

# Parse string into argv/argc

Asked 14 years, 11 months ago Modified 8 months ago Viewed 65k times



Is there a way in C to parse a piece of text and obtain values for argv and argc, as if the text had been passed to an application on the command line?

40



This doesn't have to work on Windows, just Linux - I also don't care about quoting of arguments.



c arguments



Share Improve this question

Follow

edited Feb 5, 2017 at 14:57



Tunaki

136k 46 362 433

asked Nov 10, 2009 at 9:09



codebox

20.2k 10 67 81

For what platform? How command lines get parsed into argc/argv is quite different between Windows and UNIX-based systems, for example. On UNIX, the shell typically transforms the command-line significantly, including doing globbing (file pattern expansion) as well as variable substitution. On Windows the file pattern expansion is not done by the shell (unless you're using something like cygwin or MKS Toolkit, of course). – Laurence Gonsalves Nov 10, 2009 at 9:18

If you don't even need to handle quoted args, I really would suggest coding your own function rather than introducing a 3rd party library just for this task. – Remo.D Nov 10, 2009 at 10:15

- 2 Did you try getopt()? (man 3 getopt). You can see most of UNIX/Linux standard tools sources for examples, HUGE number of them. Even man page (at least Linux one) contains decent example. There is also number of wrappers (you see recommendations here) but getopt() seems to be the only one available for ANY UNIX platform (actually it seems to be part of POSIX standard). – Roman Nikitchenko Nov 10, 2009 at 11:26

If ur still interested and want industrial strength from scratch, in small code package. Search this page for `nargv` By far best solution I have seen here from pure c code. Please Vote this Answer Up! So others may find it. – user735796 Apr 9, 2012 at 10:32

@user735796 I did search for `nargv` and your comment is the only hit. So I googled: [github.com/hypersoft/nargv](https://github.com/hypersoft/nargv) ... Some comments though. This uses C99, so it won't work on the Microsoft C compiler. Also an idea is to have unit tests with a bunch of test cases that verify each type of scenario for the parser to verify it works as expected. – Joakim Mar 1, 2015 at 11:14

14 Answers

Sorted by: Highest score (default)



I'm surprised nobody has provided the simplest answer using standard POSIX functionality:

32

<http://www.opengroup.org/onlinepubs/9699919799/functions/wordexp.html>



Share Improve this answer Follow

answered Oct 3, 2010 at 4:26

R.. GitHub STOP  
HELPING ICE

214k 36 394 731

That may do more than you want. E.g. it does shell word expansions including environment variable substitution, e.g. it substituting `$PATH` with the current path. – [Craig McQueen](#) Apr 3, 2013 at 0:57

I guess it depends on what you mean by parse into argv/argc; certainly that involves some of what the shell does (processing quoting), but variable expansion and other things are more questionable. BTW `wordexp` does have an option to disable command expansion. – [R.. GitHub STOP HELPING ICE](#) Apr 3, 2013 at 1:49

If you mean `WRDE_NOCMD`, that doesn't seem to prevent expansion of `$PATH`, nor expanding `*` to the names of files in the current directory. – [Craig McQueen](#) Apr 3, 2013 at 1:57

I didn't say it prevented variable expansion, just that one other thing you might want to turn off, command expansion, can be turned off. – [R.. GitHub STOP HELPING ICE](#) Apr 3, 2013 at 2:21

- 1 This is exactly what I was looking for and seems to work very well. I needed it to pass a user-defined command to `posix_spawn`, not knowing whether there would be additional arguments. However, a short code example would make this answer so much better. Yeah, even now, more than seven years later. :- ) – [domsson](#) Feb 19, 2018 at 20:43



17



Here's my contribution. Its nice and short, but things to be wary of are:

- The use of `strtok` modifies the original "commandLine" string, replacing the spaces with `\0` end-of-string delimiters
- `argv[]` ends up pointing into "commandLine", so don't modify it until you're finished with `argv[]`.

The code:

```
enum { kMaxArgs = 64 };
int argc = 0;
char *argv[kMaxArgs];

char *p2 = strtok(commandLine, " ");
while (p2 && argc < kMaxArgs-1)
{
    argv[argc++] = p2;
    p2 = strtok(0, " ");
}
argv[argc] = 0;
```

You can now use `argc` and `argv`, or pass them to other functions declared like "foo(int argc, char \*\*argv)".

Share Improve this answer Follow

edited Jun 25, 2015 at 16:05

answered Nov 8, 2012 at 2:09



Jonathan Leffler

750k 145 943 1.3k




sstteevvee

389 2 5

Thanks, that saved some time. To anyone else using this: "char\* p1" (though your compiler would have told you =] ) – [jrr](#) Sep 30, 2013 at 21:02

Does this account for escaped args like "some/long path/to file.txt"? – [Greedo](#) Mar 17, 2022 at 15:19

No, you'd have to look for and handle quotes yourself after. If your code is running on a "real" OS, then I'd recommend seeing what it offers. For example, the glib one as suggested in another solution, which should give you those sorts of features. – [sstteevvee](#) Mar 19, 2022 at 8:58 



If glib solution is overkill for your case you may consider coding one yourself.

16

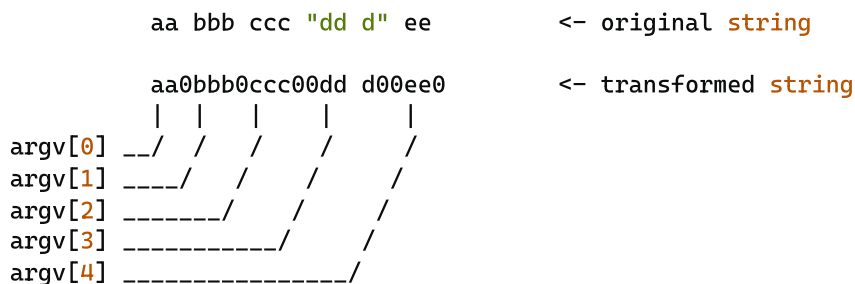
Then you can:



- scan the string and count how many arguments there are (and you get your argc)
- allocate an array of char \* (for your argv)
- rescan the string, assign the pointers in the allocated array and replace spaces with '\0' (if you can't modify the string containing the arguments, you should duplicate it).
- don't forget to free what you have allocated!



The diagram below should clarify (hopefully):



A possible API could be:

```
char **parseargs(char *arguments, int *argc);
void freeparsedargs(char **argv);
```

You will need additional considerations to implement freeparsedargs() safely.

If your string is very long and you don't want to scan twice you may consider alternatives like allocating more elements for the argv arrays (and reallocating if needed).

EDIT: Proposed solution (desn't handle quoted argument).

```
#include <stdio.h>

static int setargs(char *args, char **argv)
{
    int count = 0;

    while (isspace(*args)) ++args;
    while (*args) {
        if (argv) argv[count] = args;
        while (*args && !isspace(*args)) ++args;
        if (argv && *args) *args++ = '\\0';
        while (isspace(*args)) ++args;
    }
}
```

```

        count++;
    }
    return count;
}

char **parsedargs(char *args, int *argc)
{
    char **argv = NULL;
    int    argn = 0;

    if (args && *args
        && (args = strdup(args))
        && (argn = setargs(args, NULL))
        && (argv = malloc((argn+1) * sizeof(char *)))) {
        *argv++ = args;
        argn = setargs(args, argv);
    }

    if (args && !argv) free(args);

    *argc = argn;
    return argv;
}

void freeparsedargs(char **argv)
{
    if (argv) {
        free(argv[-1]);
        free(argv-1);
    }
}

int main(int argc, char *argv[])
{
    int i;
    char **av;
    int ac;
    char *as = NULL;

    if (argc > 1) as = argv[1];

    av = parsedargs(as, &ac);
    printf("== %d\n", ac);
    for (i = 0; i < ac; i++)
        printf("[%s]\n", av[i]);

    freeparsedargs(av);
    exit(0);
}

```

Share Improve this answer Follow

edited Nov 19, 2009 at 8:43

answered Nov 10, 2009 at 9:46




Remo.D

16.5k 6 49 79

- 5 because getopt does a different job. It takes an array of arguments and look for options into it. This question is about splitting a string of "arguments" into an array of char \* which is something that getopt is not able to do – [Remo.D](#) Nov 10, 2009 at 20:28

If you transform input string like that you can't do string concatenation with quotes" like "this" or 'this'. See my answer for a full featured solution. – [user735796](#) Apr 9, 2012 at 18:32

- 2 (*nit-picking ahead*) Note that there is one small thing missing to be compliant with the standard `argc / argv` layout: The entry behind the last valid one in `argv` is always set to `NULL` ( "foo

bar" : argv[0] -> "foo" , argv[1] -> "bar" , argv[2] -> NULL ). – [Max Truxa](#) Oct 20, 2014 at 13:11 



11



The always-wonderful [glib](#) has [g\\_shell\\_parse\\_args\(\)](#) which sounds like what you're after.

If you're not interested in even quoting, this might be overkill. All you need to do is tokenize, using whitespace as a token character. Writing a simple routine to do that shouldn't take long, really.

If you're not super-stingy on memory, doing it in one pass without reallocations should be easy; just assume a worst-case of every second character being a space, thus assuming a string of  $n$  characters contains at most  $(n + 1) / 2$  arguments, and (of course) at most  $n$  bytes of argument text (excluding terminators).

Share Improve this answer Follow

edited Sep 30, 2021 at 15:49

answered Nov 10, 2009 at 9:22



[piec](#)

105 5



[unwind](#)

399k 64 483 616



7



Here's a solution for both Windows and Unix (tested on Linux, OSX and Windows). Tested with [Valgrind](#) and [Dr. Memory](#).

It uses [wordexp](#) for POSIX systems, and [CommandLineToArgvW](#) for Windows.

Note that for the Windows solution, most of the code is converting between `char **` and `wchar_t **` with the beautiful Win32 API, since there is no `CommandLineToArgvA` available (ANSI-version).

```

#ifdef _WIN32
#include <windows.h>
#else
#include <wordexp.h>
#endif

char **split_commandline(const char *cmdline, int *argc)
{
    int i;
    char **argv = NULL;
    assert(argc);

    if (!cmdline)
    {
        return NULL;
    }

    // Posix.
    #ifndef _WIN32
    {
        wordexp_t p;

        // Note! This expands shell variables.
        if (wordexp(cmdline, &p, 0))
        {
            return NULL;
        }
    }

```

```

*argc = p.we_wordc;

if (!(argv = calloc(*argc, sizeof(char *)))
{
    goto fail;
}

for (i = 0; i < p.we_wordc; i++)
{
    if (!(argv[i] = strdup(p.we_wordv[i])))
    {
        goto fail;
    }
}

wordfree(&p);

return argv;
fail:
    wordfree(&p);
}
#else // WIN32
{
    wchar_t **wargs = NULL;
    size_t needed = 0;
    wchar_t *cmdlinew = NULL;
    size_t len = strlen(cmdline) + 1;

    if (!(cmdlinew = calloc(len, sizeof(wchar_t))))
        goto fail;

    if (!MultiByteToWideChar(CP_ACP, 0, cmdline, -1, cmdlinew, len))
        goto fail;

    if (!(wargs = CommandLineToArgvW(cmdlinew, argc)))
        goto fail;

    if (!(argv = calloc(*argc, sizeof(char *)))
        goto fail;

    // Convert from wchar_t * to ANSI char *
    for (i = 0; i < *argc; i++)
    {
        // Get the size needed for the target buffer.
        // CP_ACP = Ansi Codepage.
        needed = WideCharToMultiByte(CP_ACP, 0, wargs[i], -1,
                                     NULL, 0, NULL, NULL);

        if (!(argv[i] = malloc(needed)))
            goto fail;

        // Do the conversion.
        needed = WideCharToMultiByte(CP_ACP, 0, wargs[i], -1,
                                     argv[i], needed, NULL, NULL);
    }

    if (wargs) LocalFree(wargs);
    if (cmdlinew) free(cmdlinew);
    return argv;
}
#endif // WIN32

```

```

if (argv)
{
    for (i = 0; i < *argc; i++)
    {
        if (argv[i])
        {
            free(argv[i]);
        }
    }

    free(argv);
}

return NULL;
}

```

Share Improve this answer Follow

edited Mar 2, 2015 at 9:17

answered Mar 1, 2015 at 3:40



Joakim

11.9k 9 47 50



I just did this for an embedded project in plain C, where I have a little CLI that parses serial port input and executes a limited set of commands with the parameters.

4



This is probably not the neatest, but as small and efficient as I could get it:



```

int makeargs(char *args, int *argc, char ***aa) {
    char *buf = strdup(args);
    int c = 1;
    char *delim;
    char **argv = calloc(c, sizeof (char *));

    argv[0] = buf;

    while (delim = strchr(argv[c - 1], ' ')) {
        argv = realloc(argv, (c + 1) * sizeof (char *));
        argv[c] = delim + 1;
        *delim = 0x00;
        c++;
    }

    *argc = c;
    *aa = argv;

    return c;
}

```

to test:

```

int main(void) {
    char **myargs;
    int argc;

    int numargs = makeargs("Hello world, this is a test", &argc, &myargs);
    while (numargs) {
        printf("%s\r\n", myargs[argc - numargs--]);
    };
}

```

```
    return (EXIT_SUCCESS);
}
```

Share Improve this answer Follow

answered Jun 15, 2014 at 11:42



Andi

1,161

1

7

2

To be a bit closer to the standard `argv`, add an extra position at the end with `NULL`. This is done in case a programmer ignores `argc` and just `while(process(++argv));` until they hit that `NULL`. There would, of course, need to be more to handle quoted arguments (and escaped quotes).

– [Jesse Chisholm](#) Feb 10, 2016 at 13:47

your buf is `strdup` from args. It's memory leaked. – [liuyang1](#) Jun 18, 2020 at 8:45



2



Share Improve this answer Follow

answered Nov 10, 2009 at 9:11



Michael Burr

339k

50

548

769



With the small problem that is C++ and not C :) – [Remo.D](#) Nov 10, 2009 at 9:14

Rename the file to `argcargv.c` and it's C. Literally. – [Michael Burr](#) Nov 10, 2009 at 15:10

- Mr Peitrek's library appears to be very weak when compared to Microsoft's actual rules for separating a command line into `argc/argv` (see [msdn.microsoft.com/en-us/library/17w5ykft.aspx](https://msdn.microsoft.com/en-us/library/17w5ykft.aspx) for their rules.) He doesn't appear to handle embedded quoted strings, multiple backslashes or even escaped quote characters. Not a problem if that's not needed, of course, but folks should be sure they get what they need! – [Steve Valliere](#) May 23, 2013 at 15:07

Also, it's totally unnecessary since Microsoft doesn't just give you the specification how they parse the command line, they also provide an API for this: [CommandLineToArgvW](#) – [Joakim](#) Feb 28, 2015 at 23:20



2



I ended up writing a function to do this myself, I don't think its very good but it works for my purposes - feel free to suggest improvements for anyone else who needs this in the future:

```
void parseCommandLine(char* cmdLineTxt, char*** argv, int* argc){
    int count = 1;

    char *cmdLineCopy = strdupa(cmdLineTxt);
    char* match = strtok(cmdLineCopy, " ");
    // First, count the number of arguments
    while(match != NULL){
        count++;
        match = strtok(NULL, " ");
    }
}
```





```

*argv = malloc(sizeof(char*) * (count+1));
(*argv)[count] = 0;
**argv = strdup("test"); // The program name would normally go in here

if (count > 1){
    int i=1;
    cmdLineCopy = strdupa(cmdLineTxt);
    match = strtok(cmdLineCopy, " ");
    do{
        (*argv)[i++] = strdup(match);
        match = strtok(NULL, " ");
    } while(match != NULL);
}

*argc = count;
}

```

Share Improve this answer Follow

answered Nov 10, 2009 at 12:55



codebox

20.2k 10 67 81

- 1 I like the brevity of your solution but I'm not a big fan of strtok() or strdupa(). I'm also not very clear on what the strdup("test") is for. The major drawback to me seems the fact that you have many strdup and, hence, you will have to do many free() when done. I posted an alternative version in my answer, just in case it may be useful for somebody. – Remo.D Nov 10, 2009 at 16:42

@Remo.D I know it's a long time ago, but I was working on this same general problem myself and about to use strtok. It seems designed for just such a case. So, I'm curious: Why are you "not a big fan of strtok()" – Telemachus Oct 6, 2013 at 20:14

@Telemachus - strtok 1: modifies the buffer it parses, 2: remembers your buffer across calls, which makes it 3: not thread safe as it is not re-entrant. It is not the designed purpose of strtok but the designed in side effects that are annoying. :) :) – Jesse Chisholm Feb 10, 2016 at 13:53



## Consider yet another implementation. [Run.](#)

1

```

#include <cctype> // <ctype.h> for isspace()

/**
 * Parse out the next non-space word from a string.
 * @note No nullptr protection
 * @param str [IN] Pointer to pointer to the string. Nested pointer to string
 * will be changed.
 * @param word [OUT] Pointer to pointer of next word. To be filled.
 * @return pointer to string - current cursor. Check it for '\0' to stop calling
 * this function
 */
static char* splitArgv(char **str, char **word)
{
    constexpr char QUOTE = '\\';
    bool inquotes = false;

    // optimization
    if( **str == 0 )
        return NULL;

    // Skip leading spaces.
    while (**str && isspace(**str))
        (*str)++;

```

```

    if( **str == '\0')
        return NULL;

    // Phrase in quotes is one arg
    if( **str == QUOTE ){
        (*str)++;
        inquotes = true;
    }

    // Set phrase begining
    *word = *str;

    // Skip all chars if in quotes
    if( inquotes ){
        while( **str && **str!=QUOTE )
            (*str)++;
        //if( **str!= QUOTE )
    }else{
        // Skip non-space characters.
        while( **str && !isspace(**str) )
            (*str)++;
    }
    // Null terminate the phrase and set `str` pointer to next symbol
    if(**str)
        *(*str)++ = '\0';

    return *str;
}

/// To support standart convetion last `argv[argc]` will be set to `NULL`
///\param[IN] str : Input string. Will be changed - splitted to substrings
///\param[IN] argc_MAX : Maximum a rgc, in other words size of input array \p
argv
///\param[OUT] argc : Number of arguments to be filled
///\param[OUT] argv : Array of c-string pointers to be filled. All of these
strings are substrings of \p str
///\return Pointer to the rest of string. Check if for '\0' and know if there is
still something to parse. \
///      If result !='\0' then \p argc_MAX is too small to parse all.
char* parseStrToArgcArgvInsitu( char *str, const int argc_MAX, int *argc, char*
argv[] )
{
    *argc = 0;
    while( *argc<argc_MAX-1 && splitArgv(&str, &argv[*argc]) ){
        ++(*argc);
        if( *str == '\0' )
            break;
    }
    argv[*argc] = nullptr;
    return str;
};

```

## Usage code

```

#include <iostream>
using namespace std;

void parseAndPrintOneString(char *input)
{
    constexpr size_t argc_MAX = 5;
    char* v[argc_MAX] = {0};
    int c=0;

```

```

char* rest = parseStrToArgcArgvInsitu(input,argc_MAX,&c,v);
if( *rest!='\0' ) // or more clear `strlen(rest)==0` but not efficient
    cout<<"There is still something to parse. argc_MAX is too small."<<endl;

cout << "argc : "<< c << endl;
for( int i=0; i<c; i++ )
    cout<<"argv["<<i<<" ] : "<<v[i] <<endl;
/*//or condition is `v[i]`
for( int i=0; v[i]; i++ )
    cout<<"argv["<<i<<" ] : "<<v[i] <<endl;*/
}

int main(int argc, char* argv[])
{
    char inputs[][500] ={
        "Just another TEST\r\n"
        , " Hello my world 'in quotes' \t !"
        , "./hi 'Less is more'"
        , "Very long line with \"double quotes\" should be parsed several
times if argv[] buffer is small"
        , " \t\f \r\n"
    };

    for( int i=0; i<5; ++i ){
        cout<<"Parsing line \""<<inputs[i]<<"\" : "<<endl;
        parseAndPrintOneString(inputs[i]);
        cout<<endl;
    }
}

```

## Output:

Parsing line "Just another TEST\r\n":

```

argc : 3
argv[0] : Just
argv[1] : another
argv[2] : TEST

```

Parsing line " Hello my world 'in quotes' !":

There is still something to parse. argc\_MAX is too small.

```

argc : 4
argv[0] : Hello
argv[1] : my
argv[2] : world
argv[3] : in quotes

```

Parsing line "./hi 'Less is more'":

```

argc : 2
argv[0] : ./hi
argv[1] : Less is more

```

Parsing line "Very long line with "double quotes" should be parsed several times if argv[] buffer is small":

There is still something to parse. argc\_MAX is too small.

```

argc : 4
argv[0] : Very
argv[1] : long
argv[2] : line
argv[3] : with

```

Parsing line "

```
":  
argc : 0
```

Share Improve this answer Follow

edited Jun 2, 2017 at 12:24

answered Jun 2, 2017 at 10:39



kyb

7,981

10

61

115

**Solution for those that don't want to use dynamic memory allocation (E.g. embedded)**

1

I written `tokenise_to_argc_argv()` for an embedded project, which uses `strtok_r()` as the basis for tokenising a command string into `argc` and `argv` form. Unlike most answers here, I usually allocate memory statically. Thus my implementation assumes that you have an upper bound of `argv_length`. For most typical embedded applications, this is more than enough. I included example code below as well so you can quickly use it.

```
int tokenise_to_argc_argv(  
    char    *buffer,    ///< In/Out : Modifiable String Buffer To Tokenise  
    int     *argc,      ///< Out   : Argument Count  
    char    *argv[],    ///< Out   : ArgumentString Vector Array  
    const int argv_length ///< In     : Maximum Count For `*argv[]`  
)  
{ /* Tokenise string buffer into argc and argv format (req: string.h) */  
    int i = 0;  
    for (i = 0 ; i < argv_length ; i++)  
    { /* Fill argv via strtok_r() */  
        if ( NULL == (argv[i] = strtok_r( NULL , " ", &buffer)) ) break;  
    }  
    *argc = i;  
    return i; // Argument Count  
}
```

**Note:**

- The provided character buffer must be modifiable (as `strtok_r()` inserts `\0` into the buffer to delimitate string tokens).
- `strtok_r` in this function is currently using `" "` space character as the only delimitator. This emulates the behaviour `main(int argc, char *argv[])` in typical commandline interfaces.
- This function does not use `malloc` or `calloc`, instead you will have to allocate the `argv` array separately, and supply the length of `argv` explicitly. This is because I intend to use this in embedded devices and thus would rather allocate it manually.
- `strtok_r()` is used because it is threadsafe (Since `strtok()` uses an internal static pointer). Also it is part of the standard C library `string.h` thus is very portable.

Below are the demonstration code as well as it's output. In addition, this shows that `tokenise_to_argc_argv()` can handle most string cases and thus has been tested. Also this function does not rely on `malloc` or `calloc` and thus is suitable for embedded usage (after using `stdint.h` types).

## Demonstration Code

```

/*****
Tokenise String Buffer To Argc and Argv Style Format
Brian Khuu 2017
*****/
#include <stdio.h> // printf()
#include <ctype.h> // isprint()
#include <string.h> // strtok_r()

/**-----
@brief Tokenise a string buffer into argc and argv format

Tokenise string buffer to argc and argv form via strtok_r()
Warning: Using strtok_r will modify the string buffer

Returns: Number of tokens extracted

-----*/
int tokenise_to_argc_argv(
    char    *buffer,    ///< In/Out : Modifiable String Buffer To Tokenise
    int     *argc,      ///< Out   : Argument Count
    char    *argv[],    ///< Out   : Argument String Vector Array
    const int argv_length ///< In     : Maximum Count For `*argv[]`
)
{ /* Tokenise string buffer into argc and argv format (req: string.h) */
    int i = 0;
    for (i = 0 ; i < argv_length ; i++)
    { /* Fill argv via strtok_r() */
        if ( NULL == (argv[i] = strtok_r( NULL, " ", &buffer)) ) break;
    }
    *argc = i;
    return i; // Argument Count
}

/*****
Demonstration of tokenise_to_argc_argv()
*****/

static void print_buffer(char *buffer, int size);
static void print_argc_argv(int argc, char *argv[]);
static void demonstrate_tokenise_to_argc_argv(char buffer[], int buffer_size);

int main(void)
{ /* This shows various string examples */
    printf("# `tokenise_to_argc_argv()` Examples\n");
    { printf("## Case0: Normal\n");
      char buffer[] = "tokenising example";
      demonstrate_tokenise_to_argc_argv(buffer, sizeof(buffer));
    }
    { printf("## Case1: Empty String\n");
      char buffer[] = "";
      demonstrate_tokenise_to_argc_argv(buffer, sizeof(buffer));
    }
    { printf("## Case2: Extra Space\n");
      char buffer[] = "extra space here";
      demonstrate_tokenise_to_argc_argv(buffer, sizeof(buffer));
    }
    { printf("## Case3: One Word String\n");
      char buffer[] = "one-word";
      demonstrate_tokenise_to_argc_argv(buffer, sizeof(buffer));
    }
}

```

```

static void demonstrate_tokenise_to_argc_argv(char buffer[], int buffer_size)
{ /* This demonstrates usage of tokenise_to_argc_argv */
    int  argc    = 0;
    char *argv[10] = {0};

    printf("* **Initial State**\n");
    print_buffer(buffer, buffer_size);

    /* Tokenise Command Buffer */
    tokenise_to_argc_argv(buffer, &argc, argv, sizeof(argv));

    printf("* **After Tokenizing**\n");
    print_buffer(buffer, buffer_size);
    print_argc_argv(argc, argv);
    printf("\n\n");
}

static void print_buffer(char *buffer, int size)
{
    printf(" - Buffer Content `");
    for (int i = 0 ; i < size; i++) printf("%c", isprint(buffer[i])?buffer[i]:'0');
    printf("` | HEX: ");
    for (int i = 0 ; i < size; i++) printf("%02X ", buffer[i]);
    printf("\n");
}

static void print_argc_argv(int argc, char *argv[])
{ /* This displays the content of argc and argv */
    printf("* **Argv content** (argc = %d): %s\n", argc, argc ? "" : "Argv Is Empty");
    for (int i = 0 ; i < argc ; i++) printf(" - `argv[%d]` = `%s`\n", i, argv[i]);
}

```

## Output

## tokenise\_to\_argc\_argv() Examples

### Case0: Normal

- **Initial State**
  - Buffer Content tokenising example0 | HEX: 74 6F 6B 65 6E 69 73 69 6E 67 20 65 78 61 6D 70 6C 65 00
- **After Tokenizing**
  - Buffer Content tokenising0example0 | HEX: 74 6F 6B 65 6E 69 73 69 6E 67 00 65 78 61 6D 70 6C 65 00
- **Argv content (argc = 2):**
  - argv[0] = tokenising
  - argv[1] = example

### Case1: Empty String

- **Initial State**
  - Buffer Content `0` | HEX: 00
- **After Tokenizing**
  - Buffer Content `0` | HEX: 00
- **Argv content** (argc = 0): Argv Is Empty

## Case2: Extra Space

- **Initial State**
  - Buffer Content `extra space here0` | HEX: 65 78 74 72 61 20 20 73 70 61 63 65 20 68 65 72 65 00
- **After Tokenizing**
  - Buffer Content `extra0 space0here0` | HEX: 65 78 74 72 61 00 20 73 70 61 63 65 00 68 65 72 65 00
- **Argv content** (argc = 3):
  - `argv[0] = extra`
  - `argv[1] = space`
  - `argv[2] = here`

## Case3: One Word String

- **Initial State**
  - Buffer Content `one-word0` | HEX: 6F 6E 65 2D 77 6F 72 64 00
- **After Tokenizing**
  - Buffer Content `one-word0` | HEX: 6F 6E 65 2D 77 6F 72 64 00
- **Argv content** (argc = 1):
  - `argv[0] = one-word`

### Missing string.h or strtok\_r() in your toolchain somehow?

If for some reason your toolchain does not have `strtok_r()`. You can use this simplified version of `strtok_r()`. It is a modified version of the GNU C implementation of `strtok_r()`, but simplified to only support space character.

To use this, just place it on top of `tokenise_to_argv_argv()` then replace `strtok_r( NULL, " ", &buffer)` with `strtok_space(&buffer)`

```
/**-----
@brief Simplified space delimited only version of strtok_r()
```

- save\_ptr : In/Out pointer to a string. This pointer is incremented by this function to find and mark the token boundry via a '\0' marker. It is also used by this function to find mutiple other tokens via repeated calls.

Returns:

- NULL : No token found
- pointer to start of a discovered token

```
-----*/
char * strtok_space(char **save_ptr)
{ /* strtok_space is slightly modified from GNU C Library `strtok_r()`
   implementation.
   Thus this function is also licenced as GNU Lesser General Public License*/
  char *start = *save_ptr;
  char *end = 0;

  if (*start == '\0') {
    *save_ptr = start;
    return NULL;
  }

  /* Scan leading delimiters. */
  while(*start == ' ') start++;
  if (*start == '\0') {
    *save_ptr = start;
    return NULL;
  }

  /* Find the end of the token. */
  end = start;
  while((*end != '\0') && (*end != ' ')) end++;
  if (*end == '\0') {
    *save_ptr = end;
    return start;
  }

  /* Terminate the token and make *SAVE_PTR point past it. */
  *end = '\0';
  *save_ptr = end + 1;
  return start;
}
```

Share Improve this answer Follow

edited Jul 28, 2017 at 16:46

answered Jul 26, 2017 at 13:24



Brian

576 8 9

nice but it does not handle quotes nor double quotes – Zibri Feb 10, 2019 at 13:32



This one I wrote also considers quotes (but not nested)

1

Feel free to contribute.



```
/*
Tokenize string considering also quotes.
By Zibri <zibri AT zibri DOT org>
https://github.com/Zibri/tokenize
*/
```





```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

int main(int argc, char *argv[])
{
    char *str1, *token;
    int j;
    char *qstart = NULL;
    bool quoted = false;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s string\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    for (j = 1, str1 = argv[1];; j++, str1 = NULL) {
        token = strtok(str1, " ");
        if (token == NULL)
            break;
        if ((token[0] == 0x27) || (token[0] == 0x22)) {
            qstart = token + 1;
            quoted = true;
        }
        if ((token[strlen(token) - 1] == 0x27) || (token[strlen(token) - 1] == 0x22))
        {
            quoted = false;
            token[strlen(token) - 1] = 0;
            printf("%d: %s\n", j, qstart);
        } else {
            if (quoted) {
                token[strlen(token)] = 0x20;
                j--;
            } else
                printf("%d: %s\n", j, token);
        }
    }

    if (quoted) {
        fprintf(stderr, "String quoting error\n");
        return EXIT_FAILURE;
    } else
        return EXIT_SUCCESS;
}

```

Example output:

```

$ ./tokenize "1 2 3 '4 5 6' 7 8 \"test abc\" 10 11"
1: 1
2: 2
3: 3
4: 4 5 6
5: 7
6: 8
7: test abc
8: 10
9: 11

```

Share Improve this answer Follow

answered Feb 10, 2019 at 14:47



Zibri

9,685

3

57

47

Note: it destroys the source string (because it uses strtok) keep it in mind and if needed add a strdup  
– Zibri Feb 10, 2019 at 15:00

I ran your code, but it fails if I input "1 2 3 \'3 4\"567\' \' \"bo1\'oba1a\" 2x2=\"foo\""  
– Lê Quang Duy Apr 7, 2020 at 2:13

@LêQuangDuy feel free to modify it to suit your needs and post here your better solution ;) – Zibri Jul 12, 2020 at 12:42



1



My project requires breaking a string into `argc` and `argv`.

Found a pretty excellent code of [Torek](#). But it alters the input buffer so I made some modifications to fit my needs.

I just put a little bit more to handle quote mixing when input in the command line so the behavior is more (not completely) like Linux Shell.

**Note:** This function doesn't edit the original string, so you can reuse the input buffer (error report,etc).

```
void remove_quote(char* input){
    //Implementing yourself to remove quotes so it would be completely like Linux
    shell
}
size_t cmd_param_split(char *buffer, char *argv[], size_t argv_max_size)
{
    char *p, *start_of_word;
    int c, i;
    enum states { DULL=0, IN_WORD, IN_STRING, QUOTE_DOUBLE, QUOTE_SINGLE } state =
    DULL;
    size_t argc = 0;
    int quote = 0;
    for (p = buffer; argc < argv_max_size && *p != '\0'; p++) {
        c = (unsigned char) *p;
        printf("processing %c, state = %d\n", c, state);
        switch (state) {
            case DULL:
                if (isspace(c)) {
                    continue;
                }

                if (c == '"' || c == '\\') {
                    quote = c;
                    state = IN_STRING;
                    start_of_word = p + 1;
                    continue;
                }
                state = IN_WORD;
                start_of_word = p;
                continue;

            case IN_STRING:
                if (c == '"' || c == '\\') {
                    if (c != quote)
```

```

        continue;
    else
        quote = 0;
        strncpy(argv[argc],start_of_word, p - start_of_word);
        remove_quote(argv[argc]);
        argc++;
        state = DULL;
    }
    continue;

case IN_WORD:
    if(quote==0 && (c == '\'' || c == '\"'))
        quote = c;
    else if (quote == c)
        quote = 0;

    if (isspace(c) && quote==0) {
        strncpy(argv[argc],start_of_word, p - start_of_word);
        remove_quote(argv[argc]);
        argc++;
        state = DULL;
    }
    continue;
}

}

if (state != DULL && argc < argv_max_size){
    strncpy(argv[argc],start_of_word, p - start_of_word);
    remove_quote(argv[argc]);
    argc++;
}

if (quote){
    printf("WARNING: Quote is unbalanced. This could lead to unwanted-
behavior\n");
    for(i = 0;i<argc;i++){
        printf("arg %d = [%s]\n",i,argv[i]);
        printf("Original buffer: [%s]\n",buffer);
    }
    return argc;
}

int main()
{
    int i=0;
    int argc;
    char* argv[64];
    for(i=0;i<64;i++){
        argv[i] = malloc(256);
        memset(argv[i],0x0,256);
    }
    char* buffer="1 2 3 \'3 4\"567\' \'bol\'obala\" 2x2=\"foo\"";
    argc = cmd_param_split(buffer,argv,64);
    for(i = 0;i<argc;i++)
        printf("arg %d = [%s]\n",i,argv[i]);

    return 0;
}

```

Tested with below strings

```

1. "1 2 3 \'3 4\"567\' \'bol\'obala\" 2x2=\"foo\""
arg 0 = [1]
arg 1 = [2]

```

```

arg 2 = [3]
arg 3 = [3 4"567]
arg 4 = [bo1'obala]
arg 5 = [2x2="foo"]
2. "./foo bar=\"Hanoi HoChiMinh\" exp='foo123 \"boo111' mixquote \"hanoi \'s\""
arg 0 = [./foo]
arg 1 = [bar="Hanoi HoChiMinh"]
arg 2 = [exp='foo123 "boo111']
arg 3 = [mixquote]
arg 4 = [hanoi 's]

```

However, Linux shell would remove quotes, even in mixed case, as below when running from cmd line, tested in a RaspberryPi.

```

./foo bar="Hanoi HoChiMinh" exp='foo123 "boo111' mixquote "hanoi 's"
arg 0 = [./foo]
arg 1 = [bar="Hanoi HoChiMinh"]
arg 2 = [exp=foo123 "boo111]
arg 3 = [mixquote]
arg 4 = [hanoi 's]

```

So if you really want to mimic the whole Linux shell's behavior, just put a little bit more effort into removing quotes `remove_quote()` function as I leave blank above.

Share Improve this answer Follow

edited Apr 7, 2020 at 3:03

answered Apr 7, 2020 at 2:52



Lê Quang Duy

787 8 14



Unfortunately C++ but for others which might search for this kind of library i recommend:

0



Really small and really easy.



```

p.addParam("long-name", 'n', ParamContainer::regular,
           "parameter description", "default_value");

```

```

programname --long-name=value

```

```

cout << p["long-name"];
>> value

```

From my experience:

- very useful and simple
- stable on production
- well tested (by me)



bua

4,831

1

27

33

- 1 You're right, I've post it because when I was looking at sources some time ago, I remember it was generic, OOD free code, it looked almost like C. But I think its worth to keep this here. – bua Nov 10, 2009 at 9:37



0



I had to this in C++, I eventually did a C version as well.

This version does not perform dynamic memory allocation, is double-quotes aware, is escaped-double-quotes aware, is single-quotes aware (single quotes don't perform any escaping).

It is the caller responsibility to provide a zero-initialised temporary buffer (the `buf` argument) *at least* as big as the parsed arguments-string (no checks are performed on the buffer size in the parser).

What I noticed when playing around with command line arguments:

- quotes (either `"` or `'`) do *not* define a new argument, only whitespaces do. For example `a"b c"` evaluates to a single argument `ab c`.
- single quotes do *not* perform any escaping on their contents. For example `'hello\'` evaluates to the argument `hello\`.

If there is a spec for how C/C++ programs parse their command-line arguments, I'd be glad to hear about it.

Live gdb: [https://onlinegdb.com/i\\_6\\_G-u5E](https://onlinegdb.com/i_6_G-u5E)

Implementation:

```
#include <string.h>
#include <stdbool.h>

/*
    in args      the string to parse
    in buf       a zero-initialized buffer with size "strlen(args) + 1" or
greater
    out argc     arguments count
    out argv     a char* array buffer
    argvlen     argv array size
*/
void str_to_argc_argv(const char* args, char* buf, int* argc, char** argv, int
argvlen) {
    enum PARSE_STATE {
        GET_CHAR,
        SEEK_DOUBLE_QUOTE_END,
        SEEK_SINGLE_QUOTE_END,

        } parser_state = GET_CHAR;

    size_t commit = 0; //character commit (from "args" into "buf") count
```

```

for (size_t c = 0; c < strlen(args); c++) {
    switch (parser_state) {
    case GET_CHAR: {
        switch (args[c]) {
        case ' ':
        case '\\t': {
            if (commit != 0 && buf[commit - 1] != '\\0') { //ignore leading
whitespaces
                buf[commit++] = '\\0';
            }
            break;
        }
        case '\"': {
            parser_state = SEEK_DOUBLE_QUOTE_END;
            break;
        }
        case '\\\'': {
            parser_state = SEEK_SINGLE_QUOTE_END;
            break;
        }
        default: {
            buf[commit++] = args[c];
            break;
        }
    }
    }
    case SEEK_DOUBLE_QUOTE_END: {
        switch (args[c]) {
        case '\"': {
            bool escaped = args[c - 1] == '\\\' && args[c - 2] != '\\\''; //safe
(c can't be less than 2 here)
            if (!escaped) {
                parser_state = GET_CHAR;
            }
            else {
                buf[commit++] = '\"';
            }
            break;
        }
        default: {
            buf[commit++] = args[c];
            break;
        }
    }
    }
    case SEEK_SINGLE_QUOTE_END: {
        switch (args[c]) {
        case '\\\'': {
            parser_state = GET_CHAR;
            break;
        }
        default: {
            buf[commit++] = args[c];
            break;
        }
    }
    }
    }
}

int argc_ = 0;
while (*buf != '\\0' && argc_ < argvlen) {
    *(argv + argc_++) = buf;
}

```

```

        buf += strlen(buf) + 1;
    }
    *argc = argc_;
}

```

Test:

```

#include <stdio.h>
#include "str_to_argc_argv.h"

#define MAX_ARGS 128
#define BUFFER_LENGTH 128

int main(int argc, char** argv) //command-line argument: hello "world" -opt "an
op\"tion\" 'bla \' bla' "blalba"
{
    for (size_t a = 0; a < argc; a++) {
        puts(argv[a]);
    }
    puts("\n");

    const char* args = "    hello \"world\" -opt \"an op\\\\\"tion\" 'bla \\\' bla'
\\\"blalba\\\"";
    char buf[BUFFER_LENGTH] = { 0 }; //must be at least the size of ^ and 0-
initialized.

    int argc_;           //return value
    char* argv_[MAX_ARGS]; //return value

    str_to_argc_argv(args, buf, &argc_, argv_, MAX_ARGS);

    for (size_t a = 0; a < argc_; a++) {
        puts(argv_[a]);
    }

    return 0;
}

```

Edit:

The escaped test should use this function instead:

```

bool escaped(const char* args, size_t pos) {
    bool escaped_ = false;
    for (size_t c = pos - 1; c != 0; c--) {
        if (args[c] == '\\') {
            escaped_ = !escaped_;
        }
        else {
            return escaped_;
        }
    }
    return escaped_;
}

```

Share Improve this answer Follow edited Feb 28 at 15:29

answered Feb 28 at 13:11



PinkTurtle

7,042 3 29 48

