break;
          }
        *d++=*s++;
```

```
142             }
143           }
144         else
145           {
146             /* The executable path ends at the next space, no matter what */
147             while (*d && *d!=' ' && *d!='\t')
148               d++;
149             s=d;
150             if (*s)
151               s++;
152           }
153         /* close the executable path */
154         *d++=0;
155         /* skip to the first argument and initialize it if any */
156         while (*s==' ' || *s=='\t')
157           s++;
158         if (!*s)
159           {
160             /* There are no parameters so we are all done */
161             argv[argc]=NULL;
162             *numargs=argc;
163             return argv;
164           }
165
166         /* Split and copy the remaining arguments */
167         argv[argc++]=d;
168         qcount=bcount=0;
169         while (*s)
170           {
171             if ((*s==' ' || *s=='\t') && qcount==0)
172               {
173                 /* close the argument */
174                 *d++=0;
175                 bcount=0;
176
177                 /* skip to the next one and initialize it if any */
178                 do {
179                   s++;
180                 } while (*s==' ' || *s=='\t');
181                 if (*s)
182                   argv[argc++]=d;
183               }
184             else if (*s=='\\')
185               {
186                 *d++=*s++;
187                 bcount++;
188               }
189             else if (*s=='"')
190               {
191                 if ((bcount & 1)==0)
192                   {
193                     /* Preceded by an even number of '\', this is half that
```

```
                 * number of '\', plus a quote which we erase.
                 */
                d-=bcount/2;
                qcount++;
              }
            else
              {
                /* Preceded by an odd number of '\', this is half that
                 * number of '\' followed by a '"'
                 */
                d=d-bcount/2-1;
                *d++='"';
              }
            s++;
            bcount=0;
            /* Now count the number of consecutive quotes. Note that qcount
             * already takes into account the opening quote if any, as well as
             * the quote that lead us here.
             */
            while (*s=='"')
              {
                if (++qcount==3)
                  {
                    *d++='"';
                    qcount=0;
                  }
                s++;
              }
            if (qcount==2)
              qcount=0;
          }
        else
          {
            /* a regular character */
            *d++=*s++;
            bcount=0;
          }
      }
    *d='\0';
    argv[argc]=NULL;
    *numargs=argc;

    return argv;
  }

  /**************************************************************************
   * CommandLineToArgvA
   *
   * https://github.com/ola-ct/actilog/blob/master/actiwin/CommandLineToArgvA.cpp
   */
  PCHAR *CommandLineToArgvA_ola(PCHAR CmdLine, int *_argc) {
    PCHAR *argv;
    PCHAR  argv;
```

```
247       ULONG len;
248       ULONG argc;
249       CHAR a;
250       ULONG i, j;
251
252       BOOLEAN in_QM;
253       BOOLEAN in_TEXT;
254       BOOLEAN in_SPACE;
255
256       len = strlen(CmdLine);
257       i = ((len + 2) / 2) * sizeof(PVOID) + sizeof(PVOID);
258
259       argv = (PCHAR *)LocalAlloc(LMEM_FIXED, i + (len + 2) * sizeof(CHAR));
260
261       _argv = (PCHAR)(((PUCHAR)argv) + i);
262
263       argc = 0;
264       argv[argc] = _argv;
265       in_QM = FALSE;
266       in_TEXT = FALSE;
267       in_SPACE = TRUE;
268       i = 0;
269       j = 0;
270
271       while (a = CmdLine[i]) {
272         if (in_QM) {
273           if (a == '\"') {
274             in_QM = FALSE;
275           } else {
276             _argv[j] = a;
277             j++;
278           }
279         } else {
280           switch (a) {
281           case '\"':
282             in_QM = TRUE;
283             in_TEXT = TRUE;
284             if (in_SPACE) {
285               argv[argc] = _argv + j;
286               argc++;
287             }
288             in_SPACE = FALSE;
289             break;
290           case ' ':
291           case '\t':
292           case '\n':
293           case '\r':
294             if (in_TEXT) {
295               _argv[j] = '\0';
296               j++;
297             }
298             in_TEXT = FALSE;
```

```
299             in_SPACE = TRUE;
300             break;
301           default:
302             in_TEXT = TRUE;
303             if (in_SPACE) {
304               argv[argc] = _argv + j;
305               argc++;
306             }
307             _argv[j] = a;
308             j++;
309             in_SPACE = FALSE;
310             break;
311           }
312         }
313         i++;
314       }
315       _argv[j] = '\0';
316       argv[argc] = NULL;
317
318       (*_argc) = argc;
319       return argv;
320     }
321
322     /**************************************************************************
323      * CommandLineToArgvA TINYCLIB version
324      *
325      * http://www.wheaty.net/
326      */
327     #define _MAX_CMD_LINE_ARGS  128
328     // argv stored in below buffer
329     char * _ppszArgv[_MAX_CMD_LINE_ARGS+1];
330     int CommandLineToArgvA_wheaty(LPSTR pszSysCmdLine) {
331       int argc;
332       int cbCmdLine;
333       PSTR pszCmdLine;
334
335       // Set to no argv elements, in case we have to bail out
336       _ppszArgv[0] = 0;
337
338       // First get a pointer to the system's version of the command line, and
339       // figure out how long it is.
340       cbCmdLine = lstrlen( pszSysCmdLine );
341
342       // Allocate memory to store a copy of the command line.  We'll modify
343       // this copy, rather than the original command line.  Yes, this memory
344       // currently doesn't explicitly get freed, but it goes away when the
345       // process terminates.
346       pszCmdLine = (PSTR)HeapAlloc( GetProcessHeap(), 0, cbCmdLine+1 );
347       if ( !pszCmdLine )
348         return 0;
349
350       // Copy the system version of the command line into our copy
351       lstrcpy( pszCmdLine, pszSysCmdLine );
```

```
352
353       if ( '"' == *pszCmdLine )   // If command line starts with a quote ("),
354         {                           // it's a quoted filename.  Skip to next quote.
355           pszCmdLine++;
356
357           _ppszArgv[0] = pszCmdLine;  // argv[0] == executable name
358
359           while ( *pszCmdLine && (*pszCmdLine != '"') )
360             pszCmdLine++;
361
362           if ( *pszCmdLine )      // Did we see a non-NULL ending?
363             *pszCmdLine++ = 0;  // Null terminate and advance to next char
364           else
365             return 0;           // Oops!  We didn't see the end quote
366         }
367       else    // A regular (non-quoted) filename
368         {
369           _ppszArgv[0] = pszCmdLine;  // argv[0] == executable name
370
371           while ( *pszCmdLine && (' ' != *pszCmdLine) && ('\t' != *pszCmdLine) )
372             pszCmdLine++;
373
374           if ( *pszCmdLine )
375             *pszCmdLine++ = 0;  // Null terminate and advance to next char
376         }
377
378       // Done processing argv[0] (i.e., the executable name).  Now do th
379       // actual arguments
380
381       argc = 1;
382
383       while ( 1 )
384         {
385           // Skip over any whitespace
386           while ( *pszCmdLine && (' ' == *pszCmdLine) || ('\t' == *pszCmdLine) )
387             pszCmdLine++;
388
389           if ( 0 == *pszCmdLine ) // End of command line???
390             return argc;
391
392           if ( '"' == *pszCmdLine )   // Argument starting with a quote???
393             {
394               pszCmdLine++;   // Advance past quote character
395
396               _ppszArgv[ argc++ ] = pszCmdLine;
397               _ppszArgv[ argc ] = 0;
398
399               // Scan to end quote, or NULL terminator
400               while ( *pszCmdLine && (*pszCmdLine != '"') )
401                 pszCmdLine++;
402
403               if ( 0 == *pszCmdLine )
```

```
404               return argc;
405
406             if ( *pszCmdLine )
407               *pszCmdLine++ = 0;  // Null terminate and advance to next char
408           }
409         else                        // Non-quoted argument
410           {
411             _ppszArgv[ argc++ ] = pszCmdLine;
412             _ppszArgv[ argc ] = 0;
413
414             // Skip till whitespace or NULL terminator
415             while ( *pszCmdLine && (' '!=*pszCmdLine) && ('\t'!=*pszCmdLine) )
416               pszCmdLine++;
417
418             if ( 0 == *pszCmdLine )
419               return argc;
420
421             if ( *pszCmdLine )
422               *pszCmdLine++ = 0;  // Null terminate and advance to next char
423           }
424
425         if ( argc >= (_MAX_CMD_LINE_ARGS) )
426           return argc;
427       }
428     return argc;
429   }
430
431   // http://stackoverflow.com/a/13281447/4612829
432   // NO memory alloc, just modified source command line string
433   enum { kMaxArgs = 64 };
```

```
435   int CommandLineToArgvA_simple(LPSTR commandLine) {
436     int argc=0;
437     char *p2 = strtok(commandLine, " ");
438     while (p2 && argc < kMaxArgs - 1) {
439       simpleArgv[argc++] = p2;
440       p2 = strtok(0, " ");
441     }
442     simpleArgv[argc] = 0;
443     return argc;
444   }
445
446   //*************************************************
447   // TEST CODE
448   //*************************************************
449
450   void test_simple(void) {
451     int argc = CommandLineToArgvA_simple(GetCommandLineA());
452     for (int i = 0; i<argc; i++) {
453       sprintf(strArgc, "simple: arg %i / %i", i+1, argc);
454       MessageBox(NULL, simpleArgv[i], strArgc, MB_OK);
455     }
```

```c
  // since modified source command line, no need to free memory
}

void test_ola(void) {
  int argc;
  LPSTR * argv = CommandLineToArgvA_ola(GetCommandLineA(), &argc);
  for (int i = 0; i<argc; i++) {
    sprintf(strArgc, "ola: arg %i / %i", i+1, argc);
    MessageBox(NULL, argv[i], strArgc, MB_OK);
  }
  LocalFree(argv);
}

void test_wine(void) {
  int argc;
  LPSTR * argv = CommandLineToArgvA_wine(GetCommandLineA(), &argc);
  for (int i = 0; i<argc; i++) {
    sprintf(strArgc, "wine: arg %i / %i", i+1, argc);
    MessageBox(NULL, argv[i], strArgc, MB_OK);
  }
  LocalFree(argv);
}

void test_wheaty(void) {
  int argc = CommandLineToArgvA_wheaty(GetCommandLineA());
  for (int i = 0; i<argc; i++) {
    sprintf(strArgc, "wheaty: arg %i / %i", i+1, argc);
    MessageBox(NULL, _ppszArgv[i], strArgc, MB_OK);
  }
  // since use global _ppszArgv, no need free memory
}

/* int main(int argc, char *argv[]) { */
int WINAPI WinMain(HINSTANCE currentInstance, HINSTANCE previousInstance, LPSTR commandl
  MessageBox(NULL, GetCommandLineA(), "Command line is", MB_OK);

  test_wheaty();
  test_ola();
  test_wine();

  // since it's modified, have to be tested lastly
  test_simple();

  return 0;
}
```