Princeton University
COS 217: Introduction to Programming Systems
Assignment 6 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. Optionally 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 specified file and whose output is a **token list**.

Write the high-level code that calls your lexical analyzer. The code should first interpret commands from file ~/.ishrc until it reaches EOF. (It should print each line that it reads from ~/.ishrc immediately after reading it.) Then the code should interpret commands from stdin until it reaches EOF (simulated by ^D).

Testing: Create temporary code that prints the token list 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 list** and whose output is a **pipeline** consisting of **commands**.

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

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

Stage 3: Built-In Command Execution

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

Write the high-level code that calls your built-in command execution code.

Testing: Use ish to execute the **exit** command. (See the next stage for testing of **cd**, **setenv**, and **unsetenv**.)

Stage 4: Executable Binary Command Execution

Enhance the execution phase of ish so it can execute pipelines. For now assume that a pipeline consists of a single executable binary command (i.e., no pipes), and that neither stdin nor stdout are redirected. Use the **fork** and **execvp** or **execlp** system calls.

Testing: Use ish to execute numerous executable binary commands (**cat**, **more**, etc.) with and without arguments. Test the **cd** built-in command (implemented in Stage 3) by executing it and the **pwd** and **ls** executable binary commands. Test the **setenv** and **unsetenv** built-in commands (implemented in Stage 3) by executing them and the **printenv** executable binary command.

Stage 5: Executable Binary Command Execution with execv or execl

Enhance the execution phase of ish so it uses **execv** or **execl** instead of **execvp** or **execlp**.

Testing: Repeat the tests for previous stages. Then attempt to execute some commands that are not in the PATH, and make sure ish prints appropriate error messages. Attempt to execute some commands whose files are not executable, and make sure ish prints appropriate error messages. Attempt to execute some commands that are directories, and make sure ish prints appropriate error messages.

Stage 6: Executable Binary Command Execution with I/O Redirection

Enhance the execution phase of ish so it can execute executable binary commands that redirect stdin and/or stdout.

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

Stage 7: Pipeline Execution

Enhance the execution phase of ish so it can execute pipelines consisting of multiple executable binary commands connected with pipes. Use the **fork**, **execv** or **execl**, **pipe**, 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 8: Process Control

Enhance ish so that ^C does not kill ish, but does kill all child processes forked by ish that are currently running.

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 9: History (for extra credit)

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