In this assignment I began to have a deeper understanding for how the shell parses commands from the user. For this homework I have began to understand how robust a shell must be in order to parse commands from a user. During this assignment it was difficult to understand how to keep the program adaptable enough to ensure that there won't be a dramatic change between the future assignments.

For my test cases I used the examples provided and received the expected output. In addition I passed additional arguments to make sure that the program could handle multiple arguments for a command. For the output redirection I made sure that all the different redirect ">>", ">", and "<" would be detected.

```
//Author: Cory McDonald
#include <stdio.h>
#include <string.h>

//Gotta have my bools
typedef int bool;
#define true 1
#define false 0

int main ( int argc, char *argv[] )
{
  //Making sure arguments were passed in
        if(argc>0)
        {
          //Token to replace in the arguments passed in
                char s[2] = " ";
                char *token;
                //If the output is redirected we must know
                bool isOutputRedirected = false;
                char outputRedirectedTo[3] = ""; //Could be >>,>,<
                //If the command is piped out then we will want to read in the command the user
wants to complete
                bool reset = true;

                token = strtok(argv[1], s); //Tokenizing
                while(token != NULL )
                {
                        if (strstr(token, "quit")) //Quiting
                        {
                                printf("Program terminates successfully by the user\n");
                                break;
                        }
                        else if(reset == true) //Taking in command, otherwise we will assume it is
an argument
                              {
                                        printf("The user command or program is: [%s]\n", token );
                                        reset = false;
                              }
                              else if(strstr(token, "|")) //Pipin'
                              {
                                        reset = true;
                                        printf("Pipe: yes\n");
                              }
                              else if(strstr(token,">>") || strstr(token,">") || strstr(token,"<")) //Output
redirected
                              {
                                        isOutputRedirected = true;
                                        strncpy(outputRedirectedTo, token, sizeof(outputRedirectedTo));
                                        outputRedirectedTo[sizeof(outputRedirectedTo) - 1] = '\0';
                                        printf("Output Direction: %s\n", token);
                              }
                              else if (isOutputRedirected == true)
                              {
                                        if(strstr(outputRedirectedTo,">>"))
                                        {
                                                  printf("Output file: %s\n", token);
                                        }
                                        else if (strstr(outputRedirectedTo,">"))
                                        {
                                                  printf("Output overwritten: %s\n", token);
                                        }
                                        else if(strstr(outputRedirectedTo,"<"))
                                        {
                                                  printf("Input: %s\n", token);
                                        }
                              }
                              else
                              {
                                        printf("The command line argument to the user command and
program is: [%s]\n", token );
                              }

                              token = strtok(NULL, s);
                    }
          }
          return 0;
}
```