Lab 6: Static Member Variables and Functions

CPSC 1021

School of COMPUTING Clemson University

Lab Objective

- Learn and apply C++ File I/O + Manipulators
- Writing a simple Date class
- Using the std::sort function
- Using static variables and functions
- Using stringstreams to quickly build formatted strings
- Using vectors

Introduction

In today's lab, we will be combining several of the concepts and techniques we have covered in lecture. You will be implementing a basic Date class that will use public and private members, *static functions*, and *static data*. We will also use objects with vectors. In addition, you will explore three new topics: using I/O manipulation (reading from a file and writing output to a file), a new C++ class known as a stringstream, and the algorithm library's sort function.

Collaboration Policy

Work with your assigned team members from Lab 0. ***Please make sure to integrate your team members who – because of COVID requirements – have to participate remotely.*** You can use the MS Teams video conferencing feature to collaborate with a remote team member. Use the search bar on the top of the MS Teams app to search for your team member with their CU user name, then click the video icon in the top-right corner of the screen to start a video call. You and your partners can work on the assignments together, but every student has to submit their own files.

Prior to Working on This Handout

Make sure you can read in and write out to files in C++. This involves using C++ style streams. Using files for data storage is covered in Chapter 5.12, in our textbook. You can also read about C++ I/O operations here: http://www.cplusplus.com/doc/tutorial/files/

We will be using std::string's and stringstreams in this lab, but they will not be covered extensively. Still, these pages will be useful:

http://www.cplusplus.com/reference/string/string/

http://www.cplusplus.com/reference/sstream/stringstream/

Lab Instructions

Create four files, **main.cpp, Date.h, Date.cpp, and a Makefile**. The main program should create an executable **main.out** that accepts **two command line arguments:** an input text file and an output text file for writing results. For your program to accept command line arguments, declare main as follows:

```
main(int argc, char const *argv[])
```

You will be given an input file, dates.txt, who's first entry is the **number of date entries in the list**. This file is uploaded to Canvas and looks like this:

```
4

10 5 1993

3 4 1982

1 28 1943

1 24 1943
```

Each record begins with a **month** followed by **day** and then **year**.

Your Objectives

- Read through the input file and create and store a date object in a vector
- Using the sort() function, sort the list in ascending order
- Convert the month of the Date into a string with the month's name
- Print out the sorted dates to the output file

Sample Output

Before we go any further, here is the expected output for the example file above.

```
JANUARY    24   1943
JANUARY    28   1943
MARCH      4    1982
OCTOBER    5    1993
```

The month field should be set to a width of **10** characters

The day column should be set to a width of **3** characters

The year column should be set to a width of **5** characters

Years are ordered from earliest to latest

Date Class

You need to build a Date class to store each date. Here is the UML class diagram for Date:

Date
- month: int - day: int - year: int + MONTHS[12]: string static const
+ getMonth(): int const + setMonth(m: int): void + getDay(): int const + setDay(d: int): void + getYear(): int const + setYear(y: int): void + print(): string + compare(d1: Date const, d2: Date cont): bool static

The Date class is similar to the one you built for Project 1, but with some important differences. Please read the following specifications carefully!

Additional specifications:

- Use in-class initialization for all member variables. Use 01/01/1900 as the default date.
- Add a default constructor Date() and a constructor Date(m: int, d: int, year: int) with member list initialization to allow creating an instance of Date with user-provided values.
- For simplicity, input validation is NOT required for this Date class. Instead, we assume that all records in the input file are valid.
- `MONTHS[]` is a static string array used in `print()` to get month names.
- `print()` should use a **stringstream** to build and return a string that will be printed to the output file in main(). See sample output above.
- `compare()` takes as arguments two instances of Date, which should be passed as const references. The compare function is declared static.

Static compare function

The static keyword in the compare function tells C++ to make the function global to the class. This means that all Date objects share the same compare function and do not get their own copy. This is useful because the compare function is meant to be used on two instances of Date.

Implement the compare function such that it returns true **if the first date is earlier than the second date.** Note that if dates share a year, you need to check their months, and if they share a month you must compare them by day.

To call the compare function, you would execute the code:

```
Date d1 {10,5,1993}, d2 {4,3,2001};   // this line creates two instances of date
bool earlier = Date::compare(d1, d2); // this line compares the two instances and
                                      // checks which date is earlier
```

Note that **we must specify the Date scope to get access to this function.**

Static data MONTHS

Now that we have touched on the idea of **static members**, note that there is also an array of string month names that is also static. The static keyword means that all Date objects share the same array and do not get their own copy. Use this array (with the Date:: scope operator) in your print function to correctly output the name of the month instead of the number.

sort

We will use std::sort which is in the <algorithm> package to sort our dates. However, a Date is a complex object and not a simple string or integer. To make the sort() function work correctly, we will be **passing it the compare function** so that it knows how to compare two Dates.

http://www.cplusplus.com/reference/algorithm/sort/

Sort takes **3 arguments**: a beginning location, an ending location, and a compare function. If we were to store our dates in an array **dates** of size **num_dates**, we would call sort as follows:

```
sort(&dates[0], &dates[num_dates], Date::compare);
```

However, we are using vectors to store our instances of date. The link above gives you an example of how to iterate through the vector using the **begin** and **end** functions provided by the vector class.

stringstreams

We have already worked with strings. Next, we will discuss stringstreams, which is a stream wrapper for strings. Much like we have streams for reading input files and printing output, we can also use stream operators on basic strings.

For example, if we run the following code:

```
stringstream ss;

ss << "Hello World";

cout << ss.str();
```

we would print the string “Hello World” to the terminal. Stringstreams are useful because we **have access to the <iomanip> library**.

For example:

```
stringstream ss;

ss << left << setw(15) << "Hello World";

cout << ss.str();
```

would print the same string, but with a fixed width of 15 and justified left.

Use stringstreams in the Date.print() method such that the returned string follows the output formatting above.

Program Structure

Once you have complete implementing the Date.h and Date.cpp file, your **main.cpp** file should behave as follows:

1. Open input and output streams given as command line arguments.
2. Read in the first number from the file, which is the number of dates
3. Create a vector to store the instances of date
4. In a loop, read in the month/day/year entries
5. Add the date to the vector created in 3 above
6. Call sort() passing the static compare method
7. Print out the Dates to the output file, using the Date print() method to get a string for each.

Note: You know that there are four records in the input file. However, do NOT hard code the number 4 in your program. Instead, read the number of valid records from the first line in the input file into a variable. When using a loop, use that variable or other concepts we discussed in class (e.g., the range based loop).

Submission instructions

Submit your complete and correct program to Gradescope. Your submission must include the following four files:

1. main.cpp
2. Date.h
3. Date.cpp
4. Makefile (this file should create main.out)

Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.

Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.