Shell Project

```
[arun@archlinux CS365]$ gcc shellProj.c -o prog
[arun@archlinux CS365]$ ./prog
Evgeny's shell: ls -l -a|nl|nl|nl|nl|nl
     1       1       1       1       1  total 100
     2       2       2       2       2  drwxr-xr-x 3 arun users  4096 Oct 27 17:25 .
     3       3       3       3       3  drwxr-xr-x 3 arun users  4096 Oct 27 11:51 ..
     4       4       4       4       4  drwxr-xr-x 2 arun users  4096 Oct 20 14:16 .vscode
     5       5       5       5       5  -rw-r--r-- 1 arun users   628 Oct 20 14:18 ogram.c
     6       6       6       6       6  -rwxr-xr-x 1 arun users 21592 Oct 27 17:25 prog
     7       7       7       7       7  -rwxr-xr-x 1 arun users 24880 Oct 27 17:23 shellProj
     8       8       8       8       8  -rw-r--r-- 1 arun users  3577 Oct 27 17:23 shellProj.c
     9       9       9       9       9  -rw-r--r-- 1 arun users     0 Oct 20 14:20 week4
    10      10      10      10      10  -rwxr-xr-x 1 arun users 24760 Oct 27 14:22 wk4day2
[arun@archlinux CS365]$
```

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


struct Pipes{
    int fd[2];
};
typedef struct Pipes Pipes;

const int READ = 0;
const int WRITE = 1;

size_t Split(const char *command, const char *delimeter, char **commands){
    size_t command_count = 0;

    char *command_copy = strdup(command);

    char *token = strtok(command_copy, delimeter); //primer

    while (token != NULL){
        commands[command_count] = malloc(sizeof(char) * strlen(token));
        strcpy(commands[command_count++], token);

        token = strtok(NULL, delimeter);
```

```
    }
    commands[command_count] = NULL;
    return command_count;
}

void eggXecute(char **command, Pipes *pipes, const size_t
current_pipe_index, const int command_count){
    // ["ls",-l", "-a", NULL]
    pid_t pid = fork();

    if(pid == 0){ //child
        // 1. first command
        if(current_pipe_index == 0){
            close(pipes[current_pipe_index].fd[READ]);
            //set up pipe
            if(dup2(pipes[current_pipe_index].fd[WRITE], STDOUT_FILENO) <
0){
                char msg[256];
                strcat("Failed to redirect for Command: ", command[0]);
                perror(msg);
                exit(EXIT_FAILURE);
            }
        }
        // 2. last command
        else if(current_pipe_index == command_count -1){
            close(pipes[current_pipe_index - 1].fd[WRITE]);
            //set up pipe
            if(dup2(pipes[current_pipe_index - 1].fd[READ], STDIN_FILENO) <
0){
                char msg[256];
                strcat("Failed to redirect for Command: ", command[0]);
                perror(msg);
                exit(EXIT_FAILURE);
            }

        }
        // 3. middle command
        else{

            //set up pipe
```

```
            close(pipes[current_pipe_index - 1].fd[WRITE]);
            if(dup2(pipes[current_pipe_index - 1].fd[READ], STDIN_FILENO) <
0){
                char msg[256];
                strcat("Failed to redirect for Command: ", command[0]);
                perror(msg);
                exit(EXIT_FAILURE);
            }
            close(pipes[current_pipe_index].fd[READ]);
            if(dup2(pipes[current_pipe_index].fd[WRITE], STDOUT_FILENO) <
0){
                char msg[256];
                strcat("Failed to redirect for Command: ", command[0]);
                perror(msg);
                exit(EXIT_FAILURE);
            }
        }
        execvp(command[0], command);
    }
    else{//parent
        if( current_pipe_index != 0 )
            close(pipes[current_pipe_index - 1].fd[WRITE]);

        wait(NULL);
    }



}
void printPar(size_t cmdCount, char **commands){
    for(size_t i = 0; i < cmdCount; ++i){
        printf("%s\n", commands[i]);
    }

}
int main(){
    char *commands[256];

    char simple_command[256];
    const char *PIPE = "|";
    const char *SPACE = " ";
```

```
    const char *redirect = ">";

    printf("Evgeny's shell: ");
    scanf("%[^\n]", simple_command);
    size_t command_count = Split(simple_command, PIPE, commands);

    if(command_count == 1){
        char *single_command[256];
        size_t single_command_count = Split(commands[0], SPACE,
single_command);
        execvp(single_command[0], single_command);
    }
    else{
        Pipes *pipes = malloc(sizeof(Pipes) * command_count - 1);
        for(size_t i = 0; i < command_count - 1; ++i){
            pipe(pipes[i].fd);
        }

        for(size_t i = 0; i < command_count; ++i){
            char *single_command[256];
            size_t single_command_count = Split(commands[i], SPACE,
single_command);
            single_command[single_command_count] = NULL;
            eggXecute(single_command, pipes, i, command_count);
        }
    }



    return 0;
}
```