CPTS 223 Advanced Data Structure C/C++ Fall 2024

PA1: Matching Game for Linux Commands and C++ Review

1 Learning Objectives

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

- Develop a C++ program using templates;
- List and define 30 popular Linux commands;
- Construct a class template for List;
- Open, edit, parse, and close .csv files in C++;
- Create and maintain a repository on Github;
- Build the program using cross-platform tool CMake;
- Discuss the advantages and disadvantages of applying a linked list and an array to store data in this assignment.

2 Prerequisite

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 Unix-like environment (Linux, macOS or WSL);
- Describe and implement a linked list;

3 Overview and Requirements

The objective of the matching game is to allow a player to match Linux *commands* to appropriate *descriptions* of those commands. If the player matches a command with an appropriate description, then the player earns 1 point. If the player does not match the command with an appropriate description, then the player loses 1 point. Negative point totals are possible. The player selects the number of matching questions at the beginning of the game. The game continues until this number is reached. Each player's profile (i.e., `name, points`, see below) should be saved.

Commands should be stored in a file called "commands.csv". Each line of this file is 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."

Command Name	Command Name	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

Table 1: 30 command names that are required in the matching game.

Please note that you should place "" around your description so that it is interpreted as one value in the .csv file. Each pair should be placed on its own line in the file. Commands may be added and removed from the file through the program (see Section 3.6 and 3.7). At least the 30 commands in Table 1 should be supported at the beginning of the game.

You are free to manually populate the commands.csv file to store the initial 30 commands. It is suggested to explore more 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 of each player's profile file is:

```
name, points
```

Below subsections detail the requirements of this matching game.

3.1 Load Data from Files

Upon startup of the program, the Linux command and description pairs from "commands.csv" should be loaded into a singly linked list that was constructed from a 2-parameter template (see "Design considerations" in Section 3.4). All commands can be inserted at the front of the list as they are read from the file. On the other hand, the user name and points pairs from "profiles.csv" should be loaded into a profile array. All user name and points pairs must also be loaded at the front (index 0) of the array.

Is the profile's array a poor design? You will discuss this as part of the assignment!
See Section 3.9.

3.2 Main Menu

Display a main menu. Options include:

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

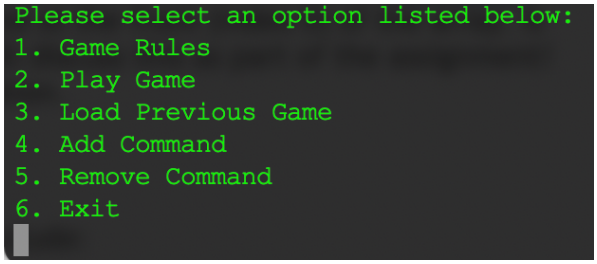

Figure 1: Main menu.

After each option is complete, the program should return to the "main" menu. Error checking for user input is required. An example of the main menu is in Figure 1.

3.3 Game Rules

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, as shown in Figure 2. You need to summarize the rules for YOUR version of the game in your own words.

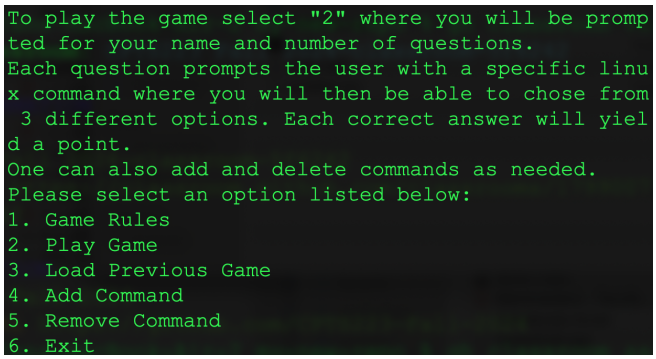

Figure 2: Option 1.

3.4 Game Requirements

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 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, as shown in Figure 3.

Depending on the user's answer in the context of Figure 3, the program has different responses:

- Case 1: user chooses 3.

```
>> 3
<< Correct.
   You have been awarded 1 point.
   Your point total is 1.
```

- Case 2: user input 1 or 2:

```
>> 2

<< Incorrect!
   You have lost 1 point.
   Your point total is -1.
```

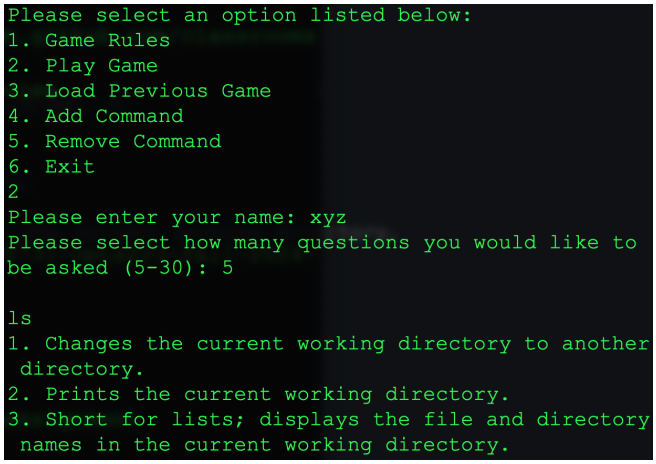

Figure 3: Option 2.

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

During a single game play, a single command should not be listed more than once. 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.

Design Considerations. You should populate a linked list, constructed from a template with 2 parameters, with the commands and corresponding descriptions loaded from "commands.csv". You should NOT apply the Standard Template Library (STL) List in this assignment. Each node in the list contains 2 data member types. For this specific program, one type for the command and the other type for the description. You should update the corresponding user profile stored in the profiles array after "Exit" command is requested.

3.5 Load Previous Game

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 Section 3.4. You should store the (name, points) pairs in a single array. (Should it be an array of struccts or classes?)

3.6 Add Command

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

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

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 by:

- 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 starting with "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 starting with "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 How to Submit: Github (and Share with TA)

1. Create a **private** repository on github.com. Name this repo as "CPTS223_assignments". This is a YouTube video showing how to: https://www.youtube.com/watch?v=vpRkAoCqX3o. Use your terminal to get a local copy from it to your computer (by using the "git clone" command with the link copied from the HTTPS link). If you have any issue regarding the Github authentication to push your repository to GH, please try `gh auth login` (via gh rather than via git) to login. See this YouTube video https://www.youtube.com/watch?v=cJYmjSzGnXM.

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. Also refer to this YouTube video https://www.youtube.com/watch?v=Wbz8zM_5iCc). 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 (provided in PA1 on Canvas). Please feel free to add more files.

3. All files for PA1 should be added (e.g., "git add header.h source.cpp"), committed (e.g., "git commit -m 'initial'"), and pushed ("git push" or "git push -u origin PA1" for the firt time push to PA1) 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 should build properly by CMake and be tested on your local computer. Otherwise partial credit will be given.

5. Refer to the CMake example at 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.

Here is a checklist for submitting your code:

- Invite all TAs and instructor as collaborators of your created repository "CPTS223_assignments". Their github username can be found in Syllabus on Canvas.
- Commit and push your local code to your repository.
- Make sure that your repository reflects and contains all your latest files.
- Copy the link to your repository in the Canvas submission portal.

5 Grading Guidelines

This assignment is worth 100 points. Your assignment will be evaluated based on a successful **cross-platform** 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, as in Section 3.9.
- 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, as in Section 3.9.
- 30 pts for developing a correct class List template with 2 parameters:
  - 6 pts for correct insertAtFront() (or other names) – needed when loading from "commands.csv";
  - 7 pts for correct removeNode() (or other names) – 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 (Section 3.2) and game rules (Section 3.3).
- 20 pts for satisfying the Game Requirement (Section 3.4).

  - 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 (Section 3.5).
- 5 pts for satisfying the Add Command Requirement (Section 3.6).
- 5 pts for satisfying Remove Command Requirement (Section 3.7).
- 10 pts for satisfying the Exit Requirement (Section 3.8):
  - 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.