Princeton University
COS 217: Introduction to Programming Systems
Ish: Development Stages

Stage 0: Preliminaries

Learn the overall structure of *ish* and the pertinent background information.

Study the assignment specification. Study the lecture notes on system calls, processes, and signals. Study literature on UNIX system calls, processes, and signals. Chapter 7 of the book *The UNIX Programming Environment* (Kernighan and Pike, Prentice Hall, Englewood Cliffs, NJ, 1984) is appropriate.

Decide, at least tentatively, on the key modules in your program.

Stage 1: Lexical Analysis

Create the lexical analysis phase of *ish*. That is, create a lexical analyzer whose input is a sequence of characters from a **character array** and whose output is a **token array**.

Write the high-level code that calls your lexical analyzer. The code should first read lines from the .ishrc that resides in your HOME directory until it reaches EOF. (It should print each line that it reads from .ishrc immediately after reading it.) Then the code should read lines from stdin until it reaches EOF (simulated by Ctrl-d).

Testing: Create temporary code that prints the token array that your lexical analyzer produces.

Stage 2: Syntactic Analysis (alias Parsing)

Create the syntactic analysis phase of *ish*. That is, create a parser whose input is a **token** a**rray** and whose output is a **command**.

Write the high-level code that calls your parser. The code should pass the token array (created by your lexical analyzer) to your parser.

Testing: Create temporary code that prints the command that your parser produces.

Stage 3: Executable Binary Commands

Create an initial version of the execution phase of *ish* so it can execute executable binary commands. For now, assume that neither stdin nor stdout are redirected. Use the **fork()** and **execvp()** system calls.

Write the high-level code that calls your executable binary command execution code.

Testing: Use *ish* to execute numerous executable binary commands (**cat**, **more**, etc.) with and without arguments.

Stage 4: Shell Built-In Commands

Enhance the execution phase of *ish*. Specifically, create code that executes the built-in commands **exit**, **cd**, **setenv**, **unsetenv**.

Testing: Test the **cd** built-in command by executing it and the **pwd** and **ls** executable binary commands. Test the **setenv** and **unsetenv** built-in commands by executing them and the **printenv** executable binary command. Execute the **exit** command.

Stage 5: I/O Redirection

Enhance the execution phase of *ish* so it can execute executable binary commands that redirect stdin and/or stdout. Use the **creat()**, **open()**, **close()**, and **dup()** or **dup2()** system calls.

Testing: Repeat the tests for previous stages, adding I/O redirection.

Stage 6: Process Control

Enhance *ish* so it ignores SIGINT signals, but so that its child processes do not necessarily ignore SIGINT signals.

Testing: Execute *ish*, and type Ctrl-c at its prompt; *ish* should ignore the signal. Create a program that intentionally enters an infinite loop. Use *ish* to execute the infinitely looping program. Type Ctrl-c to kill the infinitely looping program.

Stage 7: History (for extra credit)

Enhance *ish* to implement the **history** built-in command and the **!*prefix*** facility.