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 statement. Study the lecture notes on system calls, processes and pipes, and signals. Study literature on UNIX system calls, processes, pipes, 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 ^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 **pipeline** consisting of **commands**.

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 pipeline 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 commands are not part of pipelines, and 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: Pipelines

Suggestion: As a preliminary exercise, write a program that executes the pipeline "ls | sort | more". Note that the parent process should:

- Call **pipe** to create two pipes.
- Call **fork** to create three child processes.
- Call **close** and **dup** so the first child's stdout is connected to the first pipe, the second child's stdin is connected to the first pipe, the second child's stdout is connected to the second pipe, and the third child's stdin is connected to the second pipe.
- Call **execvp** so the first child executes "ls", the second child executes "sort", and the third child executes "more".

Enhance the execution phase of ish so it can execute pipelines consisting of multiple executable binary commands connected with pipes. Use the **fork**, **execvp**, **pipe**, **close** and **dup** (or **dup2**) system calls. Note that the first command of a pipeline may redirect stdin, and that the last command may redirect stdout.

Testing: Repeat the tests for previous stages, adding pipes. Use ish to execute the given sample_ishrc.txt file.

Stage 7: 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 ^C at its prompt; ish should ignore the signal. Create a program that intentionally enters an infinite loop. Use ish to execute the program. Type ^C to kill the program.

Stage 8: History (for extra credit)

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