

# Project 1

*Due: 8 May 2009*

Beginning with the proto-shell you implemented last week in lab, for this project you will implement `bsh`, a basic shell. It will be able to run arbitrary commands in the foreground or background and redirect the input and output of those commands.

## 1 Spec

Your shell will mimic a subset of the behaviours of `tcsh`, the shell that you are used to interacting with.

Your `bsh` should print a prompt “-> ” and accept input; unescaped spaces in the input separate arguments that are to be passed to the command, where the command is the first token (that is, piece of input) on the line.

If the character ‘>’ appears, the next token of the input is the name of the file to which the output of the command should be sent.

If the character ‘<’ appears, the next token of the input is the name of the file from which the input of the command should be taken.

If the line ends with ‘&’, the command will be run in the background; your shell will immediately print a job number and the process id of the new process, and then give the next prompt. Later, when the process completes, you should print the job number and the text of the command that has completed.

To enter a command or filename that has special characters in it (such as space or ampersand), the special character is preceded with a backslash.

When the user types “`exit`” or hits `^D` on a line by itself, the shell terminates.

That’s it! The rest is just advice.

## 2 A few things to read up on

You’ll need `dup2` to do the redirection. This system call lets you renumber existing open file descriptors—permitting you to put other things in the

slots for standard input (0) and standard output (1).

You'll need `waitpid` or similar to do backgrounding: it lets you provide an option that makes it non-blocking, so that if no child processes have exited, you get control right back rather than hanging.

The `strtok` function and its nonstandard but more robust cousins `strtok_r` and `strsep` take an existing string and parse it into tokens, modifying it by replacing some characters with `'\0'`. This will turn out to be convenient.

The `realloc` function we've seen before: if you have memory that was allocated using `malloc` that turns out to be not big enough, you can `realloc` it, and its prior contents are auto-copied to the new location. Also convenient.

The `sleep` command is a command-line program that sleeps for a certain number of seconds, then exits. Useful for testing.

### 3 A few things to assume at first

Below are some things you can *temporarily* assume in order to get a start on this: they will make it easier to play with the OS calls. Once you get things working, you should "fix" the assumptions one by one (after saving a copy of your work, of course).

- ... that there will never be more than, say, 8 arguments.
- ... that the command line will never take up more than, say, 50 characters.
- ... that there will always be a space between the `<` or `>` or `&` and the tokens that precede and follow them.
- ... that you'll never see a filename with spaces or ampersands or anything that requires a backslash.
- ... that you'll never have more than, say, four processes in the background.
- ... that child processes never exit except just before you need to print a prompt.

## 4 A very incomplete list of things you don't need to worry about

Pretty much, if it's not listed above, you needn't worry about it. There are a lot of things that a complete shell does. Here are a few of them that are clearly related to `bash`'s abilities, but still not required for this project. It's ok if you don't know what some of these things are (although, hey, why not go read up on them too?).

- Redirection of standard error (using “&”).
- Append-redirection (“>>”).
- Here-documents (“<<”).
- Command piping (“|”).
- Switching between foreground and background (“fg”, “bg”, “^Z”).
- Job management (“jobs”, “%1”).
- File globbing (using wildcards “\*”, “?”).
- Path modification (“set path”, “setenv PATH”).
- Environments (“setenv”).
- Filename quoting (enclosing with single- or double-quotes instead of backslash-escaping).

If you're interested in trying some of these things, you might be able to get some extra credit. You should talk to me first, though: some are trickier than others (and I can help you pick an extension of appropriate scope), plus I won't grant extra credit unless at least the vast majority of the base project is working; that needs to be your first priority.

## 5 Handing in

Hand in all relevant `.c` and `.h` files as `proj1`.