**Program Summary**

The purpose of this program was to simulate the operations of a linux shell as an extension of the first homework assignment. Commands may be provided by the user at run time or else a prompt is displayed and the program waits for the user to input a command. The command is passed into the predefined parse\_command function. In this function, a character pointer iterates over each character in the line argument. When an identifier is found (space, pipe, null, or redirect), the previous characters are copied into a local token variable. The program checks if the token is null or empty to avoid adding spaces as arguments in the case of a space as the first character in the command or multiple spaces in a row. The program then checks the global flag variables to determine where the token should be copied to. If there has not been a pipe or redirect yet, cmd1 dynamically allocates space for the token. If a pipe has occurred but no redirects have, cmd2 allocates space. If a redirect has been found, the program determines which redirect has occurred and copies the token to either outfile or infile. Null terminators are appended to cmd1 and cmd2. The function returns the value of a separate function that checks all the global flags to determine the commands' complexity. If the user enters "quit" as a command, the program exits.

One of the most difficult parts of this assignment was getting the user input to parse correctly. Apparently, the fgets() function will put a newline character at the end of the string and so executing the command would not work. The other difficult part of this assignment was getting the function to recognize token terminators (that is, spaces, nulls, pipes, and redirects). I rewrote the function from the first assignment to not use the strtok() function, and instead count the number of characters until a token terminator is found and then dynamically allocate enough space for the token. When the program finds the beginning of an output redirection (>) it checks if the next character is also an output redirection, sets the appropriate flag, and increases the pointer indices if necessary.

**Program ToDo List**

This program fixed the problem with the first assignment not dynamically allocating space for command arguments. However, this program still assumes only one pipe and one redirect can be used. After two or more pipes, the arguments are continuously put into cmd2. Likewise, if more than one redirect is detected, the outfile or infile will be overwritten. This can be fixed by having an array pointer arrays to store multiple arguments after each pipe, and having pointer arrays to store multiple outfile or infiles. This program is also limited in that it only currently executes simple commands (no pipe, no redirect).

**Test Cases**

All of the test cases provided in the assignment description were applied and the program output the correct response. The program outputs the result from the parse\_command function and the result from executing the command (given that it is a simple command). Test cases were designed such that each result option (0 - 9) was returned.