Programming Assignment 1: Learning Linux Commands Matching Game and C++ Review

1 Learner Objectives

At the conclusion of this programming assignment, participants should be able to:

- Develop a small C++ program with templates
- List and define 30 popular Linux commands
- Construct a class List template
- Open, edit, parse, and close .csv files in C++
- Create and maintain a repository on Github

Discuss the advantages and disadvantages of applying a linked list and an array to store data in this assignment.

2 Prerequisites

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements and apply top-down design principles for a problem
- Design, implement, and test medium programs in an object-oriented language
- Edit, build, and run programs through a Linux environment
- Describe and implement a linked list

3 Overview & Requirements

Develop and implement an interactive text-based single-player matching game. The objective of the game is to match Linux commands to appropriate descriptions of those commands. If a command is matched, then the player earns 1 point. If the command is not matched, then the player loses a point. Yes, negative point totals are possible. The player selects the number of match questions at the beginning of the game. The game continue until the number is reached. Each player's profile may be saved.

Commands should be stored in a file called `commands.csv`, in the form

command, "description"

For example, the first row could be:
ls, "Short for lists; displays the file and directory names in the current working directory."

Please note that you should place "" around your description so that it is interpreted as one value in the .csv file. Each ***command, "description"*** pair should be placed on its own line in the file. Commands may be added and removed from the file through the program. At least the following 30 commands should be supported at the beginning of the game:

Command Name	Command Name	Command Name
pwd	ls	cd
mkdir	rmdir	rm
cp	mv	ssh
scp	man	g++
gcc	make	ps
kill	top	who
chmod	cat	alias
chown	df	grep
echo	exit	clear
find	finger	free

You are free to manually populate the `commands.csv` file to store the initial 30 commands. I suggest that you explore the commands at https://www.geeksforgeeks.org/linux-commands/ and some other sites to find the descriptions. Please list the base command description without any options, i.e., "ls" rather than "ls -l".

Player profiles must be stored in a file called `profiles.csv`. Each profile includes a name and current points. The format for a profile is:

name, points

3.1 What is required?

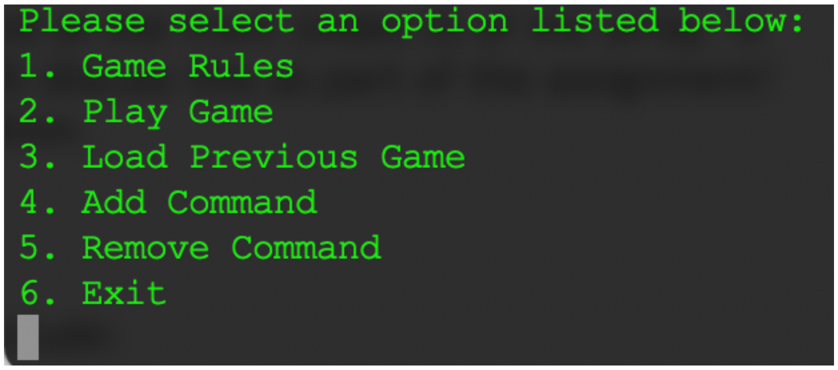
Upon startup of the program, the Linux command and description pairs from `commands.csv` must be loaded into a singly linked list that was constructed from a 2-parameter *template* (see "3.4.1 Design considerations" under "3.4 Play Game Requirement"). All commands may be inserted at the front of the list as they are read from the file. Also, the user name and points pairs from `profiles.csv` must be loaded into a *profiles* array. All user name and points pairs must also be loaded at the front (index 0) of the array. Is this a poor design decision? You will discuss this as part of the assignment! See the "3.9 Other Requirements" section.

3.2 Main Menu Requirement

Display a "main" menu. Options include:

1. Game Rules
2. Play Game
3. Load Previous Game
4. Add Command
5. Remove Command
6. Exit

After each option is complete, the program should return to the "main" menu. Error checking for user input is required. An example:



```
Please select an option listed below:
1. Game Rules
2. Play Game
3. Load Previous Game
4. Add Command
5. Remove Command
6. Exit
```

3.3 Game Rules Requirement

When option 1 is selected from the "main" menu, the user should be provided with the rules of the game. Once the action is complete, return to the "main" menu.

3.3.1 Design considerations

You need to summarize the rules for YOUR version of the game in your own words.

3.4 Play Game Requirement

When option 2 is selected from the “main” menu, initially prompt the user for a name. Next, prompt the user for how many commands/questions to generate for matching. The user should have the ability to run through 5 – 30 questions. Once the number of questions is selected, a random Linux command should be displayed, and 3 descriptions should be displayed along with it. One of those descriptions must be the correct description. The others are randomly selected from the other commands. The 3 descriptions should be listed in a random order. For example (the lines do not need to be displayed):

ls	1. Changes the current working directory to another directory.
	2. Prints the current working directory.
	3. Short for lists; displays the file and directory names in the current working directory.

User input scenario 1:

>> 3

<< Correct! ls - Short for lists; displays the file and directory names in the current working directory. You have been awarded 1 point. Your point total is 1.

User input scenario 2:

>> 2

<< Incorrect! ls - Short for lists; displays the file and directory names in the current working directory. You have lost 1 point. Your point total is -1.

No matter the result of each scenario, the next question is displayed.

During a single game play, a single command should <u>not be listed more than once</u>. If the user selects the correct answer, then a point is awarded to the player’s total. Otherwise, a point is subtracted from the player’s total. Once the game is over, the user’s profile is updated. However, the updates should not be written to `profiles`.csv until the “Exit” command is requested. This saves the overhead of accessing the disk storage memory more than necessary. Return to the “main” menu.

3.4.1 Design considerations
You must populate a linked list, constructed from a *template* with 2 parameters, with the commands and corresponding descriptions loaded from `commands.csv`. You may NOT apply the Standard Template Library (STL) List. Each *node* in the list contains 2 data member types. For this specific program; one type for the *command* and one type for the *description*.

You must update the corresponding user profile stored in the *profiles* array.

3.5 Load Previous Game Requirement
When option 3 is selected from the “main” menu, initially, prompt the user for a profile name. If the profile is located in the *profiles* array, then display the stored points to the screen. From this point, the game should run the same as in the “Play Game Requirement”.

3.5.1 Design considerations
You should store the (*name*, *points)* pairs in a single array. Should it be an array of structs or classes?

3.6 Add Command Requirement
When option 4 is selected from the “main” menu, the user should be prompted for the command to add to the current list of commands. Adding the command should be completed in 2 parts. First, ask for the command. Second, ask for the description. At this point, only the linked list should be changed. The `commands.csv` file should not be modified until the “Exit” command from the “main” menu is selected. This saves the overhead of accessing the disk storage memory more than necessary. Duplicates should NOT be allowed. If a duplicate is encountered, then the user should be warned and prompted until a valid unique command is entered or until the user “quits”. Once the action is completed, return to the “main” menu.

3.7 Remove Command Requirement
When option 5 is selected from the “main” menu, the user should be prompted for the command to remove from `commands.csv`. At this point, only the linked list should be changed. The `commands.csv` file should not

be modified until the “Exit” command from the “main” menu is selected. This saves the overhead of accessing the disk storage memory more than necessary. If the command does not exist, then the user should be warned and prompted until a valid command is entered or until the user “quits”. Once the action is completed, return to the “main” menu.

3.8 Exit Requirement

When option 6 is selected from the “main” menu, all commands and user profiles should be saved to the correct files. All memory allocated for the linked list should be deallocated. All files should be closed, and the program should exit.

- The contents of the linked list should be written to a file called `commands.csv`. Each line in the file should consist of the command and description pairs in the form *command,”description”*.
- Each profile includes a name and current points, and should be written to a file called `profiles.csv`. If another profile exists that matches the name, then it should be completely overwritten with the new points. The format for a profile is *name,points*.

3.9 Other Requirements

- You will list 1 advantage and 1 disadvantage of using a linked list for the data structure involved with storing the commands and descriptions; you will relate your ideas to the way the list is used in THIS assignment. Your advantage and disadvantage must be listed in the comment block at the top of your `main.cpp` file under a clearly marked area called “ADVANTAGES/DISADVANTAGES LINKED LIST:”.
- You will list 1 advantage and 1 disadvantage of using an array for the data structure involved with storing the user profiles; you will relate your ideas to the way the list is used in THIS assignment. Your advantage and disadvantage should be listed in the comment block at the top of your `main.cpp` file under a clearly marked area called “ADVANTAGES/DISADVANTAGES ARRAY:”.

- You should use **CMake** to compile your project and make sure that your submission includes necessary files for TA to re-produce your project using CMake as well.

4 Submitting Assignment: Git (sharing with TA by creating a Github repository)

1. Create a **private** repository on github.com (You may name this private repository such as "CPTS223_assignments", etc.). Use your terminal to get a local copy from it to your computer (by using the "git clone" command).
2. On your local file system, and inside of your Git repo for the class, create a new branch called PA1 (by using the "git checkout -b PA1" command). In the current working directory, also create a new directory called PA1. Place all PA1 files in the directory, if you are not directly working in your repo. This directory should have at least one header file (a .h file), two C++ source files (one of which must be called main.cpp) and a CMakeFiles.txt. Please feel free to add more files.
3. All files for PA 1 should be added (e.g., "git add header.h source.cpp"), committed (e.g., "git commit -m 'initial'"), and pushed ("git push") to the remote origin in your **private** GitHub repo. Do not push new commits the branch after you **submit your link to Canvas**, otherwise it might be considered as late submission.
4. Your project must build properly by GitHub Action (example: https://github.com/DataOceanLab/CPTS-223-Examples/actions). The most points an assignment can receive if it does not build properly is 65 out of 100.
5. You can start from the GitHub template project: https://github.com/DataOceanLab/CPTS-223-Examples
6. Submission: You must submit a URL link to the branch of your private GitHub repository. Please **add the GitHub accounts of TAs** (see Syllabus/Schedule page and check TA's names as well as their Github usernames before submitting) as the collaborators of your repository. Otherwise, we will not be able to see your repository and grade your submission.

5 Grading Guidelines

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

4 pts (2 pts/advantage and disadvantage) for listing 1 advantage and 1 disadvantage of using a linked list for the data structure involved with storing the commands and descriptions; you must relate your ideas to the way the list is used in THIS assignment; your advantages and disadvantages must be listed in the comment block at the top of your `main.cpp` file under a clearly marked area called "ADVANTAGES/DISADVANTAGES LINKED LIST:".

4 pts (2 pts/advantage and disadvantage) for listing 1 advantage and 1 disadvantage of using an array for the data structure involved with storing

the user profiles; you must relate your ideas to the way the list is used in THIS assignment; your advantages and disadvantages must be listed in the comment block at the top of your `main.cpp` file under a clearly marked area called "ADVANTAGES/DISADVANTAGES ARRAY:".

30 pts for developing a correct class List template with 2 parameters
- 6 pts for correct insertAtFront () – needed when loading from `commands.csv`
- 7 pts for correct removeNode () – needed when removing commands from the list
- 10 pts for other functions needed to operate on the list
- 4 pts for correct data members
- 3 pts for correct constructors/destructors

5 pts for satisfying the *Main Menu Requirement*

20 pts for satisfying the *Play Game Requirement*
- 2 pts for correctly prompting user for the number of questions.
- 8 pts for correctly generating a command and 3 descriptions
- 3 pts for verifying answer
- 2 pts for updating player points
- 5 pts for generating correct number of questions

7 pts for satisfying the *Load Previous Game Requirement*

5 pts for satisfying the *Add Command Requirement*

5 pts for satisfying *Remove Command Requirement*

10 pts for satisfying the *Exit Requirement*
- 4 pts for correctly writing to `commands.csv`
- 3 pts for correctly writing to `profiles.csv`
- 2 pts for deallocating linked list memory
- 1 pt for closing the files

5 pts for appropriate class and top-down design

5 pts for adherence to proper programming style established for the class and comments