1 Input Reading

In the `main` function, the code initializes the last executed command to `null` and enters a while loop for the shell. It then performs the following steps:

1. Initializes variables needed for parsing tokens.
2. Calls `printPrompt` function to display a user prompt.
3. Retrieves the input line from the user and checks for an empty line.
4. Initializes variables for background processes and creates copies of the command line.
5. Tokenizes the input and checks for specific commands (`exit`, `alias`).

2 Alias Handling

If the token is `alias`, the code parses the line in alias format and creates a new alias using the `addAlias` function. This function checks for the existence of a file to store alias commands. If the file does not exist, it creates a file named `AliasFile` and writes the alias; otherwise, it appends the alias to the existing file. The last executed command becomes the alias if executed correctly.

3 Alias Lookup

If the token is not `exit` or `alias`, the code checks for an alias defined for the token using the `findAlias` function. This function searches for the token in the `AliasFile` and returns the corresponding command if found. If an alias is found, the command is added to `cmdLine`, and parsing continues with the new command line.

4 Command Execution

After parsing the first command line, memory is allocated for `argv` (tokens), and another while loop starts parsing a copy of `cmdLine`. The code checks for the `&` sign and sets the background flag accordingly. If the command is `bello`, the `displayUserInfo` function is called with the last executed command. If the command is not `bello`, the `executeCommand` function is called with `argv`, background, and `argc` parameters.

5 Command Execution Details

In the `executeCommand` function, the code initializes and checks for redirection operations. If `>` or `>>` is present, it sets the appropriate flags and `outputFile` variable, removing the output name and redirection operation sign from `argv`. The code handles the `PATH` for command execution, checks for accessibility, and sets the `successExecutablePath` variable. If the path is not accessible, an error message is printed.

5.1 Output Handling

The code creates a pipe for `>>>` operation, forks, and waits for the child to return if not in the background. In the child process (`pid = 0`), it checks the output file and flags, sets the correct file flags, and executes the command. If the `invert` flag is set, the parent process reads the result from the pipe, inverts it, and writes it to the file.

6 Conclusion

The main function returns to the shell loop and frees memory allocations at the end of the program execution.