```c
/*******************************************
*   Project 2
*
*   Sean Crolwey
*
*   This program creates a tree from an infix
*   equation and creates seperate processes
*   to calculate the equation
*
*******************************************/

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

#define MAXINPUT 50
#define READ 0
#define WRITE 1

int equate(char **tokens, int numTokens);

int main(){
    //Stored user input
    char *equation = malloc(MAXINPUT);

    //Breaks user input into tokens
    char **tokens = malloc(MAXINPUT *       (char*));

    //Number of tokens
    int numTokens = 0;

    //Get user input and remove new line
    fgets(equation, MAXINPUT, stdin);
    strtok(equation, "\n");

    //Breaks user input into seperate tokens and keeps track of total tokens
    equation = strtok(equation, " ");
         (equation != NULL){
        tokens[numTokens] = equation;
        numTokens++;
        equation = strtok(NULL, " ");
    }

    //Solves the equation
    equate(tokens, numTokens);

           0;
}

int equate(char **tokens, int numTokens){
    //Equation elements
    int left = 0;
    int right = 0;
    char op;
    int answer = 0;

    //Iterator
    int i = 0;

    //Piping
    int fd[2];

    //Process elements
    pid_t pid;
```

```
    int status;

    //Loop boolean
    int loop = 1;

    //Stores answers into file
    FILE* file = fopen("answers", "a+");

    //My goal for this was to solve a 2 operator equation
         (loop){

        //Checks the operator type
          (*tokens[i] == '+' || *tokens[i] == '-' || *tokens[i] == '*' || *tokens[i] == '/'){

            //Creates pipe
               (pipe (fd) < 0) {
                perror ("plumbing problem");
                exit(1);
            }

            //Spawns a process
               ((pid = fork()) < 0) {
                perror("fork failure");
                exit(1);

            //Child Process
            }        (pid == 0) {
                //Pipes to a file
                dup2 (fileno(file), fileno(stdout));

                //Stores input into equation elements
                sscanf(tokens[i-1], "%d", &left);
                op = *tokens[i];

                printf("PID : %i\t PPID : %i\t%i\t%i\t%c\n", getpid(), getppid(), left, right, op);

                //Checks if at the end of the equation
                  (tokens[i+2] == 0){
                    //If at the end then stores the next number into right
                    sscanf(tokens[i+1], "%d", &right);

                    //Equates
                      (op == '+')
                        answer = left + right;
                           (op == '-')
                        answer = left - right;
                           (op == '*')
                        answer = left * right;
                           (op == '/')
                        answer = left / right;

                    printf("%i\n", answer);

                    exit(0);
                }    {
                    //Set to 3 to start at the second operator
                    i = 3;

                    //Creates pipe
                       (pipe (fd) < 0) {
                        perror ("plumbing problem");
                        exit(1);
                    }

                    //Spawns a process
                       ((pid = fork()) < 0) {
                        perror("fork failure");
```

```
                        exit(1);

                    //Child process
                    }        (pid == 0) {
                        //Since at the end of the equation it stores all elements including right
                        sscanf(tokens[i-1], "%d", &left);
                        op = *tokens[i];
                        sscanf(tokens[i+1], "%d", &right);

                        printf("PID : %i\t PPID : %i\t%i\t%i\t%c\n", getpid(), getppid(), left, right,
op);


                        //Equates
                          (op == '+')
                            answer = left + right;
                               (op == '-')
                            answer = left - right;
                               (op == '*')
                            answer = left * right;
                               (op == '/')
                            answer = left / right;

                        printf("%i\n", answer);

                        exit(0);

                    }    {
                        //Waits on child
                        wait(&status);
                    }
                }
            }    {
                //Waits on child
                wait(&status);
            }
        }

        //Increments
        i++;

        //Prevents from overlooping
          (i==numTokens){
            loop = 0;
        }
    }

           0;
}
```