ENGR 151

Fall 2021

Lab 8: File I/O and Vectors

Files to turn in

Refer to the table below for the due date of the lab 8 assignment.

Lab Day	Due Date
Monday	11/05/2023, 11:59 P.M.
Tuesday	11/06/2023, 11:59 P.M.
Friday	11/09/2023, 11:59 P.M

You have two files to submit to the autograder:

- data_entry.cpp
- parking.cpp

Introduction

In this lab, you'll be working with File I/O and Vectors. You will open, read from, write to, and close files in your code. You will also use some C++ Strings. These are not as low-level as C strings, meaning you don't have to worry about memory and the size of the string. They also have many functions available in the `<string>` library.

Completion Criteria

Use this checklist to ensure that you've completed all activities in this lab and to understand what we'll be looking for when grading for completeness.

- **data_entry.cpp**
    - **Input**: uses file stream to read in an arbitrary number of entries from file called `data_entry_input.txt`
    - **Output**: uses file stream to output corresponding values to file called `data_entry_output.txt`
    - Remember to separate numbers with **tabs** and handle any exception cases properly (see page 6 for how to add tabs)
- **parking.cpp**

- Contains finished function `parking_police`
- Properly call `parking_police` function in `main`
- `parking_police` should fill the 2 vectors: `expired` and `illegal`, and **return** the number of empty spaces left in the lot
- Edit the `print_info` function to write to an output filestream (file name is `parking_output.txt`) instead of an iostream

Concepts Review

File Streams

A few reminders for file stream syntax and functions:
Checking successful opening of a file:
(`#include <cstdlib>` if you use the exit function)

```
     if(infile.fail()) {
          cout << "problem reading file\n";
          exit(1);
     }
```

Reading file of an unknown length:

```
     while(infile>>word) {
          //loops as long as you can read a word from infile
     }

     while(getline(stream, line)) {
          //loops as long as you can read a line from infile
     }
```

Reminder: Don't mix >> and `getline()`!

Strings

When working with C++ strings, we have access to many fun functions through the `string` library. Remember to `#include <string>`. Refer to the following table to understand the functions provided by the C++ string.

Reminder: May need to cast `str.size()` from `size_t` to `int` (Note that `size_t` is just a type that changes depending on the platform being used, but it's always `unsigned`, unlike `int`!)

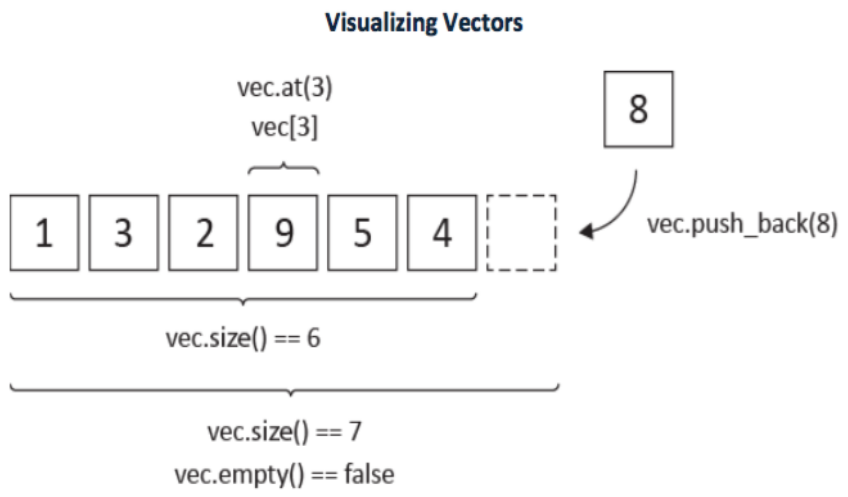
Name	**Return Type**	**Functionality**
`str.at(i)`	`char`	Index into the i-th element of a string *disallows indexing past the end of a string by raising an exception* Remember: C++ indexing starts at 0
`str[i]`	`char`	the unsafe version of at(): indexes into string and beyond!
`str.size()`	`size_t`	number of chars in the string (which is NOT the index of the last char)
`str.append(s2)`	`string&`	appends string s2 to str
`str.push_back(c)`	`void`	appends char c to str
`str.find(s2)`	`size_t`	search for s2 in str and returns the location of the first character of the first match (Returns str.npos (no position) if no match was found
`str.insert(offset,s2)`	`string&`	insert s2 in str starting at offset
`str.erase(offset,num)`	`string&`	erase num chars from str, starting at offset
`str.replace(offset,num,s2)`	`string&`	erase plus insert
`str.substr(offset,len)`	`string`	returns a new string that is the part of str that starts at offset and is len long
`str.empty()`	`bool`	tests whether the string is empty
`str.c_str()`	`const char*`	Returns the c-string equivalent of string str. Needed when using a string as a filename, for example.

Vectors

A vector is a C++ data structure that can hold multiple elements. More specifically, vectors can contain many values of the same data type. Vectors are something called a "templated" class, which means they can hold any type of variable (such as int, double, char, string, and even other vectors).

Like C++ strings, vectors have a flexible size and a nice library of functions available in the <vector> library. Below are some common functions of the vector class.

Function Name	**Return Type**	**Functionality**
`vec.at(i)`	Same as type contained by `vec`	Returns the element at the i^{th} position of vec. If i is out of range, it will abort and crash the program through "SIGABRT".
`vec[i]`	Same as type contained by `vec`	Returns the character at the i^{th} position of vec. If i is out of range, it will abort and crash the program through "SEGFAULT". This is a nasty error; try to avoid it by using the above.
`vec.size()`	`int`	Returns the length of the vector.
`vec.resize(s)`	`void`	Resizes the vector to be of size s.
`vec.push_back(elem)`	`void`	Adds the element "elem" to the end of the vector. Note: elem must be of the same data type as the vector.
`vec.clear()`	`void`	Removes all elements from vec, resulting in an empty vector.
`vec.empty()`	`bool`	Returns true if the vector is empty, false otherwise.



Exercise 1: Date and Time

Natural language processing is a topic in computer science which deals with computers' ability to understand and analyze human language. In this exercise, we will be parsing the human representation of a date and time entry.

Note: this is purely syntactic processing and we will be not dealing with any real semantics of language.

Suppose that some researchers have enlisted your help. They have been studying cell behavior over several decades and have accumulated thousands of data entries (start and end times of each experiment). Unfortunately, when they designed the experiment, their only goal was for an intuitive interface, so the entries in their database are as follows:

    10:35 AM on September 3rd, 1996

This is not an easy form in which to analyze the data. Your task is to convert the date/time into the following format:

    Min   Hour   Day   Month   Year

Where the hour will be in military time format (i.e. 1 PM is 13). For example, given the input:

7:15 PM on October 21st, 2019
12:01 AM on May 5th, 2000

the output would be:

```
15	19	21	10	2019
1	0	5	5	2000
```

Think about any edge cases there might be and make sure you handle them properly!

You can assume that the format of the input will always be consistent. The only variation is if the hour is 1 or 2 digits. Also, remember that there are a fixed number of suffixes for numbers.

In your **output**, each entry should be on **a separate line** with each number **separated by a tab** (so it lines up nicely). Just as `cout << '\n'` prints a newline, `cout << '\t'` prints a tab.

An example output file **data_entry_output.txt** can be downloaded from Canvas. This sample output file is the output you should obtain after running your code with the **data_entry_input.txt** also downloaded from Canvas. Add in your own test cases to make sure you've accounted for any edge cases you can think of!

Remember

- This code should work for an input file of any number of lines
- Switch statements cannot take strings as cases
- If you want to use the function stoi, you may need to compile with the flag -std=c++11
- See page 2 Concepts Review for how to check for error opening a file and what the **exact output** you should give to the iostream should look like.

Create a file and name it **data_entry.cpp**. On Canvas you will find a file called **data_entry_input.txt**. Given that file as input, use your program to generate the file **data_entry_output.txt**.

Exercise 2: Parking Police

Use the file in Canvas called **starter_parking.cpp.** You will also need to download **parking_lot.txt** and **license_plates.txt** from Canvas, which will provide the input for this problem.

In this exercise you will **write one function**, **call it in** `main`, and **edit an existing function.**

In this part you will be given two 2D vectors: `parking_lot` (which is a vector of `ints`) and `license_plates` (which is a vector of `strings`). These vectors are the same size as each other. In `parking_lot` each element represents a parking spot, and in `license_plates`, each element is the license plate of the car parked in the corresponding spot.

These 2 vectors are filled with data from the files **parking_lot.txt** and **license_plates.txt,** which you downloaded. The function to read file input is already written for you. This function also reads the dimensions of the parking lot and assigns the values to the variables `rows` and `cols`. We strongly recommend you read the implementation of the prewritten functions so you understand the inputs, outputs, and the implementation of the functions.

Your Task

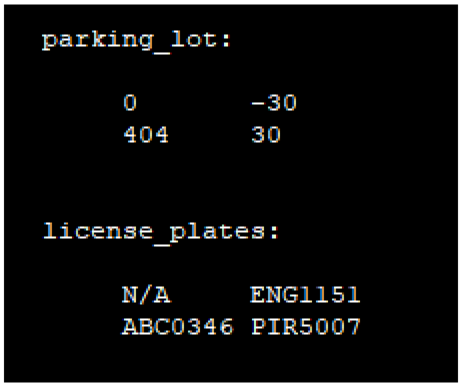
Write a function called **parking_police** that will traverse through the `parking_lot` vector and find the cars that are either parked illegally or their parking time is expired.

- If a car is parked illegally, then it's value in `parking_lot` equals `404`.
- If a car's time is expired, then it's value in `parking_lot` is a negative number.
- If the parking space is empty, then that spot in `parking_lot` is 0.

- When you find a car that is parked illegally, add that car's license plate to the vector `illegal`.
- When you find a car whose time has expired, add that car's license plate to the vector `expired`.

The order you add them does not matter. As in, the order in which you check if a parking space is empty, a car is parked there illegally, or the parking space has expired doesn't matter. (Ask yourself, why?) Also, keep track of how many empty spaces there are in the parking lot using the variable `free_spaces`

For example, consider the mini 2x2 parking lot shown below. After traversing `parking_lot`, the vector `illegal` would contain ABC0346, the vector `expired` would contain ENG1151, and `free_spaces` would be 1.



```
parking_lot:

     0       -30
     404     30


license_plates:

     N/A     ENG1151
     ABC0346 PIR5007
```

Your function will traverse the entire `parking_lot`, fill the 2 vectors `illegal` and `expired` with the license plates of the offending cars, and return an integer which is the number of empty spaces in the parking lot. No function header is provided, so you will have to think about which variables to pass to your function.

For example, since you need to change the variables `illegal` and `expired`, will they need to be passed by value or by reference? Also think about what other values you will need to access in this function and make sure to pass those as inputs to your function.

Make sure to call your function in `main`. A print function will `cout` the results after your function call, so you can check your output.

The last task for this exercise is to edit the `print_info` function so that it outputs the same output but to a file instead of to the terminal. **Replace all outputs to iostream with outputs to an ofstream.** You should write the outputs to a file called **parking_output.txt** using your knowledge of File I/O. **DO NOT modify the actual textual outputs! Or else you will not pass the Autograder!**