Assignment 4 - COEN 244

Stuart Thiel

September 18, 2023

Introduction

This assignment is worth **5%** of your grade. This is an **individual assignment** (so individual, I put *your* student ID on it) and you should not share your assignment with anyone else. The **assignment is due Nov 24th at 11:59pm, Montreal time**.

All submissions must go through Moodle. I like EAS, but the new VPN rules suck.

Q1) Filtering Files

Assume the information of all the TAs for the ECE department is stored in a file that is called TAs.txt. In order to be eligible for the TA position, a student should be currently registered (i.e. not be an alumni). However, the file contains old data; it includes some records of TAs who have already graduated and who are no longer allowed to TA. The format of each TA record in the file is as follows:

```
Age    Student_Id    Status    Year_Hired    Working_Hours
```

The first line of the file should show the total number of students records (max number of records is 100, you don't need to check for going over).

Note that the TAs can be classified as any of the following (even if they don't all appear in your example):

- **Grad**
- **UGrad**
- **Alum**

Part A)

You need to write a program that reads the file, removes any lines with invalid TAs (those who have Alum as the value for classification) and updates the original file in the same format, just having removed invalid TAs.

Work with a vector of TA objects.

Example 1

Initial Student File:

```
5
25    9134091    UGrad    2014    10
25    6442707    UGrad    2016    7
35    8656886    UGrad    2017    9
26    1938356    Grad    2019    9
23    2282790    UGrad    2014    10
```

After Correction:

```
5
25    9134091    UGrad    2014    10
25    6442707    UGrad    2016    7
35    8656886    UGrad    2017    9
26    1938356    Grad    2019    9
23    2282790    UGrad    2014    10
```

Note: Depending on the randomly generated data, it is possible that all TAs in the initial file are valid and no lines need to be removed. If this is the case, experiment with adding a known invalid student and test your code with various combinations to ensure its correctness before submitting.

Part B)

Implement a function `addNewTA()` that prompts the user for the information representing one TA. When a user selects 1, ask the user to enter new TAs: TA information should be entered in the same order as the fields are stored in the file:

```
Age Student_Id Status Year_Hired Working_Hours
```

Please enter TA information separated by tabs to avoid confusion when parsing.

Adding a TA in this manner should add the TA into the existing file. However, if a user enters a TA with a `Student_Id` that is already in the file, the program should reject the input and keep looping until a new `Student_Id` is entered (but not entering the other data, that should be remembered).

Your program should handle any exceptions (or other problems) if the user tries to enter information of an unexpected type, like a string for the `Student_Id`.

Here is an example of how to handle an exception in C++:

```cpp
int main() {
    std::string input;
    std::cout << "Enter an integer: ";

    try {
        std::getline(std::cin, input);
        int number = std::stoi(input);
        std::cout << "You entered " << number << std::endl;
    } catch (const std::invalid_argument& e) {
        std::cerr << "Error: " << e.what() << " is not a valid integer.\n";
    } catch (const std::out_of_range& e) {
        std::cerr << "Error: " << e.what() << " is out of range.\n";
    }
```

Figure 1: Handling an exception in C++

Note: This menu should be offered after the original behavior from Part A is complete. When the user selects a menu item, that functionality should complete before returning to the menu.

Note: Creating sample data that you can test your program with is a great way to ensure your code works as expected. We highly recommend you create your own sample data to test your program, in addition to the examples we provide in this assignment. You can create sample data by generating multiple TAs with different student IDs and classifications. Be sure to include invalid student IDs to test that your program correctly rejects them and keeps looping until a new, valid ID is entered.

Part C)

Extend your TA record management system to offer a new option. If the user enters a 2, they should be prompted to select from a list of columns to order by, and then choose to order in ascending or descending order. The new ordering should be written out to the file in the same format as before.

To accomplish this, use the TA class you implemented in the previous part of the assignment, and store your TA instances in a vector. Then, use the `sort` function from the C++ algorithm library to perform the sorting. Make sure your compiler is set to use at least C++11 so that you can use lambdas in your sort function.

When sorting the TAs, be careful to ensure that strings are ordered as strings and numbers are ordered as numbers (i.e. 10 should not end up before 2). Consult the `sort` function documentation for guidance on how to accomplish this.

Here is an example of how to use the `sort` function with a lambda to sort a vector of integers in descending order:

```
21    std::vector<int> vec{5, 2, 8, 1, 6};
22    std::sort(vec.begin(), vec.end(), [](int a, int b) {
23        return a > b;
24    });
```

Figure 2: Using `sort` function with a lambda to sort a vector of integers in descending order

Note: Creating sample data that you can test your program with is a great way to ensure your code works as expected. We highly recommend you create your own sample data to test your program, in addition to the examples we provide in this assignment. Try creating a few TAs and sorting them by different fields to make sure your sorting function works as intended