```
// 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
#define MAX_LINE 80 /* The maximum length of a command */
#define BUFFER_SIZE 50
#define buffer "\nShell Command History:\n"
// Clearing the shell using escape sequences
#define clear() printf("\033[H\033[J")
//declarations
char history[10][BUFFER_SIZE]; //history array to store history commands
int count = 0;
// Greeting shell during startup
void 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("\nuab_sh >> ");
        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);
}

// 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 Help()
{
        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>hello"
                "\n>fibonacci sequence"
        "\n>histoty");

        return;
}

void fibo(){
        int i, num;
        int t1 = 0, t2 = 1;
        printf("How many elements you want to display: ");
        scanf("%d", &num);
        if(num!=0){
        int nextTerm = t1 + t2;
        printf("The first %d value: %d %d ", num, t1, t2);
        for (i = 3; i <= num; ++i) {
                printf("%d ", nextTerm);
                t1 = t2;
                t2 = nextTerm;
                nextTerm = t1 + t2;
                }
        printf("\n");
        }

}
/*
void history(){
//      printf("\nHISORY\n");
}
*/
//function to display the history of commands
void my_history()
{

    printf("Shell command history:\n");

    int i;
    int j = 0;
    int histCount = count;

    //loop for iterating through commands
    for (i = 0; i<10;i++)
    {
```

```
        //command index
        printf("%d.  ", histCount);
        while (history[i][j] != '\n' && history[i][j] != '\0')
        {
                //printing command
            printf("%c", history[i][j]);
            j++;
        }
        printf("\n");
        j = 0;
        histCount--;
        if (histCount ==  0)
            break;
    }
    printf("\n");
}
//Fuction to get the command from shell, tokenize it and set the args parameter

int formatCommand(char inputBuffer[], char *args[],int *flag)
{
        int length; // # of chars in command line
        int i;     // loop index for inputBuffer
        int start;  // index of beginning of next command
        int ct = 0; // index of where to place the next parameter into args[]
        int hist;
        //read user input on command line and checking whether the command is !! or !n

        length = read(STDIN_FILENO, inputBuffer, MAX_LINE);


    start = -1;
    if (length == 0)
        exit(0);   //end of command
    if (length < 0)
    {
        printf("Command not read\n");
        exit(-1);  //terminate
    }

   //examine each character
    for (i=0;i<length;i++)
    {
        switch (inputBuffer[i])
        {
            case ' ':
            case '\t' :               // to seperate arguments
                if(start != -1)
                {
                    args[ct] = &inputBuffer[start];
                    ct++;
                }
                inputBuffer[i] = '\0'; // add a null char at the end
                start = -1;
                break;

            case '\n':                 //final char
                if (start != -1)
                {
                    args[ct] = &inputBuffer[start];
                    ct++;
                }
                inputBuffer[i] = '\0';
                args[ct] = NULL; // no more args
```

```
                break;

            default :
                if (start == -1)
                    start = i;
                if (inputBuffer[i] == '&')
                {
                    *flag  = 1; //this flag is the differentiate whether the child proc
ess is invoked in background
                    inputBuffer[i] = '\0';
                }
        }
    }

    args[ct] = NULL; //if the input line was > 80

if(strcmp(args[0],"history")==0)
        {
               if(count>0)
                {

                my_history();
                }
                else
                {
                printf("\nNo Commands in the history\n");
                }
                return -1;
        }

        else if (args[0][0]-'!' ==0)
        {       int x = args[0][1]- '0';
                int z = args[0][2]- '0';

                if(x>count) //second letter check
                {
                printf("\nNo Such Command in the history\n");
                strcpy(inputBuffer,"Wrong command");
                }
                else if (z!=-48) //third letter check
                {
                printf("\nNo Such Command in the history. Enter <=!9 (buffer size is 10
 along with current command)\n");
                strcpy(inputBuffer,"Wrong command");
                }
                else
                {

                        if(x==-15)//Checking for '!!',ascii value of '!' is 33.
                        {        strcpy(inputBuffer,history[0]);  // this will be your
10 th(last) command
                        }
                        else if(x==0) //Checking for '!0'
                        {        printf("Enter proper command");
                                strcpy(inputBuffer,"Wrong command");
                        }

                        else if(x>=1) //Checking for '!n', n >=1
                        {
                                strcpy(inputBuffer,history[count-x]);

                        }
```

```
                }
        }
 for (i = 9;i>0; i--) //Moving the history elements one step higher
        strcpy(history[i], history[i-1]);

    strcpy(history[0],inputBuffer); //Updating the history array with input buffer
    count++;
        if(count>10)
        { count=10;
        }
}

// Function to execute builtin commands
int ownCmdHandler(char** parsed)
{
        int NoOfOwnCmds = 5, i, switchOwnArg = 0;
        char* ListOfOwnCmds[NoOfOwnCmds];
        char* username;
        //char **history = malloc(sizeof(char) * bufsize);

        ListOfOwnCmds[0] = "exit";
        ListOfOwnCmds[1] = "cd";
        ListOfOwnCmds[2] = "help";
        ListOfOwnCmds[3] = "hello";
        ListOfOwnCmds[4] = "fibonacci";
        ListOfOwnCmds[5] = "history";

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

        switch (switchOwnArg) {
        case 1:
                printf("\nQuit\n");
                exit(0);
        case 2:
                chdir(parsed[1]);
                return 1;
        case 3:
                Help();
                return 1;
        case 4:
                printf("\nHello World!\n");
                return 1;
        case 5:
                fibo();
                return 1;
        case 6:
                my_history();
                return 1;

        default:
                break;
        }

        return 0;
}

// function for finding pipe
int parsePipe(char* str, char** strpiped)
```

```
{
        int i;
        for (i = 0; i < 2; i++) {
                strpiped[i] = strsep(&str, "|");
                if (strpiped[i] == NULL)
                        break;
        }

        if (strpiped[1] == NULL)
                return 0; // returns zero if no pipe is found.
        else {
                return 1;
        }
}

// function for parsing command words
void parseSpace(char* str, char** parsed)
{
        int i;

        for (i = 0; i < MAXLIST; i++) {
                parsed[i] = strsep(&str, " ");

                if (parsed[i] == NULL)
                        break;
                if (strlen(parsed[i]) == 0)
                        i--;
        }
}

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;
        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);
        }
    {
    char inputBuffer[MAX_LINE]; /* buffer to hold the input command */
    int flag; // equals 1 if a command is followed by "&"
    char *args[MAX_LINE/2 + 1];/* max arguments */
    int should_run =1;

    pid_t pid,tpid;
    int i;


    while (should_run) //infinite loop for shell prompt
    {
        flag = 0; //flag =0 by default
        printf("osh>");
        fflush(stdout);
        if(-1!=formatCommand(inputBuffer,args,&flag)) // get next command
        {
                pid = fork();

                if (pid < 0)//if pid is less than 0, forking fails
                {

                        printf("Fork failed.\n");
                        exit (1);
                }

                 else if (pid == 0)//if pid ==0
                {

                        //command not executed
                        if (execvp(args[0], args) == -1)
                        {

                                printf("Error executing command\n");
                        }
                 }

                 // if flag == 0, the parent will wait,
                // otherwise returns to the formatCommand() function.
                else
                {
                        i++;
                        if (flag == 0)
                         {
                                i++;
```

```
                                wait(NULL);
                            }
                        }
                    }
                }
            }

        return 0;
}
```