Project 2a: Shell (Linux)

Important Dates and Other Stuff

Due: Monday, 10/10 by midnight.

This project is to be done by yourself.

Tests: Tests will be made available shortly.

Questions?

Send questions using piazza or use office hours. If the question is about your code, copy all of of your code into your handin directory (details below) and include your login in your email (you are free to modify the contents of your handin directory prior to the due date). Do **not** put code snippets into piazza (unless they are very short). Also include all other relevant information, such as cutting and pasting what you typed and the results from the screen. In general, the more information you give, the more we can help.

Overview

The basic project description is found here. Please read this carefully in order to understand exactly what to do.

This project is to be done on the lab machines (listed here), so you can learn more about programming in C on a typical UNIX-based platform (Linux).

Differences

There are some modifications to that basic description above. They are:

- There is no need to do parallel commands. However, you can if you like; it's not too bad, really.
- There is a special built-in command called `if` you need to add, which we describe below.

The `if`-statement

The if statement can be used like in the snipped below.

```
prompt> if some_command == 0 then some_other_command fi
```

In this example the wish shell should compare the return code of `some_command` to the number provided (0) and run `some_other_command`, if they are equal.

If statements should always follow the following form and be specified in a single line: `if COMMAND OPERATOR CONSTANT then COMMAND fi`

Example:

Imagine you have a very basic binary named `one` that always returns 1, e.g.

```
int main() {
    return 1;
}
```

and then some binary named `hello` that prints to stdout, e.g.,

```
#include "stdio.h"

int main() {
    printf("Hello World!");
    return 0;
}
```

Now the following command should run `hello` print `"Hello World!"` on the screen:

```
promt> if one == 1 then hello fi
```

However, the following line should not run `hello` and not print anything to the screen:

```
promt> if one != 1 then hello fi
```

More details:

- The left side of the operator is always a command and the right side is a constant. You can also implement support for comparing commands to commands or constants to constants, but that is not required.
- The if statement should allow equal (==) and not-equal (`!=`) operators. You can implement others, e.g. less-than or greater-than, but this is not required for the project.
- Return codes are integers, so you do not need to support string or floating point comparisons. Again, you can add that if it brings you joy, but we will not test for it.
- You only need to support ASCII characters. It is fine if you shell crashes on some obscure unicode character.
- The shell should never crash (except for the case of non-ASCII characters being passed to it). When an invalid command is given or the `if` statement is used incorrectly, the shell should terminate properly and return a non-zero error code.

Notes

Before beginning: If you don't remember much about the Unix/C environment, read this tutorial. It has some useful tips for programming.

This project should be done alone. Copying code (from others) is considered cheating. Read this for more info on what is OK and what is not. Please help us all have a good semester by not doing this.

Handing It In

You should turn in one file, `wish.c` , which we will compile and test.

The handin directory is `~cs537-$sec/handin/$login/p2a` where `$sec` is your section and `$login` is your login. For example, Remzi's login is remzi and he is section 1, and thus he would copy his beautiful code into ~cs537-1/handin/remzi/p2a. Copying of these files is accomplished with the cp program, as follows:

```
prompt> cp wish.c ~cs537-1/handin/remzi/p2a/
```

When done, type `ls ~cs537-1/handin/remzi/p2a` to see that all the files are in place correctly.

Finally, in your p2a directory, please include a README file. In there, describe what you did a little bit. There is no particular requirement for the length of the README; just get in the habit of writing a little bit about what you did, so that another human could understand it.