CSCI 347 Computer Systems II
Summer 2018
Assignment 2

Submitting Your Work

This assignment is worth 15% of the grade for the course. Save your program files (including header files and makefile) in a zipped tar file and submit the file via the **Assignment 2 Submission** item on the course web site. You must submit your assignment by 2:00 pm on Monday, August 6.

Your assignments will be evaluated on correct functionality and conformance to the coding standards described at the end of this assignment specification.

Command Line Interpreter

Your task for this assignment is to implement a command line interpreter (CLI) which will issue a prompt "`$>` " on the terminal window on which the program is run, read a command from the keyboard and execute that command. The CLI must terminate when the `exit` command is received or `Crtl-D` is input from the keyboard, signaling end of input.

Commands handled by the CLI may have command line arguments and may use the Unix input and output redirection symbols "<" and ">", respectively. The CLI must be able to run any Unix command or utility and any program which is accessible from the CLI's current working directory. This is to be done by creating a child process and passing the command and its NULL-terminated array of command line arguments to an appropriate exec function within the child process. Any I/O redirection in the CLI command must be used by the child process.

The CLI must also implement its own version of the following five commands:

1. `exit`

   This causes the CLI to terminate. It is an alternative to `Ctrl-D`.

2. `cd` *dir*

   The `cd` command changes the CLI's current working directory to the directory specified by the absolute or relative path given as a command line argument. This is to be implemented as an **internal** command, using instructions within the CLI, without creating a child process. You should use the `chdir()` function declared in `unistd.h` to implement this command. The CLI's `cd` command, unlike the Unix `cd` command, is to produce output to `stdout`:

   `cwd changed to` *path*

If the `chdir()` function fails, the output of the `cd` command should be

`No such directory: dir`

Note that `cd` does not use `stdin`, but it does use `stdout`.

Examples of the `cd` command are shown below:

```
david@Argento:~/CSCI352Sp2015/Assignment2/Solution$ ./Assg2
$> cd ..
cwd changed to /home/david/CSCI352Sp2015/Assignment2
$> cd nosuchdirectory
no such directory: nosuchdirectory
$> cd Solution
cwd changed to /home/david/CSCI352Sp2015/Assignment2/Solution
$> cd junk
cwd changed to /home/david/CSCI352Sp2015/Assignment2/Solution/junk
$> cd ..
cwd changed to /home/david/CSCI352Sp2015/Assignment2/Solution
$>
```

3. `pwd`

The `pwd` command must display the absolute path of the CLI's current working directory. This command must also be implemented as an **internal** command, using the `getcwd()` function declared in `unistd.h`. Unlike the Unix `pwd` command, the CLI's `pwd` command does not skip to a new line after displaying the path, thus making the directory path part of the prompt for the next command (if `stdout` has not been redirected to a file).

Note that `pwd` does not use `stdin`, but it does use `stdout`.

```
david@Argento:~/CSCI352Sp2015/Assignment2/Solution$ ./Assg2
$> pwd
/home/david/CSCI352Sp2015/Assignment2/Solution $>
```

4. `listf dir1 dir2 …`

The `listf` command is CLI's replacement for the Unix `ls` command. Its purpose is to list the contents of one or more directories or the names and details of individual files specified as command line arguments. The files in each directory are to be listed in alphabetic order, ignoring upper- and lower-case. If no directory is specified, `listf` must list the contents of CLI's current working directory.

The `listf` command must be implemented as an **external** command. It must be a separate program which is run by a child of the CLI process. It must be possible to run the `listf` command regardless of the CLI's current working directory.

The `listf` command does not read from `stdin`, but does write to `stdout`.

The `listf` command has the following command line options:

`-l` (letter "el", not the digit "one") Display the long form of output, including the following information, separated by a single space:

- File type ("`_`" for regular files, "`d`" for directories, "l" for links, "o" for any other types)
- File read, write and execute access permission bits for the user (owner), group, and other, using the "`rwx_`" format of the Unix `ls` command.
- Number of links to the file
- Name of the file owner.
- Name of the file owner's group.
- File size in bytes, right-justified in a width exactly needed to display the size of the largest file in the directory.
- Last modification date, in the format "`Mmm/dd/yy-hh:nn`", where "mm" is the 2-digit month, "dd" is the 2-digit day of the month, "yy" is the 2-digit year, "hh" is the 2-digit hour of the day in 24-hour format and "nn" is the 2-digit display of the minute past the hour. The modification time is preceded by "M".
- The file name

An example of this output is shown below.

```
$> listf -l
5 entries found
 -rw-rw-r-- 1 david david   10 M04/27/15-12:17  calcout
 -rw-rw-r-- 1 david david  275 M04/27/15-12:15  data
 drwxrwxr-x 2 david david 4096 M04/27/15-15:56  Examples
 drwxrwxr-x 3 david david 4096 M04/27/15-15:47  Solution
 drwxrwxr-x 2 david david 4096 M04/25/15-17:18  test
```

`-a` Provided `-l` is also specified, this causes the same output as the `-l` option, except that the last access time is shown instead of the last modification time. The `-a` option has no effect if the `-l` option is not used. The access time is preceded by "A".

```
$> listf -la
5 entries found
 -rw-rw-r-- 1 david david   10 A04/27/15-12:17  calcout
 -rw-rw-r-- 1 david david  275 A04/27/15-12:15  data
 drwxrwxr-x 2 david david 4096 A04/28/15-11:15  Examples
 drwxrwxr-x 3 david david 4096 A04/27/15-15:47  Solution
 drwxrwxr-x 2 david david 4096 A04/28/15-11:07  test
```

`-c` Provided `-l` is also specified, this causes the same output as the `-l` option, except that the creation time is shown instead of the last modification time. The `-c` option has no effect if the `-l` option is not used. The creation time is preceded by "C".

```
$> listf -lc
5 entries found
 -rw-rw-r-- 1 david david   10 C04/27/15-12:17  calcout
 -rw-rw-r-- 1 david david  275 C04/27/15-12:15  data
 drwxrwxr-x 2 david david 4096 C04/28/15-11:15  Examples
 drwxrwxr-x 3 david david 4096 C04/27/15-15:47  Solution
 drwxrwxr-x 2 david david 4096 C04/25/15-17:18  test
$>
```

`-m` Provided `-l` is also specified, this causes the same output as the `-l` option. The `-m` option has no effect if the `-l` option is not used.

These command line options may be specified separately, as in:

```
listf -l -a
```

Or together, as in:

```
listf -la
```

More than one of the date options may be used, for example:

```
listf -lmac
```

Will display all three times for each file, as in the example below.

```
$> listf -lmac
5 entries found
 -rw-rw-r-- 1 david david   10 M04/27/15-12:17 A04/27/15-12:17 C04/27/15-12:17  calcout
 -rw-rw-r-- 1 david david  275 M04/27/15-12:15 A04/27/15-12:15 C04/27/15-12:15  data
 drwxrwxr-x 2 david david 4096 M04/27/15-15:47 A04/27/15-15:47 C04/27/15-15:47  Examples
 drwxrwxr-x 3 david david 4096 M04/27/15-15:47 A04/27/15-15:47 C04/27/15-15:47  Solution
 drwxrwxr-x 2 david david 4096 M04/25/15-17:18 A04/27/15-08:50 C04/25/15-17:18  test
```

If no command line options are used, `listf` displays a list of the names files in the current or specified directory, all on one line, separated by a single space, as in the example below.

```
$> listf
5 entries found
calcout data Examples Solution test
```

5. `calc`

This is a simple calculator, which takes single arithmetic operations from `stdin` and writes the formatted expression and the result to `stdout`, as shown in the examples below. The `calc` command only handles integer numbers, addition ("+" operator), subtraction ("-" operator), multiplication ("*"

operator) and integer division ("/" operator). There may be multiple lines of input to `calc`, but only one arithmetic expression per line. The `calc` command terminates on receiving a `Ctrl-D` (end of file) from `stdin`. The `calc` command must be able to handle arbitrary (including non-existent) spacing between the operands and operator in each expression.

The `calc` command must be implemented as an **external** command. It must be a separate program which is run by a child of the CLI process. It must be possible to run the `calc` command regardless of the CLI's current working directory.

The `calc` command reads from `stdin`, and writes to `stdout`.

An example of input and output for `calc` is shown below.

```
$> calc
23/5
23 / 5 = 4
18          *4
18 * 4 = 72
92+ 17
92 + 17 = 109
92-17
92 - 17 = 75
```

I/O Redirection

The CLI must recognize the standard Unix I/O redirection symbols ">" and "<" on the command line.

The `stdin` redirection symbol "<" is followed by optional spaces and then the name of the file to be used as input instead of the keyboard.

The `stdout` redirection symbol ">" is followed by optional spaces and then the name of the file to be used for output instead of the terminal window.

The I/O redirection can be anywhere on the command line, following the name of the command.

On completion of the command, `stdin` and `stdout` must be restored to their defaults for the next command. The "`%>` " prompt for the next command must always appear in the CLI terminal window and never be written to a file which is the target of output redirection.

Saving your files in a zipped tar file

You need to submit all your C and header files, as well as your makefile. First you need to bundle them up into a single *tar* file. The term "tar" is an abbreviation of "tape archive" and goes back to the days when people would save a back-up copy of their files on magnetic tape. Nowadays, with the price of large disk drives so low, nobody uses tape anymore, but the concept of tar files survives.

1. Use the command:

   ```
   tar -cf myfiles.tar *.c *.h Makefile
   ```

The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

Following the name of the tar file, we list the files to be included in the tar file. In this case, we want all the files in your current directory whose names end with `".c" or ".h",` as well as `Makefile`.

2. If you now use the command `ls` you should now see the file myfiles.tar in your directory.
3. If you use the command

   ```
   tar -tf myfiles.tar
   ```

   it will list the files within the tar file.
4. Now compress the tar file using the gzip program:

   ```
   gzip myfiles.tar
   ```
5. By using the `ls` command again, you should see the file myfiles.tar.gz in your directory. This is the file that you need to submit through the **Assignment 2 Submission** link in the course web site.

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.
2. Avoid the repeated use of numeric constants. For any numeric constants used in your program, use #define preprocessor directives to define a macro name and then use that name wherever the value is needed.
3. Use comments at the start of the program to identify the purpose of the program, the author and the date written.
4. Use comments at the start of each function to describe the purpose of the function, the purpose of each parameter to the function, and the return value from the function.
5. Use comments at the start of each section of the program to explain what that part of the program does.
6. Use consistent indentation.
7. Do not let functions get large. Any more than 20 lines of code in a function probably means that the function is too complex and you need to break its task into subtasks, each handled by a separate function.