```
// Jessica Elkins
// CS332 Lab 7
// 3/5/20

//TO COMPILE: gcc lab7.c -o lab7
//TO RUN: ./lab7 <commands file>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <time.h>

void forkFunc(char *array[BUFSIZ], char line[BUFSIZ]);
void outputFunc(char line[BUFSIZ], char *startT, char *endT);

void tokFunc(char line[BUFSIZ]){
        int i = 0;
        char line2[BUFSIZ];
        strcpy(line2, line);
        char *array[BUFSIZ];
        char *token = strtok(line, " \n");
        //tokenizing the input line
        while(token != NULL){
                array[i++] = token;
                token = strtok(NULL, " \n");
        }
        // calling fork function
        forkFunc(array, line2);
}

void forkFunc(char *array[BUFSIZ], char line[BUFSIZ]){

        // to get the time
        time_t currentT;
        time(&currentT);
        char *startT;
        char *endT;
        startT = (char*)malloc(sizeof(char)*100);
        endT = (char*)malloc(sizeof(char)*100);

        pid_t pid;
        int status;

        char *pathName;
        pathName = (char*)malloc(sizeof(char)*BUFSIZ);
        strcpy(pathName, array[0]);

        //start time
        startT = ctime(&currentT);

        pid = fork();
        //if in the child process
        if(pid == 0){
                startT = ctime(&currentT);
                execvp(pathName, array);
                printf("If you see this then excl failed.\n");
                perror("execv");
                exit(-1);
        }else if(pid > 0){
```

```
                wait(&status);
                if(WIFEXITED(status)){
                        // use print statement for debugging
                        //printf("Child process exited with status = %d\n", WEXITSTATUS
(status));

                        //end time
                        endT = ctime(&currentT);

                        //printing output to log file
                        outputFunc(line, startT, endT);
                } else {
                        printf("Child process did not exit normally.\n");
                }
        }else {
                perror("fork");
                exit(EXIT_FAILURE);
        }


}

void outputFunc(char array[BUFSIZ], char *startT, char *endT){
        //opening output file
        FILE *fptr = fopen("output.log", "a");

        // if file pointer is null
        if(fptr == NULL){
                printf("Error opening output.log. Exiting. \n");
                exit(-1);
        }

        //format output to output log
        fprintf(fptr, "%s \t %s \t %s\n", array, startT, endT);

        //closing file pointer
        fclose(fptr);

}

int main(int argc, char *argv[]){
        char line[BUFSIZ];

        //error message if program is not executed right
        if(argc != 2){
                printf("Usage: %s <commands file> \n", argv[0]);
                exit(-1);
        }

        //opening the input file
        FILE *fptr = fopen(argv[1], "r");

        //error message if file pointer is null
        if(fptr == NULL){
                printf("Error opening %s. Exiting.\n", argv[1]);
                exit(-1);
        }

        //reading each line of the file and sending it to tokFunc to get tokenized

        while(fgets(line, BUFSIZ, fptr) != NULL){
                tokFunc(line);
        }
```

```
        //closing file pointer
        fclose(fptr);

        return 0;
}
```