```
/*****************************************************************************
/
/      filename:  utils.c
/
/   description:  Impliments all of the IO and exec for myshell.
/
/        author:  Paladino, Zac
/      login id:  cps346−n1.16
/
/         class:  CPS 346
/    instructor:  Perugini
/    assignment:  PJ #1
/
/      assigned:  January 28, 2009
/           due:  February 20, 2009
/
/****************************************************************************/

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

#include "makeargv.h"
#define DELIMITERS " \t"
typedef int bool;
#define TRUE 1
#define FALSE 0
#define CREATE_FLAGS (O_WRONLY | O_CREAT | O_TRUNC)
#define CREATE_AP_FLAGS (O_WRONLY | O_CREAT | O_APPEND)
#define CREATE_READ_FLAG (O_RDONLY)
#define CREATE_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

//stripIO()
//This function takes out all of the input and output files in the line.
void stripIO(char** line,char*** toklin, char** newl, char** opf, char** ipf,int
* numtok){
       bool RO = FALSE;
       bool RI = FALSE;
       (*numtok) = makeargv((*line), DELIMITERS, &(*toklin));
       int i;
       for(i=0; i<(*numtok); i++){
          if((strchr((*toklin)[i],'<'))||(strchr((*toklin)[i],'>'))){
            if(strchr((*toklin)[i],'<')){
              RI = TRUE;
            }
            else if(strchr((*toklin)[i],'>')){
              RO = TRUE;
            }
          }
          else{
           if(!RO && !RI){
             strcat((*newl), (*toklin)[i]);
             strcat((*newl), " ");
             RO=FALSE;
             RI=FALSE;
           }
           if(RO){
             strcpy((*opf), (*toklin)[i]);
             RO = FALSE;
           }
           if(RI){
             strcpy((*ipf), (*toklin)[i]);
             RI = FALSE;
```

```
            }
          }
       }
}

//RedIO()
//This function checks to see if IO redirection is needed and completes it if tr
ue.
void RedIO(char*** toklin, char** opf, char** ipf,int* REDSI,int* REDSO, int* nu
mtok,
               int* fd, int* fdi, int* fdr, int* EXIT){
       int i;
       for(i=0;i<(*numtok);i++){
           if(strchr((*toklin)[i],'>')){
               if(strstr((*toklin)[i],">>")){
                  (*fd) = open((*opf), CREATE_AP_FLAGS, CREATE_MODE);
               }
               else{
                  (*fd) = open((*opf), CREATE_FLAGS, CREATE_MODE);
               }
               if((*fd) == −1){
                  perror("Failed to open file");
                  (*EXIT) = TRUE;
                  return;
               }
               if(dup2((*fd), STDOUT_FILENO)==−1){
                  perror("Failed to redirect standard output.\n");
                  (*EXIT) = TRUE;
                  return;
               }
               if(close((*fd))==−1){
                  perror("Failed to close the file");
                  (*EXIT) = TRUE;
                  return;
               }
               (*REDSO) = TRUE;
           }
          if(strchr((*toklin)[i],'<')){
             (*fdr) = dup(STDIN_FILENO);
             (*fdi) = open((*ipf), CREATE_READ_FLAG, CREATE_MODE);
             if((*fdi) == −1){
               perror("Failed to open file");
               (*EXIT) = TRUE;
               return;
             }
             if(dup2((*fdi), STDIN_FILENO)==−1){
               perror("Failed to redirect standard input.");
               (*EXIT) = TRUE;
               return;
             }
             if(close((*fdi))==−1){
               perror("Failed to close the file");
               (*EXIT) = TRUE;
               return;
             }
             (*REDSI) = TRUE;
          }
       }
}
//execproc()
//takes care of execing the processes.
int execproc(char*** newargv, char** line, int* REDSO, int* REDSI, int* fdr, voi
d** x,FILE** input){
    pid_t childpid;
    childpid = fork();
       if(childpid == −1){
         perror("Fork Failed\n");
         return;
       }
       else if(childpid == 0){
          execvp((*newargv)[0], (*newargv));
          perror("Child failed to execvp the command");
          exit(1);
       }
       else if (childpid > 0) {
            if (childpid != wait(NULL)) {
               perror("Parent failed to wait");
               return;
            }

            if((*REDSO)){
              freopen("/dev/tty", "w", stdout);
              (*REDSO) = FALSE;
            }
            if((*REDSI)){
              dup2((*fdr), STDIN_FILENO);
              close((*fdr));
              (*REDSI) = FALSE;
            }
            fprintf(stderr, "ZShell $: ");
            (*x) = fgets ((*line), MAX_CANON, (*input));
            if((*x)) {
               if((strlen((*line))−1)!=0){
                (*line)[(strlen((*line)))−1] = ’\0’;
               }
            }
        }
}
```