

# CShell Project

Shaeq Ahmed, Anthony Liang, Sam Xu

December 3, 2016

## Abstract

You (and up to 2 partners) are to write a shell. The shell should implement as many of these features as possible. The first being the most basic up to the last which is the most complex.

## 1 Features

- Intro screen
- Basic shell functionalities and built in commands
- Implemented Cd and exit
- Simple Redirection
  - Stdin (<)
  - Stdout (>)
  - Pipe (|)
- Dynamically reallocation of user input (need to put this back)
- Parse multiple commands with ; (still need this)
- Ignore weird spacing (need this)
- Prints bash prompt in linux format
  - $$< \textbf{user} > @ < \textbf{hostname} > : < \textbf{cwd} > \$$$
  - tilde ( $\sim$ ) replaces home directory if it appears in current working directory
- Implemented autocomplete binded to TAB (get this back)
- Stores command history, can access with UP arrow key
- Cd prints out error statement if doesn't exist

## 2 Files & Function headers

### 2.1 `void introScreen();`

**Inputs:** None

**Returns:** None

**Explanation:** Prints a few pretty lines when shell is started.

### 2.2 `void shellPrompt();`

**Inputs:** None

**Returns:** None

**Explanation:** Prints out the user prompt in linux format, added custom COLORS to make it more aesthetic!

### 2.3 `int cshell_cd(char *args[]);`

**Inputs:** None

**Returns:** If the directory exists, chdir into it, otherwise return -1. 1 if cd is called by itself, chdir user back to home directory

**Explanation:** Built-in cd command, catches and prints DNE errors

### 2.4 `void cshell_exec(char **args, int background);`

**Inputs:** char \*\*args, int background

**Returns:** If pid == -1 (forking failure), return an error message. If pid == 0 (fork successful), child process runs.

**Explanation:** Fork parent process. Catches any signals and execvp to run the commands.

### 2.5 `void cshell_io(char *args[], char *inputFile, char *outputFile, int option);`

**Inputs:** char \*args[], char \*inputFile, char \*outputFile, int option)

**Returns:** None

**Explanation:** Helper function that helps control the writing and reading of files

### 2.6 `void cshell_pipeHandle(char *args[]);`

**Inputs:** char \*args[]

**Returns:** None

**Explanation:** Helper function responsible for the helper.

### 2.7 `int cshell_run(char *args[]);`

**Inputs:** char \*args[]

**Returns:** int

**Explanation:** Method used to handle the commands entered via the standard loop.

### 2.8 `void signalHandler_child(int p);`

**Inputs:** int p

**Returns:** None

**Explanation:** Signal handler for SIGCHLD

**2.9** `void signalHandler_int(int p);`

**Inputs:** `char *args[]`

**Returns:** None

**Explanation:** Signal handler for SIGINT

**2.10** `void initialize();`

**Inputs:** None

**Returns:** None

**Explanation:** Making sure the subshell is not running as a foreground job. Initialize the pid of the subshell so that it could support job control. Post initialization allows the subshell to have its own child processes. We used the approach explained here to set things up: [www.gnu.org/software/libc/manual/html\\_node/Initializing-the-Shell.html](http://www.gnu.org/software/libc/manual/html_node/Initializing-the-Shell.html)

**2.11** `char **cshell_split_line(char *line, char *delim);`

**Inputs:** `char *line, char *delim`

**Returns:** Array of pointers

**Explanation:** Function for splitting the commands by whitespace (`'\t\r\n\a'`) or any specified delimiter

**2.12** `char **parse_semicolon(char *line);`

**Inputs:** `char *line`

**Returns:** Array of pointers

**Explanation:** Parses multiple commands with `;`. For example the line `"ls -l;echo hello"` would be split into an array of two pointers `"ls -l"` and `"echo hello"`

**2.13** `int is_empty(char * line)`

**Inputs:** `char * line`

**Returns:** 0 or 1

**Explanation:** Checks whether or not the input line contains only whitespace. It returns 1 if is only white space, notifying the code to ask for input again. It returns 0 if it contains arguments.

**2.14** `int main(int argc, char *argv[], char ** envp);`

**Inputs:** `char argc, char **argv, char **envp`

**Returns:** `EXIT_SUCCESS`

**Explanation:** Main process for program. Prints our pretty intro screen and shellPrompt. Reads, parses, and executes command(s). Also implements autocompletion and command history.