COP5536 Spring 2025 Programming Project

1 Problem Description

You are hired to write a new flying broomstick management system for Office of Transportation, Ministry of Magic. The system, aiming at managing thousands of manned cleaning tools manufactured and registered every year, requires you to use **Red-Black Tree** as underlying data structure and dictionary-style data entries with the license plate numbers as key.

According to laws and regulations, all flying broomsticks must bear a license plate with **4** characters. Each character can either be a number or a capital letter. When registering a flying broomstick, the owner can either customize their plate by choosing their own plate number or let the system randomly generate one. Registration and management fee for a plate is 4 Galleons annually, with an extra 3 Galleons charged for customized plate number per year.

1.1 Functional Requirements

The system you create needs to be able to execute the following operations:

- addLicence(plateNum): register new customized license plate into the database.
  Output format:
  - If added successfully: *%plateNum% registered successfully.*
  - If %plateNum% already exists in database: *Failed to register %plateNum%: already exists.*
- addLicence(): randomly create a new license plate that does not currently exist and add into the database.
  Output format: %plateNum% created and registered successfully.
- dropLicence(plateNum): remove record of a license plate from the database.
  Output format:
  - If removed successfully: *%plateNum% removed successfully.*
  - If %plateNum% does not exist in database: *Failed to remove %plateNum%: does not exist.*

- lookupLicence(plateNum): check in the database if the license plate exists in it.
  Output format:
  - If exists: *%plateNum% exists.*
  - If not: *%plateNum% does not exist.*
- lookupPrev(plateNum): check in the database for the license plate number lexicographically previous to the input.
  Output format: %plateNum%'s prev is %plateNumFound%.
- lookupNext(plateNum): check in the database for the license plate number lexicographically next to the input.
  Output format: %plateNum%'s next is %plateNumFound%.
- lookupRange(lo, hi): check in the database and output all license plates between lo and hi in lexicographical order.
  Output format: plate numbers between %lo% and %hi%: %plateNum%, %plateNum%, %plateNum%.
- revenue(): report the annual revenue on broom registration and management fee and customized plate number fee.
  Output format: Current annual revenue is %amount% Galleons.
- quit(): stop the program.

Note: assume numeral entries come before letters in the lexicographical order of plates, i.e. "A" comes after "9".

1.2 Programming Requirements

You are required to program using either Java, C++, or Python for this project. Your program will be tested using the Java/g++/python environment provided on the thunder.cise.ufl.edu server, so you should verify that it compiles and runs as expected on the server. To access the CISE server, type "ssh yourUFUsername@thunder.cise.ufl.edu" in your terminal. Your submission must include a makefile that creates an executable file named plateMgmt.

Your program should execute using the following commands:

- C++: ./ plateMgmt inputFileName
- Java: java plateMgmt inputFileName
- Python: python3 plateMgmt inputFileName

You are required to write Red-Black tree from scratch without using built-in libraries.

1.3 I/O Requirements

Your program should be able to handle I/O from/to raw text files on the disk. Specifically:

- Read input from a text file where inputFileName is specified as part of the command line argument.
- Output should be written to a text file with file name as concatenation of inputFileName + "_" + "output.txt".

1.4 Submission Requirements

Submit your work as a single **.zip** file. Please include:

- Your code, well indented and commented, human readable;
- A makefile for easy compilation;
- A report in .pdf format including project details, program structure, function prototypes, and explanations. It should also contain your info including name, UFID and UF email.

Do not use nexted directories. All files must be in the first directory that appears after unzipping. After that, please name your compressed .zip file LastName_FirstName_UFID.zip and submit it to Canvas. Please make sure that the name you provide (both on file name and in report) is the same as the name that appears on Canvas.

Please do not submit directly to a TA as all email submissions will be ignored without further notifications. Please note that the due date is a hard deadline and no late submission will be allowed.

2 Grading Policies

Your assignment will be graded based on correctness, efficiency, elegancy, and other common criteria for grading a programming assignment. Specifically:

- Implementation and execution: 70%;
- Code style: 15%;
- Report: 15%.

Your assignment will be graded based on produced output. Therefore, it is vital to make sure that your program produces the correct output to get points. There will also be a threshold for running time. If your program runs slow, we will assume that you have not implemented the required data structures properly. You will also lose points if you do are not following the I/O and submission requirements listed above, including but not limited to:

- Source files not in a single directory after unzipping;
- Unable to process input file name;
- Error in makefile;
- Makefile does not produce an executable that can be run with the commands listed above;
- Incorrect output file name;
- Not following output format.

We reserve the right to ask you to fix above problems and demo your project.

3 Academic Honesty

This is not a team assignment. Any sorts of collaboration is strictly prohibited. Your program should be your own and you have to work by yourself. Discussion between classmates are restricted only on concept level. Your submission will be checked for plagiarism.