```
#include <stdio.h>
#include <string.h>
#include "fork.h"

#define MAX_COMMAND_LENGTH 128

void printEntryPoint()
{
    printf("262$");
    fflush(stdout);
}

void freeLinkedList(Command *head)
{
    // Traverse the linked list of commands and free the memory allocated for
each Command struct
    Command *current = head;
    while (current != NULL)
    {
        Command *temp = current;
        for (int i = 0; i < temp->argumentslength; i++)
        {
            if (temp->arguments[i] != NULL)
            {
                free(temp->arguments[i]);
                // set to null
                temp->arguments[i] = NULL;
            }
        }

        if (temp->arguments != NULL)
        {
            free(temp->arguments);
            // set to null
            temp->arguments = NULL;
        }

        current = current->next;

        if (temp != NULL)
        {
            free(temp);
            // set to null
            temp = NULL;
        }
    }
}

Command *executeHistory(Command *head, Command *tail, int index, int
commandIndex)
{
    // Find the command with the specified index
    Command *current = head;
    int i = 0;
    while (current != NULL && i < index)
    {
        current = current->next;
        i++;
    }

    // If the command is found, execute it
    if (current != NULL)
    {
        // printf("Executing command: %s\n", current->command);
        int status = call_exe(current);

        // printf("%d: ", current->index);
        // for (int i = 0; i < current->argumentslength; i++)
        // {
        //     printf("%s ", current->arguments[i]);
        // }
        // printf("\n");

        // If the command was successful, add it to the history
        if (status == 0)
        {
            Command *cmd = (Command *)malloc(sizeof(Command));
            cmd->command = current->command;
            cmd->arguments = (char **)malloc(MAX_COMMAND_LENGTH * sizeof(char
*));

            // copy arguments
            for (int i = 0; i < current->argumentslength; i++)
            {
                cmd->arguments[i] = (char *)malloc(MAX_COMMAND_LENGTH *
sizeof(char));
                strcpy(cmd->arguments[i], current->arguments[i]);
            }

            cmd->next = NULL;
            cmd->index = commandIndex;
            cmd->argumentslength = current->argumentslength;

            // set tail to point to the new command
            tail->next = cmd;
            tail = cmd;

            return tail;
        }
        else
        {
            fprintf(stderr, "error: %s\n", strerror(status));
        }
    }
    else
    {
        printf("Command not found in history\n");
    }
    return tail;
}

void printSeriesCommand(Command *current)
{
    while (current != NULL)
    {
        printf("%d: ", current->index /*current-> command*/);
        for (int i = 0; i < current->argumentslength; i++)
        {
            // if last argument, print without space
            if (i == current->argumentslength - 1)
            {
                printf("%s", current->arguments[i]);
            }
            else
            {
                printf("%s ", current->arguments[i]);
            }
        }
        printf("\n");
        current = current->next;
    }
}

int main()
{
    char input[15000];
    Command *head = NULL;
    Command *tail = NULL;
    int commandIndex = 0;
    while (1)
    {

        char *token = NULL;
        char **args = (char **)malloc(MAX_COMMAND_LENGTH * sizeof(char *));

        printEntryPoint();

        // Read the user's input and check if fgets reached the end of the file
        if (fgets(input, 15000, stdin) == NULL)
        {
            // free linked list
            freeLinkedList(head);

            if (token != NULL)
            {
                free(token);
                token = NULL;
            }

            if (args != NULL)
            {
                free(args);
                args = NULL;
            }

            break;
        }

        // Remove the newline character at the end of the input
        input[strcspn(input, "\n")] = '\0';
        input[strcspn(input, "\r")] = '\0';
        input[strcspn(input, "\t")] = '\0';

        // if last char is a space, remove it
        if (input[strlen(input) - 1] == ' ')
        {
            input[strlen(input) - 1] = '\0';
        }

        // Check if the user typed the exit command
        if (strcmp(input, "exit") == 0)
        {
            // free linked list
            freeLinkedList(head);

            if (token != NULL)
            {
                free(token);
                token = NULL;
            }

            if (args != NULL)
            {
                free(args);
                args = NULL;
            }
            break;
        }

        // check if input was just all whitespace
        // strip white space from the beginning of the input
        char *p = input;
        while (*p == ' ')
        {
            p++;
        }

        // check if p is empty
        if (*p == '\0' || *p == '\n' || *p == '\r')
        {
            // free linked args
            if (args != NULL)
            {
                free(args);
                args = NULL;
            }

            if (token != NULL)
            {
                free(token);
                token = NULL;
            }
            continue;
        }

        // check if input was just all whitespace

        if (strcmp(input, "history -c") == 0)
        {
            // Traverse the linked list of commands and free the memory
allocated for each Command struct

            Command *current = head;
            while (current != NULL)
            {
                Command *temp = current;
                for (int i = 0; i < temp->argumentslength; i++)
                {
                    if (temp->arguments[i] != NULL)
                    {
                        free(temp->arguments[i]);
                        // set to null
                        temp->arguments[i] = NULL;
                    }
                }

                if (temp->arguments != NULL)
                {
                    free(temp->arguments);
                    // set to null
                    temp->arguments = NULL;
                }

                current = current->next;

                if (temp != NULL)
                {
                    free(temp);
                    // set to null
                    temp = NULL;
                }
            }

            head = NULL;
            tail = NULL;

            commandIndex = 0;


            // free linked args
            if (args != NULL)
            {
                free(args);
                args = NULL;
            }

            if (token != NULL)
            {
                free(token);
                token = NULL;
            }
            continue;
        }

        // Parse the user's input to extract the command and its arguments

        int i = 0;
        token = strtok(input, " ");
        while (token != NULL)
        {
            args[i] = malloc(strlen(token) * sizeof(char));
            strcpy(args[i], token);
            i++;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;

        // if number of args more than 127 then print error
        if (i > MAX_COMMAND_LENGTH)
        {

            // loop through args and free each arg
            for (int j = 0; j < i; j++)
            {
                if (args[j] != NULL)
                {
                    free(args[j]);
                    args[j] = NULL;
                }
            }
            // free linked args
            if (args != NULL)
            {
                free(args);
                args = NULL;
            }

            if (token != NULL)
            {
                free(token);
```

```
                token = NULL;
            }
            printf("error: too many arguments\n");
            continue;
        }

        // if len args = 2; then check if args[0] = history and args[1] is a
number
        if (i == 2)
        {
            if (strcmp(args[0], "history") == 0 && (atoi(args[1]) != 0 ||
strcmp(args[1], "0") == 0))
            {
                // if args[1] is 0 then index = 0
                int index = atoi(args[1]);
                if (strcmp(args[1], "0") == 0)
                {
                    index = 0;
                }
                tail = executeHistory(head, tail, index, commandIndex++);

                // loop through args and free each arg
                for (int j = 0; j < i; j++)
                {
                    if (args[j] != NULL)
                    {
                        free(args[j]);
                        args[j] = NULL;
                    }
                }

                // free linked args
                if (args != NULL)
                {
                    free(args);
                    args = NULL;
                }

                if (token != NULL)
                {
                    free(token);
                    token = NULL;
                }

                continue;
            }
        }

        if (strcmp(input, "history") == 0)
        {

            printSeriesCommand(head);

            // loop through args and free each arg
            for (int j = 0; j < i; j++)
            {
                if (args[j] != NULL)
                {
                    free(args[j]);
                    args[j] = NULL;
                }
            }

            // free linked args
            if (args != NULL)
            {
                free(args);
                args = NULL;
            }

            if (token != NULL)
            {
                free(token);
                token = NULL;
            }
            continue;
        }

        // Create a new Command struct and fill in its fields
        Command *cmd = (Command *)malloc(sizeof(Command));
        cmd->command = args[0];
        cmd->arguments = args;
        cmd->next = NULL;
        cmd->index = commandIndex++;
        cmd->argumentslength = i;

        if (head == NULL)
        {
            // printf("Head is null. Adding %s as head\n", cmd->command);
            //  That is this is the first command ever. So this becomes the
HEAD.
            //  We will never modify the head again as that is where we start
our
            //  history.
            head = cmd;
            tail = cmd;
        }
        else
        {
            // printf("Head is %s; tail is %s\n", head->command, tail->command);
            //  This is not the first command and we have a head and a tail. We
            //  will be pointing the tail-> next to current cmd. Make current
cmd
            //  our new tail.
            // printf("Adding %s as next of %s\n", cmd->command, tail->command);
            tail->next = cmd;
            tail = cmd;
        }

        if (strcmp(cmd->command, "cd") == 0)
        {
            // if the command is cd, we need to change the directory of the
shell process
            int status = chdir(cmd->arguments[1]);
            if (status != 0)
            {
                fprintf(stderr, "error: %s\n", strerror(errno));
            }
            continue;
        }

        else
        {

            int status = call_exe(cmd);

            // if the command failed to execute, print out error message
            if (status != 0)
            {
                fprintf(stderr, "error: %s\n", strerror(status));
            }
        }
    }

    return 0;
}
```