Report on Command-Line Interface (CLI) Shell Implementation in C

1 Introduction

This C program implements a simple command-line interface (CLI) shell for Windows, similar to popular Unix/Linux shells such as Bash. The shell processes user input, manages command history, changes directories, lists files, and performs basic file system operations. The primary objective of this shell is to allow users to execute basic commands (cd, ls, pwd, echo, hist) and interact with the Windows operating system through a terminal interface.

2 Program Structure and Components

2.1 Libraries and Macros

The program includes several standard libraries:

- `<stdio.h>`: Standard input/output functions.
- `<stdlib.h>`: Standard library functions like `getenv()` and `strtok()`.
- `<string.h>`: Functions for string manipulation (`strncpy()`, `strcmp()`).
- `<windows.h>`: Windows-specific system calls (`GetComputerName()`, `FindFirstFile()`).
- `<direct.h>`: For directory-related functions like `_getcwd()` and `_chdir()`.
- `<limits.h>`: For `PATH_MAX` (maximum allowable path length).

Macros:

- `MAX_INPUT`: Defines the maximum size for input (1024 characters).
- `MAX_ARGS`: Sets the maximum number of command arguments (64).
- `MAX_HISTORY`: Sets the size of the command history buffer (10).

2.2 Command History Management

The program maintains a history of the last 10 commands executed by the user.

Global Variables:

- `history[MAX_HISTORY][MAX_INPUT]`: Stores the history of the last 10 commands.
- `history_count`: Tracks the current number of commands in the history.

Functions:

- `add_to_history(const char *command)`: Adds the latest command to the history. If the history buffer is full, the oldest command is removed, and the new command is added.
- `show_history()`: Displays the command history to the user.

2.3 Directory Navigation

The program provides functionality to change and print the current directory.

Functions:

- `change_directory(char *path, char *prev_dir)`: Changes the current working directory. If no path is provided or the user specifies "∼", it defaults to the user's home directory. If the user enters "-", the program switches to the previous directory.
- `print_working_directory()`: Prints the current working directory using `_getcwd()`.

2.4 File Listing

`list_files()`: Lists the files in the current directory using Windows API functions. It leverages `FindFirstFile()` and `FindNextFile()` to traverse files and directories in the current location.

2.5 Command Parsing and Execution

The shell reads user input, tokenizes the command, and executes appropriate actions.

Input Parsing:

- The shell continuously waits for user input via `fgets()`.
- The input is split into tokens using `strtok()`, and each token is stored in the `args` array.
- Commands are matched using `strcmp()` and executed based on a set of predefined commands.

Supported Commands:

- `cd`: Changes the current directory. Handles special cases like `cd -` (switch to the previous directory) and `cd` ~ (switch to the home directory).
- `ls`: Lists files in the current directory.
- `pwd`: Prints the current working directory.
- `echo`: Prints the provided arguments to the terminal.
- `hist`: Displays the command history.
- `exit`: Exits the shell program.

2.6 Command Execution Workflow

Prompt Display: The shell prints a prompt in the format [username@hostname cwd]$, where:

- username: Retrieved from the environment variable USERNAME.
- hostname: Obtained using the Windows function `GetComputerName()`.
- cwd: The current working directory, fetched using `_getcwd()`.

Input Handling:

- The shell reads user input and processes it with `fgets()`.
- If the command is not empty, it is added to the history using `add_to_history()`.

Command Execution:

- The shell tokenizes the input into commands and arguments.
- It checks for each supported command (`cd`, `ls`, `pwd`, `echo`, etc.) and invokes the corresponding function.
- If an unsupported command is entered, the shell prints an error message.

Looping: The program continuously loops until the user types `exit`.

3 Error Handling

The shell performs basic error handling:

- If `getcwd()` or `chdir()` fails, it prints an appropriate error message.
- If too many arguments are passed to the `cd` command, the program prints an error and ignores the command.
- The `FindFirstFile()` function handles cases where the directory cannot be read.

4 Key Functions

`change_directory(char *path, char *prev_dir)`: Handles changing directories and supports navigating to the home directory (∼) and switching to the previous directory (-).

`list_files()`: Uses Windows API functions to list files and directories in the current location.

`echo_command(char *input)`: This function prints the user's input to the terminal, implementing an echo command.

5 Conclusion

This program demonstrates how a simple command-line shell can be implemented in C on a Windows system. It includes key features like command history, directory navigation, file listing, and the ability to execute basic commands. The use of Windows API functions enables interaction with the file system, while standard C functions handle input/output and string manipulation.

The shell could be further extended by adding support for more commands and features like piping, file redirection, or job control to make it more powerful.