E:\shell.c

```c
// C Program to design a shell in Linux
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<readline/readline.h>
#include<readline/history.h>

#define MAXCOM 1000 // max number of letters to be supported
#define MAXLIST 100 // max number of commands to be supported

// Clearing the shell using escape sequences
#define clear() printf("\033[H\033[J")

// Greeting shell during startup
void init_shell()
{
    clear();
    printf("\n\n\n\n******************"
        "************************");
    printf("\n\n\n\t****MY SHELL****");
    printf("\n\n\t-USE AT YOUR OWN RISK-");
    printf("\n\n\n\n******************"
        "************************");
    char* username = getenv("USER");
    printf("\n\n\nUSER is: @%s", username);
    printf("\n");
    sleep(1);
    clear();
}

// Function to take input
int takeInput(char* str)
{
    char* buf;

    buf = readline("\n>>> ");
    if (strlen(buf) != 0) {
        add_history(buf);
        strcpy(str, buf);
        return 0;
    } else {
        return 1;
    }
}

// Function to print Current Directory.
void printDir()
{
    char cwd[1024];
    getcwd(cwd, sizeof(cwd));
    printf("\nDir: %s", cwd);
```

```c
}

// Function where the system command is executed
void execArgs(char** parsed)
{
    // Forking a child
    pid_t pid = fork();

    if (pid == -1) {
        printf("\nFailed forking child..");
        return;
    } else if (pid == 0) {
        if (execvp(parsed[0], parsed) < 0) {
            printf("\nCould not execute command..");
        }
        exit(0);
    } else {
        // waiting for child to terminate
        wait(NULL);
        return;
    }
}

// Function where the piped system commands is executed
void execArgsPiped(char** parsed, char** parsedpipe)
{
    // 0 is read end, 1 is write end
    int pipefd[2];
    pid_t p1, p2;

    if (pipe(pipefd) < 0) {
        printf("\nPipe could not be initialized");
        return;
    }
    p1 = fork();
    if (p1 < 0) {
        printf("\nCould not fork");
        return;
    }

    if (p1 == 0) {
        // Child 1 executing..
        // It only needs to write at the write end
        close(pipefd[0]);
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[1]);

        if (execvp(parsed[0], parsed) < 0) {
            printf("\nCould not execute command 1..");
            exit(0);
        }
    } else {
        // Parent executing
        p2 = fork();

        if (p2 < 0) {
```

```
            printf("\nCould not fork");
            return;
        }

        // Child 2 executing..
        // It only needs to read at the read end
        if (p2 == 0) {
            close(pipefd[1]);
            dup2(pipefd[0], STDIN_FILENO);
            close(pipefd[0]);
            if (execvp(parsedpipe[0], parsedpipe) < 0) {
                printf("\nCould not execute command 2..");
                exit(0);
            }
        } else {
            // parent executing, waiting for two children
            wait(NULL);
            wait(NULL);
        }
    }
}

// Help command builtin
void openHelp()
{
    puts("\n***WELCOME TO MY SHELL HELP***"
        "\nCopyright @ Suprotik Dey"
        "\n-Use the shell at your own risk..."
        "\nList of Commands supported:"
        "\n>cd"
        "\n>ls"
        "\n>exit"
        "\n>all other general commands available in UNIX shell"
        "\n>pipe handling"
        "\n>improper space handling");

    return;
}

// Function to execute builtin commands
int ownCmdHandler(char** parsed)
{
    int NoOfOwnCmds = 4, i, switchOwnArg = 0;
    char* ListOfOwnCmds[NoOfOwnCmds];
    char* username;

    ListOfOwnCmds[0] = "exit";
    ListOfOwnCmds[1] = "cd";
    ListOfOwnCmds[2] = "help";
    ListOfOwnCmds[3] = "hello";

    for (i = 0; i < NoOfOwnCmds; i++) {
        if (strcmp(parsed[0], ListOfOwnCmds[i]) == 0) {
            switchOwnArg = i + 1;
            break;
        }
```

```
167     }
168
169     switch (switchOwnArg) {
170     case 1:
171         printf("\nGoodbye\n");
172         exit(0);
173     case 2:
174         chdir(parsed[1]);
175         return 1;
176     case 3:
177         openHelp();
178         return 1;
179     case 4:
180         username = getenv("USER");
181         printf("\nHello %s.\nMind that this is "
182             "not a place to play around."
183             "\nUse help to know more..\n",
184             username);
185         return 1;
186     default:
187         break;
188     }
189
190     return 0;
191 }
192
193 // function for finding pipe
194 int parsePipe(char* str, char** strpiped)
195 {
196     int i;
197     for (i = 0; i < 2; i++) {
198         strpiped[i] = strsep(&str, "|");
199         if (strpiped[i] == NULL)
200             break;
201     }
202
203     if (strpiped[1] == NULL)
204         return 0; // returns zero if no pipe is found.
205     else {
206         return 1;
207     }
208 }
209
210 // function for parsing command words
211 void parseSpace(char* str, char** parsed)
212 {
213     int i;
214
215     for (i = 0; i < MAXLIST; i++) {
216         parsed[i] = strsep(&str, " ");
217
218         if (parsed[i] == NULL)
219             break;
220         if (strlen(parsed[i]) == 0)
221             i--;
222     }
```

```
}

int processString(char* str, char** parsed, char** parsedpipe)
{

    char* strpiped[2];
    int piped = 0;

    piped = parsePipe(str, strpiped);

    if (piped) {
        parseSpace(strpiped[0], parsed);
        parseSpace(strpiped[1], parsedpipe);

    } else {

        parseSpace(str, parsed);
    }

    if (ownCmdHandler(parsed))
        return 0;
    else
        return 1 + piped;
}

int main()
{
    char inputString[MAXCOM], *parsedArgs[MAXLIST];
    char* parsedArgsPiped[MAXLIST];
    int execFlag = 0;
    init_shell();

    while (1) {
        // print shell line
        printDir();
        // take input
        if (takeInput(inputString))
            continue;
        // process
        execFlag = processString(inputString,
        parsedArgs, parsedArgsPiped);
        // execflag returns zero if there is no command
        // or it is a builtin command,
        // 1 if it is a simple command
        // 2 if it is including a pipe.

        // execute
        if (execFlag == 1)
            execArgs(parsedArgs);

        if (execFlag == 2)
            execArgsPiped(parsedArgs, parsedArgsPiped);
    }
    return 0;
}

```