

Re: Command Line Interface (CLI): your recommendations

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-02/msg01715.html

- *From:* Barry Schwarz <schwarzb@xxxxxxxxxx>
 - *Date:* Sun, 12 Feb 2006 11:08:48 -0800
-

On Sun, 12 Feb 2006 01:14:51 +0700, "Roman Mashak" <mrv@xxxxxxxxxx> wrote:

Hello, All!

I'm implementing simple CLI (flat model, no tree-style menu etc.). Command line looks like this: <command> <param1> <param2> ... <paramN> (where N=1..4)

And idea is pretty simple:

- 1) get whole string of input line
- 2) preset table of strings matching <command>
- 3) preset table of function calls
- 4) scan <command> table for string, if match – call matching function (I declare ALL callback functions as having the same format)

Here is code implementing part of this algorithm. The problem I came across is if pressing 'Enter' continuously then sometimes output of PROMPT turns out clumsy, i.e. it's printed repeatedly in a line.

How could you execute the code with all the syntax errors?

Before I went too far, I would like to ask your opinion about concept I'm using and recommendations if possible:

You really should compile your code and correct the errors before asking for help.

```
#define _GNU_SOURCE

#include <stdio.h>
```

Re: Command Line Interface (CLI): your recommendations

```
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define COUNT(a) (sizeof(a) / sizeof(a[0]))

const char *PROMPT = "CLI>";

enum errCodes { RC_SUCCESS = 0, RC_ERROR = -1 };

/* table of <commands> */
```

If this comment were not here, would your code be any less clear? If the comment does not add something to what is immediately obvious in the code, it actually reduces readability.

```
const char *commands[] = {
    "show",
    "version",
    "help",
    "port",
    "exit"
};

/* callback function */
typedef int (*handler_ptr_t)(int, char**);

/* define command line: <command> <param1> <param2> ... <paramN> */
typedef struct cmdLineEntry_s {
#define MAX_NAME 20
```

Move these defines to either before the typedef or to the start of your source (your choice before or after the includes). Never disrupt the flow of a struct declaration like this.

```
char command[MAX_NAME];
#define MAX_ARGS 4
char params[MAX_ARGS];
```

What you have here is an array of four char. Not enough space to hold even one argument let alone four. Based on how you use it later, I am pretty certain that you wanted
char *params[...]

Re: Command Line Interface (CLI): your recommendations

```
unsigned int params_num;
handler_ptr_t cmd_handler;
} cmdLineEntry_t;

int cliShow(int argc, char *argv[])
{
    puts("cliShow() stub");
    return 0;
}

int cliVersion(int argc, char *argv[])
{
    puts("cliVersion() stub");
    return 0;
}

int cliHelp(int argc, char *argv[])
{
    puts("cliHelp() stub");
    return 0;
}

int cliPort(int argc, char *argv[])
{
    puts("cliPort() stub");
    return 0;
}

int cliExit(int argc, char *argv[])
{
    puts("cliExit() stub");
    exit(EXIT_SUCCESS);
}

/* define table functions pointers */
handler_ptr_t ftable[] = { cliShow, cliVersion, cliHelp, cliPort, cliExit };
```

You intend for this to be a global table. You have defines, typedefs, enum declarations, global definitions, and function definitions interspersed with each other. When you fill in the stub functions, you will have a hellacious time finding anything. While it is "only" a matter of style, it will make debugging much easier if you organize things better. My recommendation:

Put all your defines, typedefs, and declarations in a header file. Include prototypes for all your functions. Group like directives (all defines, then all typedefs, etc). I even include my `#include<...>` directives in this header but others prefer to leave those in their function source files.

Re: Command Line Interface (CLI): your recommendations

Put each function in its own source file. If you really want to keep them all together (such as for posting here), the order them in some way that makes it easy for others to find. I prefer main first and the others following alphabetically. That way, I can tell if I have scrolled to far when looking for a function.

```
/* parse command line and fill structure */
static int
cliParseCommandLine(char *buf, cmdLineEntry_t *cli)
{
    const char delim[] = "\t\n";
    unsigned int i;
    char *token;

    memset(cli, 0, sizeof *cli);
```

Not a problem but also of no value. Just so you know, forcing a pointer to have all bits zero is not the same as setting it to NULL. For integers and characters all bits zero is 0 but that may not be true for floats and doubles.

```
    token = strtok(buf, delim);
    for (i = 0; i < COUNT(commands); i++) {
        if (!strcmp(token, commands[i])) {
            strcpy(cli->command, token);
            cli->cmd_handler = ftable[i];
            i = 0;
```

You have a logical breakdown here. Either initialize and increment *i* in your for statement or do it in individual statements in the loop, **BUT NOT BOTH**.

```
    for (token = strtok(NULL, delim); token != NULL; ) {
        cli->params[i++] = token;
```

This is a syntax error since `params[i]` is a char and `token` is a char*.

When you fix `params`, it will hold the (up to) four addresses of the arguments in `buf`. This makes it obvious that you want to completely process each input line before obtaining the next (which is consistent with your code in `main`).

But you are inconsistent. You actually copy the command but only

Re: Command Line Interface (CLI): your recommendations

Re: Command Line Interface (CLI): your recommendations

store the argument addresses. What about the command is so special that you need to store the actual characters in `cli->command` rather than simply the address of the command in `buf`?

```
token = strtok(NULL, delim);
}
cli->params_num = i;
return 0;
}
}
return -1;
}

int main(void)
{
char buf[BUFSIZ] = {0};
cmdLineEntry_t cli = { {0}, {NULL}, 0, NULL };
}
```

Another syntax error due to the bad declaration of `params`. That first `NULL` is not a suitable value for an array of four char.

```
while (1) {
printf("%s", PROMPT);
}
```

Use `fflush` here to force the output to your screen.

```
if ( fgets(buf, BUFSIZ, stdin) ) {
/* skip LF/CR/TAB/SP */
if (buf[0] == '\n' || buf[0] == ' ' || buf[0] == '\r' || buf[0]
== '\t')
```

Look up the `isspace()` function if you really want to do this. Are you sure you want to do this since the `strtok` call in `cliParseCommandLine()` will very happily skip over leading white space for you?

```
continue;

/* parse stream */
if ( cliParseCommandLine(buf, &cli) < 0 ) {
printf("Error : invalid command!\n");
continue;
}
else {
```

Re: Command Line Interface (CLI): your recommendations

```
cli.cmd_handler(cli.params_num, cli.params);
```

Another syntax error due to params.

```
    }  
    }  
    else continue; /* EOL */
```

A continue at the end of a loop is somewhat superfluous.

```
    } /* while */  
  
    return 0;  
}
```

Remove del for email

.