A small shell with I/O redirections

General Overview:
Write a C program called **techshell.c** that repeatedly allows a user to enter input that gets interpreted as a shell command.

The general flow of processing input should be:

1. Your program first should parse the command line and determine whether it contains I/O redirection such as < for input from a file or > for output to a file.

2. The command is then executed in a forked child process that uses an **exec** call.

Concerning the prompt in the shell, your program should provide an informative prompt that contains the current working directory, followed by a $, and ending with a space. The prompt should be updated if the current working directory changes. Here's an example that illustrates this:

```
/home/alice$ cd ..
/home$
```

Your Submission:
As part of your submission, you must include a separate file called README (a text file with **no** extension) that provides the following details:

1. Your name (and your partner's name if applicable);

2. A description of the inner workings of your program; and

3. If you have not fully implemented shell functionality as described above, list the parts that work (and how to test them if it's not obvious) so that you can receive partial credit.

You must also include output from testing your program using **script**, a program that records terminal sessions (this should already included on your machine). When you run **script**, a script session of your program is started (and saved in a file called **typescript**). The following example shows what running script before running your **techshell** executable would look like, and what exiting your **techshell** program and exiting **script** would look like.

```
~$ script
Script started, file is typescript
~$ ./techshell
cd Desktop
Desktop$ exit
~$ exit
exit
Script done, file is typescript
~$
```

Submit a single **.zip** archive file that contains:

1. The README file described above;
2. Output from testing your program via **script**; make sure to demonstrate:
    1. Simple UNIX commands;
    2. I/O redirection (< and >); and
    3. Error conditions (e.g., invalid commands, errors, etc).
3. All **.c** source files required to compile your program (do not include any executable or object files); and

Template:
Be sure to implement good programming practices. For example, your **main** function should merely be a driver (i.e., farm out various functions to subroutines). A possible main program is provided as follows:

```
//// Functions to implement:

// Display current working directory and return user input
char* CommandPrompt();

// Process the user input (As a shell command)
// Note the return type is a ShellCommand struct
struct ShellCommand ParseCommandLine(char* input);

// Execute a shell command
// Note the parameter is a ShellCommand struct
void ExecuteCommand(struct ShellCommand command);


int main() // MAIN
{
    char* input;
    struct ShellCommand command;

    // repeatedly prompt the user for input
    for (;;)
    {
        // get the user's input
        input = CommandPrompt();

        // parse the command line
        command = ParseCommandLine(input);

        // execute the command
        ExecuteCommand(command);
    }

    exit(0);
}
```

Testing:
Your program should be able to handle the following examples (via sample **script** output). Note that your program will be tested with more than these commands (including more error handling cases):

```
~$ ./techshell
~$ cd Desktop
Desktop$ pwd
/home/alice/Desktop
Desktop$ cd ..
$ pwd
/home/alice
$ ls -lh techshell.c
-rw-r--r-- 1 alice alice 3.9K Oct 27 15:08 techshell.c
$ chmod 400 techshell.c
$ ls -lh techshell.c
-r-------- 1 alice alice 3.9K Oct 27 15:08 techshell.c
$ ls /root
ls: cannot open directory /root: Permission denied
$ horseface
Error 2 (No such file or directory)
$ cd /root
Error 13 (Permission denied)
$ ps
  PID TTY          TIME CMD
 6271 pts/3    00:00:00 bash
 7415 pts/3    00:00:00 techshell
 7460 pts/3    00:00:00 ps
$ whereis ps
ps: /bin/ps /usr/share/man/man1/ps.1.gz
$ /bin/ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
alice     444  0.0  0.0   7236  5244 pts/1    Ss+  07:57   0:01 bash
alice    6271  0.0  0.0   7220  5032 pts/3    Ss   14:18   0:00 bash
alice    7415  0.0  0.0   2172  1324 pts/3    S+   15:08   0:00 ./techshell
alice    7483  0.0  0.0   5228  2372 pts/3    R+   15:12   0:00 ps u
$ ls
Desktop  Documents  Music  techshell  Downloads  Pictures  techshell.c
Videos
$ ps u > ps.out
$ wc -l < ps.out
5
$ wc -l < ps.out > wc.out
$ cat wc.out
5
$ ls
Desktop  Documents  Music  techshell  Downloads  Pictures  techshell.c
Videos   ps.out     wc.out
$ rm ps.out wc.out
$ ls
Desktop  Documents  Music  techshell  Downloads  Pictures  techshell.c
Videos

$ exit
~$
```

Additional Notes:

1. Comment your source code appropriately and include your name(s) in the files.
2. In order to receive a grade for this assignment, **you need to demo your work to the instructor.**
3. You may work in groups of two for this project (after express approval from the instructor).
4. Do not use the `system(...)` function or `popen(...)`

Hints:

1. To get the value of an environment variable, **getenv** is your friend;
2. The **exec** function of most use may very well be **execvp**;
3. The following may just work to redirect **stdin** from a file:
```
FILE* infile = fopen(my_input_file, "r");
dup2(fileno(infile), 0);
fclose(infile);
```
4. The following may just work to redirect **stdout** to a file:
```
FILE* outfile = fopen(my_output_file, "w");
dup2(fileno(outfile), 1);
fclose(outfile);
```
5. To build command line arguments, **malloc** and **realloc** are your friend;
6. To copy strings, you may want to try **strdup**;
7. To handle errors, **errno** and **strerror** are your friends; and
8. **#define DEBUG** may greatly help with debugging (i.e., use toggle-able print statements).

Rubric:

- Your program compiles with no errors (50 pts)
- I/O redirections work (50 pts)
- All other commands handled through an exec function (60 pts)
- Invalid user input handled correctly (20 pts)
- README file provided (20 pts)