COS 217 / Fall 2019 / Assignment 7 Grade Report

Name: William Svoboda

Submission Date and Time: Jan 14, 13:38

Precept/Preceptor: P06A / Josh Zhang

Grader: Josh

Required files:	**Readme (basic sections):** *yes* **Feedback:** *yes*

Grading Criterion	Max Points	Points
Readme File		
Justifications of splint and critTer warnings	2	2
Quality from User's Viewpoint		
I/O behavior	8	8
Lexical analysis	9	9
Syntactic analysis	8	8
Handling of external commands	12	12
Handling of shell built-in commands	14 -1.5	12.5
Handling of redirection	8	8
Memory management	5	5
Quality from Programmer's Viewpoint		
Makefile	8	8
Modularity	14	14
Clarity	7 -0.5	6.5
Challenge Part: Signal Handling	5	5
SUBTOTAL	**100**	**98**
Lateness factor		
TOTAL	**100**	**98**

Please see the following pages for detailed grades and comments.

Unlike previous assignment grade reports, the following pages do not contain source code listings. For this assignment there was so much source code that including source code listings in the grade report was impractical.

Readme File

Grades:

Listing and justifications of splint warnings (/1 point)

Listing and justifications of critTer warnings (/1 point)

Your justifications of the splint warnings are good.

Your justifications of the critTer warnings are good.

I'm glad that you thought this assignment as good milestone.

Feedback File

Thank you for submitting the feedback file.

I/O Behavior

Grades:

Handling of empty lines (1/1 point)

Handling of lines containing only white space characters (1/1 point)

Handling of Ctrl-d (2/2 points)

Writing of prompt (1/1 point)

Writing of error messages prefixed with argv[0] (3/3 points)

The program properly handles empty lines.

The program properly handles lines that contain only white space.

The program properly handles Ctrl-d.

The program properly writes the command prompt.

The program writes error messages prefixed with argv[0], as it should.

Very good.

Lexical Analysis

Grades:

Handling of normal lines (4/4 points)

Handling of lines that contain double quotes (3/3 points)

Detection and reporting of unmatched double quote (2/2 points)

The program properly handles normal input lines.

The program properly handles lines that contain double quotes.

The program properly handles lines that contain empty tokens.

The program properly detects and reports "unmatched double quote" errors.

Very good.

Syntactic Analysis

Grades:

Handling of normal token arrays (5/5 points)

Detection and reporting of missing command name (1/1 point)

Detection and reporting of redirection without filename (1/1 point)

Detection and reporting of double redirection (1/1 point)

The program properly handles command names.

The program properly handles command-line arguments.

The program properly handles redirection of stdout.

The program properly handles redirection of stdin.

The program properly handles redirection specifications that appear before command-line arguments.

The program properly detects and reports "missing command name" errors.

The program properly detects and reports "redirection symbol without following file name" errors.

The program properly detects and reports "double redirection of stdin or stdout" errors.

Very good.

Handling of External Commands

Grades:

Handling of external commands that have no command-line arguments (4/4 points)

Handling of external commands that have command-line arguments (4/4 points)

Handling of erroneous external commands (4/4 points)

The program properly handles external commands that have no command-line arguments.

The program properly handles external commands that have command-line arguments.

The program properly handles erroneous external commands.

Very good.

Handling of Shell Built-In Commands

Grades:

Handling of cd commands with one argument (2/2 points)

Handling of cd commands with no arguments (2/2 points)

Handling of erroneous cd commands (0.5/1 point)

Handling of setenv commands with two arguments (2/2 points)

Handling of setenv commands with one argument (1/1 point)

Handling of erroneous setenv commands (0.5/1 point)

Handling of unsetenv commands with one argument (2/2 points)

Handling of erroneous unsetenv commands (0.5/1 point)

Handling of exit commands with no arguments (1/1 point)

Handling of erroneous exit commands (1/1 point)

The program properly handles normal cd commands.

The program properly handles cd commands with no arguments when the HOME environment variable is set.

The program properly handles cd commands when the HOME environment variable is not set.

The program properly handles erroneous cd commands.

The program writes an improper error message when given "cd nonexistingdirectory" or "cd protecteddirectory" or "cd nondirectory". -0.5 from "handling of erroneous cd commands."

The program properly handles normal setenv commands.

The program properly handles setenv commands with one argument.

The program properly doesn't handle some erroneous setenv commands such as "setenv = YYY". The program exits after printing error message. -0.5

The program properly handles normal unsetenv commands.

The program properly doesn't handle some erroneous unsetenv commands such as "unsetenv =". The program exits after printing error message. -0.5

The program properly handles normal exit commands.

The program properly handles erroneous exit commands.

The error messages that the program writes while handling shell built-in commands are essentially correct, but some of them differ from the error messages written by the given program. -0.5 point from one of the "handling of erroneous commands" grading categories.

Very good.

Handling of Redirection

Grades:

Handling of redirection of stdout (3/3 points)

Handling of redirection of stdin (3/3 points)

Handling of erroneous stdin redirection (file name is illegal, file does not exist) (1/1 point)

Handling of erroneous stdout redirection (file name is illegal, directory is not writable) (1/1 point)

The program properly handles redirection of stdout.

The program properly handles redirection of stdin.

The program properly handles commands that redirect both stdout and stdin.

The program properly handles erroneous stdin redirection, that is, redirection of stdin to a file whose name is illegal or that does not exist.

The program properly handles erroneous stdout redirection, that is, redirection of stdout to a file whose name is illegal.

Very good.

Meminfo Analysis

This is the Meminfo report for the program when it executes the commands in commands_demo, with problematic parts shown in boldface:

```
Errors:
Summary Statistics:
   Maximum bytes allocated at once: 362
   Total number of allocated bytes: 12351
Statistics by Line:
          Bytes   Location
           1504   command.c:45
          -1504   command.c:68
          -2112   dynarray.c:112
          -2472   dynarray.c:113
            464   dynarray.c:69
           2472   dynarray.c:84
           1648   dynarray.c:95
            272   ish.c:106
           -880   ish.c:298
           -894   ish.c:307
            608   ish.c:89
           -672   lex.c:119
            -10   lex.c:169
            697   lex.c:65
            -15   lex.c:80
            108   readline.c:31
            776   readline.c:41
             10   readline.c:54
           1824   token.c:35
            668   token.c:40
           -668   token.c:57
          -1824   token.c:58
              0   TOTAL
Statistics by Compilation Unit:
```

```
   0   command.c
   0   dynarray.c
-894   ish.c
   0   lex.c
 894   readline.c
   0   token.c
   0   TOTAL
```

Memory Management

Grades:

Clean Meminfo report when exiting via Ctrl-d (5/5 points)

When the program executes the commands in the given commands_demo file, the Meminfo tool detects no memory corruption.

When the program executes the commands in the given commands_demo file, the Meminfo tool detects no un-freed memory.

Very good.

Makefile

Grades:

"all" rule and rules to build the executable binary files (2/2 points)

Existence of dependency rules for .o files (2/2 points)

Proper dependencies of .o files upon .c and .h files (4/4 points)

readLine.c should #include readLine.h.
readLine.o doesn't depend upon readLine.h.
In general, .o files depend upon its .c and all #included header files. -0

Splint Analysis

This is the splint report for the program, with important warnings shown in boldface. The grades in the Modularity section are based, in part, upon the quality of the splint report.

```
splint ish.c command.c builtin.c syn.c dynarray.c
readline.c token.c lex.c
Splint 3.1.2 --- 19 Dec 2018

builtin.c: (in function handleBuiltIn)
```

```
builtin.c:49:16: Variable uLength used before definition
  An rvalue is used that may not be initialized to a value
on some execution
  path. (Use -usedef to inhibit warning)
builtin.c:83:16: Variable uLength used before definition
builtin.c:110:16: Variable uLength used before definition
builtin.c:118:26: Variable ptr used before definition

Finished checking --- 4 code warnings
```

The splint report is good.

Modularity

Grades: Your submission should demonstrate your understanding that:

A program should consist of multiple logically coherent modules. In particular, ish should should consist of:

A main module, a lexical analysis module, a syntactic analysis module, and a dynarray module (1/1 point)

One additional logically coherent module (2/2 points)

A second additional logically coherent module (1/1 point)

A module's interface should declare only those functions that clients must see. Other functions should be defined as static in the implementation. (2/2 points)

A module's interface should not reveal the module's data. Data should be encapsulated with functions. (3/3 points)

Global variables should be avoided. If they cannot reasonably be avoided, then they should be static if they can be and const if they can be. (1/1 point)

A module's implementation should consist of multiple logically coherent functions, each of which performs a single, small, well-defined task. (2/2 points)

A module's implementation should validate function parameters by calling assert(). (1/1 point)

A module's implementation should detect programmer errors, report them (via perror() or strerror() when possible), and exit. (1/1 point)

Modules defined:

ishlex, ishsyn, ish, dynarray, lex, syn, token, command, builtin, and readLine.

Good.

Concerning the module interfaces:

Headers look good.

Concerning the module implementations:

Implementation files look good.

CritTer Analysis

This is the critTer report for student-composed .c files, with important warnings shown in boldface. The grades in the Clarity section are based, in part, upon the quality of the critTer report.

```
Checking ___builtin.c

    ___builtin.c: line 25: medium priority:
    Function names should be prefixed with module names;
    function name handleBuiltIn does not match module name
___builtin.c

    ___builtin.c: line 28: medium priority:
    Variable/function name 'cd' is too short

    ___builtin.c: line 29: medium priority:
    Variable/function name 'ex' is too short

Checking ___command.c

    ___command.c: line 39: medium priority:
    Do you want to validate 'pcIn' through an assert?

    ___command.c: line 39: medium priority:
    Do you want to validate 'pcOut' through an assert?

Checking ___ish.c
    ___ish.c: line 226: low priority:
    A loop should consist of fewer than 35 lines;
    this loop consists of 80 lines; consider refactoring

    ___ish.c: line 67: medium priority:
    Do you want to validate 'pvElement' through an assert?

    ___ish.c: line 67: medium priority:
    Do you want to validate 'pvExtra' through an assert?
```

```
   ___ish.c: line 123: medium priority:
   Do you want to validate 'pcIn' through an assert?

   ___ish.c: line 123: medium priority:
   Do you want to validate 'pcOut' through an assert?

   ___ish.c: line 254: low priority:
   This area is deeply nested at level 4, consider
refactoring

   ___ish.c: line 259: low priority:
   This area is deeply nested at level 5, consider
refactoring

   ___ish.c: line 262: low priority:
   This area is deeply nested at level 5, consider
refactoring

   ___ish.c: line 266: low priority:
   This area is deeply nested at level 5, consider
refactoring

   ___ish.c: line 283: low priority:
   This area is deeply nested at level 5, consider
refactoring

   ___ish.c: line 274: low priority:
   This area is deeply nested at level 6, consider
refactoring

   ___ish.c: line 277: low priority:
   This area is deeply nested at level 6, consider
refactoring

   ___ish.c: line 287: low priority:
   This area is deeply nested at level 5, consider
refactoring

   ___ish.c: line 296: low priority:
   This area is deeply nested at level 5, consider
refactoring
Checking ___ishlex.c

   ___ishlex.c: line 26: medium priority:
   Do you want to validate 'pvElement' through an assert?

   ___ishlex.c: line 26: medium priority:
   Do you want to validate 'pvExtra' through an assert?
```

```
   ___ishlex.c: line 35: medium priority:
   Do you want to validate 'pvElement' through an assert?

   ___ishlex.c: line 35: medium priority:
   Do you want to validate 'pvExtra' through an assert?

Checking ___ishsyn.c

   ___ishsyn.c: line 28: medium priority:
   Do you want to validate 'pvElement' through an assert?

   ___ishsyn.c: line 28: medium priority:
   Do you want to validate 'pvExtra' through an assert?

Checking ___lex.c

   ___lex.c: line 70: low priority:
   A loop should consist of fewer than 35 lines;
   this loop consists of 124 lines; consider refactoring

   ___lex.c: line 27: medium priority:
   Function names should be prefixed with module names;
   function name freeTokens does not match module name
___lex.c

   ___lex.c: line 32: medium priority:
   Function names should be prefixed with module names;
   function name lexLine does not match module name
___lex.c

   ___lex.c: line 32: low priority:
   A function should consist of fewer than 140 lines;
   this function consists of 160 lines; consider
refactoring

   ___lex.c: line 27: medium priority:
   Do you want to validate 'pvElement' through an assert?

   ___lex.c: line 27: medium priority:
   Do you want to validate 'pvExtra' through an assert?

   ___lex.c: line 90: low priority:
   This area is deeply nested at level 4, consider
refactoring

   ___lex.c: line 116: low priority:
```

```
   This area is deeply nested at level 4, consider
refactoring

   ___lex.c: line 129: low priority:
   This area is deeply nested at level 4, consider
refactoring

   ___lex.c: line 137: low priority:
   This area is deeply nested at level 4, consider
refactoring

   ___lex.c: line 152: low priority:
   This area is deeply nested at level 4, consider
refactoring

Checking ___readline.c
Checking ___syn.c

   ___syn.c: line 55: low priority:
   A loop should consist of fewer than 35 lines;
   this loop consists of 71 lines; consider refactoring

   ___syn.c: line 19: medium priority:
   Function names should be prefixed with module names;
   function name synTokens does not match module name
___syn.c

   ___syn.c: line 76: low priority:
   This area is deeply nested at level 4, consider
refactoring

   ___syn.c: line 90: low priority:
   This area is deeply nested at level 4, consider
refactoring

   ___syn.c: line 99: low priority:
   This area is deeply nested at level 4, consider
refactoring

   ___syn.c: line 113: low priority:
   This area is deeply nested at level 4, consider
refactoring

Checking ___token.c
```

The critTer report contains some unimportant warnings that easily could be eliminated. It would have been better to eliminate them.

Clarity

Grade:

> File comments, function comments, local comments, comments on typedefs, comments on structure type definitions, and comments on structure fields (4.5/5 points)
>
> Names, line lengths, indentation, absence of magic numbers, etc. (2/2 points)

The most important function-level comment – the one on the main() function – is present but is incomplete. It does not mention when or what the program writes to stdout and writes to stderr. -0.5 point.

Good comments on typedefs.

Good comments on structure type and field definitions.

Good local comments.

Good function names. Good to prefix function names with module names.

Good enumerations.

Handling of Signals

Grades:

> Handling of SIGINT signals in the child process (1/1 point)
>
> Handling of SIGINT signals in the parent process when a child process is not running (3/3 points)
>
> Handling of SIGINT signals in the parent process when a child process is running (1/1 point)

The program properly handles SIGINT signals in the child process.

The program properly handles SIGINT signals in the parent process when a child process is not running.

The program properly handles SIGINT signals in the parent process when a child process is running.

Very good.

Miscellaneous

Excellent work.