

## Introduction

In today's lab, we will be implementing a basic Date class which we will use public and private members, static functions, and static data. We will also continue to practice with I/O manipulation and a new C++ class known as a stringstream. You will also use vectors. While we may not have covered vectors in class you have already been introduced to vectors in a previous lab. Additionally, you will use the algorithm library's sort function.

## Lab Objectives

- 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

## Prior to Lab

Make sure you can read in and write out to files in C++. This involves using C++ style streams. Read more 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 one file, **lab7.cpp**. This program should accept **two command line arguments**: an input text file and an output text file for writing results.

You will be given an input file who's first entry is the **number of date entries in the list**. This file will be provided on Canvas and looks like:

```
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**.

You may only use C++ style file I/O to manipulate data. **You may not use printf or scanf in your program.**

### 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	1	1000
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

### Date Class

You will be provided with the **Date.h** and **Date.cpp** source files. The CPP file is not entirely implemented: part of this lab is to implement the missing methods. You must change/implement the following:

- Change the regular/ parametrized constructor in the Date.h file such that it can be used as a default constructor, as well as a regular(parametrized)constructor. The default values should be 1, 1, 1900 representing month, day, year
- `bool Date::compare(Date& lhs, Date& rhs);`
- all getters and setters for the member data (month, day, year)
- `string Date::print();`

The **getters and setters** are standard functions in almost every class you'll write. They provide access to class data to outside entities. This is in following the principle of **data encapsulation**. You'll notice that the month, day, and year members of the Date class are declared **private** which means that only methods scoped to the Date class have direct access to the data. **Getters**

just return the data value and **Setters** assign a data value sent in as an argument. I have provided you with the setter and getter for month, use it as a model for the other two members.

The **print** function should use a **stringstream** to build and return a string that you'll print to file in the main(). See sample output above.

The **compare** function is interesting because it is declared **static**. We know that every object we create in C++ gets its own copy of member data and any functions we define. That is just to say that two different Date objects have their own copies of month, day, and year data variables as well as all the functions we declared (print, setters and getters).

The **static** keyword tell C++ to make a method or data member **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** and tell us which is earlier.

Static methods have many uses and are a slightly more advanced concept, but for now you can think of them as utility methods that we group under the Date namespace to keep things organized. To call the compare function, you would execute the code:

```
Date d1(10,5,1993), d2(4,3,2001);  
bool earlier = Date::compare(d1, d2);
```

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

Implement the compare function such that the compare function 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.

### 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. 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'll 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. So make sure `sort()` function 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.

## **Interesting information:**

### Array of Objects

One interesting consequence of using objects is that a simple expression like this:

```
Date dates[10];
```

calls the **constructor** for the `Date` class. This means we would get 10 default `Date` objects in our array. Since the member variables of the `Date` class are private, your program should use the **setters** to correctly input data from the input file.

## stringstreams

Last lab we learned about strings and how they are incredibly useful for storing text. 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 stringstream in the Date.print() method such that the returned string follows the output formatting above.

### Program Structure

Once you have complete implementing the Date.cpp source file, your **lab7.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. after you have add all the dates from the file to your vector create another instance of date in the following manner: Date test; This should invoke the default constructor. Oh Snap!! We did not have a default construct, so how is this going to work??????
7. call sort() passing the static compare method
8. print out the Dates to the output file, using the Date print() method to get a string for each. Hint. You know the size of the vector should be 5 (four read in from the file, plus the one from 6 above), however, you are not allowed to hard code the 5 in the loop. There are several ways to loop through the vector. (range based loop, or use the size function that is provided to us by the vector class.  
<https://en.cppreference.com/w/cpp/container/vector/size>

### What to turn in

- **lab7.cpp, Date.cpp, Date.h** which reads in a file of dates, sorts them, and prints the results to a file

### **SOME HINTS!**

Remember that streams make reading input really easy!

```
while (input >> month >> day >> year) { ... }
```

## Compile and Execute

Use g++ to compile your code as follows **and include the c++11 standard!**:

```
g++ -std=c++11 -Wall -g lab7.cpp Date.cpp -o dates
```

Execute the program

```
./dates <input> <output>
```

## FORMATTING:

1. Your program should be well documented
2. Each file should include a header:
3. Your program should consist of proper and consistent indentation
4. No lines of code should be more than 80 characters

```
// Sample Header
/*****
your name
username
Lab 1
Lab Section:
Name of TA
*****/
```

5 – 10 points will be deducted for each of the above formatting infractions.

## Submission Instructions

- Test your program on the School of Computing server prior to submitting.
- Use the tar utility to tar.gz all source files. **Do not tar an entire directory! When I untar your archive, I should see all the files you included, not a top-level directory! Failure to correctly tar may result in up to a 25-point penalty!**

**EX. `tar -czvf yfeaste-lab7.tar.gz lab7.cpp`**

- Name your tarred file `<username>-lab<#>.tar.gz` (ex. yfeaste-lab7.tar.gz)
- Use handin (<http://handin.cs.clemson.edu>) to submit your archive