C Sells C Shells by the C Shore

CMPE 322

1 Due Date

Due Date is **11.12.2023 Monday 23:59**

2 Description

In this project, you are going to implement your own shell, in which certain commands are modified. The functionalities of these commands are already available within the default shell of your Linux-based OS. However, you are required achieve desired capabilities in our own way.

The aim of this project is to develop a program to achieve the aforementioned objectives. You have to implement it either on a Mac OS or on Ubuntu since code that successfully compiles in other OS's is not guaranteed to be portable to Mac OS or Ubuntu. It is your responsibility to use/check in Ubuntu or Mac OS. Installation details of the specific GCC Compiler (C++ is not allowed) will be given in the proceeding parts.

3 GCC installation

3.1 Mac OS

Mac OS generally has a default GCC installed but if you type *gcc -version* in your terminal, you will see that the returned version is not of the GCC but Clang. That is a novelty added by Apple that we do not prefer in this project. In order to use a specific version of gcc, we need to install *homebrew*, a general-purpose package manager for Mac OS (kind of apt-get for Ubuntu).

1. You can use the following command (all-in-one-line) to install homebrew (Skip to Step 4 if you already have Homebrew installed):

   ```
   /bin/bash -c "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
   ```

2. After the installation, in order to use Homebrew, you need to add the brew executable path to your global PATH variable. Assuming you use zsh as your default shell, do

   ```
   sudo nano ~/.zshrc
   ```

   command to access your shell's initialization file. (Search and modify according to the shell of your preference).

3. Then, you should append the line

```
export PATH="$PATH:/opt/homebrew/bin"
```

to the end of the file. Press CTRL-X to exit, Y to overwrite and ENTER to accept. Now, if you restart your terminal, you can use the *brew* command.

4. At last, you can use

```
brew install gcc
```

command to install latest gcc.

5. Please check the version of the installed gcc (13.2.0) with the command:

```
gcc-13 --version
```

Note that if you want to use Apple's GCC, you should type "gcc" when compiling, but in order to use recently installed GCC, you should use "gcc-13" when compiling.

3.2 Ubuntu

By default, Ubuntu 22.04.3 LTS is shipped with GCC 11.4.0. You can check you current GCC version by using the command below:

```
gcc --version
```

If your GCC version is 11.4.0, then you are good to go. If not please refer to the instructions in

```
https://www.cyberithub.com/how-to-update-or-upgrade-gcc-to-
latest-version-on-ubuntu-debian/
```

to update your GCC version to 11.4.0.

4 Compile&Run Details

Depending on your design, whether it is a single file or multi-file program, your program will be compiled and it is executed as a new process by the terminal. A simple "make" command should be enough to compile your program. Please refer to the problem sessions about the "makefile". Your executable should be named "myshell" (without extension). To sum up, we should be able to compile and run your code by just entering the following commands:

```
make
./myshell
```

As your shell starts, the prompt should always be in the form: (the current user name on the system) (Watch out: Please do not provide the user name as hardcoded, because it will be tested on a different machine!) @ (the machine name of the current system/hostname), space, current directory, space, followed by "$---$", space. (Hint: you can check "whoami" command from ps). An example (with the echo command) is given below:

```
egemenisguder@egemenPC ~/Downloads --- echo "Bellooo!"
```

5 Implementation of Shell

Your shell program will be even simpler from the typical shell programs that exist on the system: the program waits for the command to be entered by the user, it is executed as soon as it is requested, and the program starts to wait for another command after the command is successfully completed. Your shell program will continue to run until the interrupt signal or the "exit" command is entered.

- Here are the **MUSTS** of your shell which describe the main tasks and behaviour of the shell:
  - Your shell must work in a fork-exec manner, every execution should consist of fork/exec operations.
  - You must search the executable according to the order in the PATH variable. You cannot "directly invoke" a command. You should apply the rules of PATH environment variable.
  - Your shell must not terminate with an error, if an error occurs within an execution or while parsing the input, please specify the cause of error ("Invalid argument", "Command not found", etc.). (No need for a return value of execution.)

- Here are the **CANS** and **SHOULDS** of your shell which describe the features of your shell aside from the main execution task:
  - Commands can have arguments (i.e. -c -rf..)
  - Shell can be terminated by the "exit" command.
  - Background processing (&), and redirection operators (">" and "> >") should exist and achieve the same task as in bash or zsh. No other shell-built-in operator/operation or command is needed.
  - Users should be able to create alias by entering the command this way:

    ```
    alias alias\_name = "command\_name command\_args"

    alias egemen = "gcc main.c -o executable.o"
    ```

    * Aliases will only have one command (and arguments if exist), and they must be saved and valid even after a restart of the shell.
  - A new re-redirection operator appears ("> > >") (no spaces). This operator will act exactly as ("> >"), but the order of all letters in the output will be invested! Please refer to the example below.

    ```
    echo "Hello" >>> egemen.txt

    In egemen.txt:

    "olleH"
    ```

  - The "bello" command should display eight information items about the user, respectively:
    1. Username
    2. Hostname
    3. Last Executed Command
    4. TTY
    5. Current Shell Name (not myshell)
    6. Home Location
    7. Current Time and Date
    8. Current number of proccesses being executed (just as everything in task manager)

- Here are your **RESTRICTIONS**:
  - You cannot call/run any other shell (bash, zsh etc.) to execute commands.

- Here are additional **INFO** to facilitate your implementation:
  - You do not need to implement any built-in shell commands or operations (such as bg, for-while loops etc.), execute a command only if it exists in path. No additional features (other than the ones in the **CANS** and **SHOULDS** section) are required.
  - There will always be at most one command in each input.

6 Submission Details

This project must be implemented individually. Place all code files and a PDF file of the report on your approach. This report should not exceed one page and contain details of your implementation approach, the difficulties that you have met, etc. Put all files into a folder named after your student ID (e.g. 2017400200). Zip the folder for submission and name the .zip file with the same name. Submit the .zip file through Moodle.

- Your submission should contain a makefile, a PDF report and the code files. (Zip them into one folder)
- No late submissions or mail submissions. The deadline is strict.
- A submission that cannot be compiled or without makefile is -30 points.
- Please comment on your code, as it will be graded.
- Questions about the project on Moodle forum is greatly appreciated, please don't hesitate to ask. Questions by mail will not be accepted.

7 Academic Honesty

Needless to say, honesty and trust are crucial to all of us. Cheating, plagiarism and collusion are serious offences, and they will result in an "-50" grade for the first time and, if repeated, an overall F grade and disciplinary action.

If you are not certain whether an action violates Academic Honesty please ask.,

5