ITP 365: Managing Data in C++

Homework 4: Wedding Planner Due: Friday, 10/20/2017 @ 11:59PM

Goal

Your two best friends here at USC are getting married! One is Brazilian and the other is Chinese. They need help planning the wedding so the two families can come to USC and celebrate together. The happy couple would like to get married within this calendar year. To make travel plans easy for everyone they are trying to avoid holidays in all three countries (Brazil, China, and US). You've been given the calendars for all 3 countries in CSV (comma separated values) files. You must compile all the holidays together from the CSV files first. Then you should create a basic user interface to accept a date and inform the user if there's a holiday on that date. The couple will use your program to plan their wedding!

Lab Setup

- Download and extract **HW4Start.zip** from Blackboard
- The 3 required CSV files are already in the proper location. There is one for each country. Do not modify these files.
- You have been provided a dateConv.h and dateConv.cpp. **Do not modify these files.** They provide helper functions to convert a date to epoch time and back again. Check the dateConv.h file function header for more details.
- At the top of the four files you will edit, make sure you add comments in the following format:

```
// ITP 365 Fall 2017
// HW4 – Wedding Planner
// Name: Tommy Trojan
// Email: ttrojan@usc.edu
// Platform: Windows
```

Part 0: Data file, dates, data structures

- There are 3 input files USholidays2017.csv (holds US holiday information), BRholidays2017.csv (for Brazil's holiday information) and CHholidays2017.csv (for China's holiday information). There are two important fields in this file:
  - Date (the day in MM/DD/YYYY form)
  - Holiday Name (the name of the holiday that occurs on that date)
- To make searching easier, rather than keep dates in MM/DD/YYYY form you will store Epoch Time. That is a whole number which represents the number of seconds that have elapsed since January 1, 1970. The number can be quite long, so it is a GoodIdea® to use the "unsigned long long", whenever working with Epoch Time.
- To make working with Epoch Time easier, there are some date converting functions in dateconv.h and dateconv.cpp. Check out the header for information about the functions.
- You may only the Stanford library for string convenience functions and the error function. You are to use std::vector and other STL collections instead of Stanford's collections.

Part 1: HEvent class

- In this part you will implement a class to hold holiday events. Each object of the class will encapsulate a single holiday event. The encapsulating class is called "HEvent".
- Create a hevent.h and a hevent.cpp and add them to your project. Your HEvent class will be defined in these files.
- The class data member variables are based on the data in the CSV file. They are:
    1. Epoch time of the holiday
    2. The holiday name
- The HEvent class only needs a parameterized constructor (no default constructor). This constructor should accept the 2 data members listed above (the epoch time, holiday name).
- Your HEvent class should not have any setters. All data should be set by a parameterized constructor.
- Your HEvent may have any getters you think are appropriate.
- To facilitate searching, you should also implement a comparator for HEvents – something like `operator<` or `operator>` as well as the something to test for equality (`operator==`). You can find the documentation on the function signatures here.
- You should also implement a way to display your HEvent by overloading the `operator<<`. You can use the functions provided to you in the date conversion library to turn the stored seconds into a date.
- You may add any other functions you need to assist in the above tasks.
- Be sure to test your new HEvent class. In particular, ensure that all the required functions work. Try creating a few HEvents. You could use a few examples listed in the .csv files. Then try displaying them with std::cout and checking that the comparison operators work appropriately.

Part 2: HCal class

- Now you will make a holiday calendar. Each calendar will be a collection of holiday events (or HEvent objects). The HCal class will contain a collection of HEvents. But because there may be scoping problems with the HEvents created in main, we will maintain a collection of HEvent pointers.
- Add hcal.h and hcal.cpp files to your project. You will define your HCal class in here.
- HCal will hold a vector of HEvent pointers. **Make sure to use std::vector.**
- HCal should have a default constructor and no parameterized constructor.
- HCal must have an addEvent function that will add the inputted HEvent pointer to the end of HEvent pointer vector.
- HCal should have a getter for the size of the contained vector.
- You must have way to access the underlying vector with a get function. It should accept an unsigned int as input and return a pointer to the appropriate HEvent pointer.
- You may add any other functions that will help you along the way.
- You should probably test your HCal class by dynamically allocating (with the new keyword) several HEvent variables and adding them to a HCal. Access the contained HEvents (maybe with a loop) and display the event details.

Part 3: loadCal function

- You will now create a function in main to accept string holding a file name and will return a pointer to a HCal. This function will read in the .csv file, create the appropriate holiday events, and load them into a calendar.
- Examine one of the .csv files using a text editor (you may use Excel, but use a text editor too). Be sure you understand the format and content of the text files. Notice that the files have the holidays listed in chronological order.
- Your function must process a text file by reading in the file and dynamically creating an HEvent for each line of data in the file. Each of the new HEvents should be added to the end of a new a new HCal you've also created dynamically.
- Return a pointer to the HCal when your function is done.
- Be sure to test this function before moving on by reading in a calendar or two and displaying its contents.

Part 4: merge function

- Now we need to put all the Brazilian, Chinese, and US calendars together. To do this, we'll merge calendars – 2 at a time – into a new calendar with a merge function.
- Create a merge function in main. It should accept 2 HCals (a pointer to HCal A and a pointer to HCal B) and return a new HCal (returned as a pointer).
- You may be tempted to use selection sort or another $O(n^2)$ sort function – but we know the data in each of the inputted the calendars is already in chronological order so let's take advantage of that and merge the 2 calendars on the order of O(n+m) where n length of A and m is the length of B).
- Here's the algorithm for our merge:
    1. Dynamically create a new HCal. This will be the value we return. It will be called HCal C.
    2. Create indexes for each of the 2 inputted calendars. Index i will be used to count through HCal A and index j will be used to count through HCal B.
    3. Create a loop that executes while i and j are within range (before the ends of their respective calendars).
        1. If the HEvent at A[i] occurs before the HEvent at B[j] add A[i] to the end of C.
        2. Then increment i.
        3. Otherwise add the event at B[j] to the end of C.
        4. Then increment j.
    4. When the above loop is over there will be a portion of either A or B left over (it hasn't been added to C). If A has items remaining, add them to the end of C. If B has items remaining add them to the end of C.
- You can test your merge function by merging the calendars together and displaying the contents. The contents should display in chronological order.

Part 5: search and binarySearchPart functions

- Now that we have a sorted HCal with all the HEvents we'll need to implement a way to check if a particular date exists in our HCal vector.

- Create a public function in HCal called search. It accepts an HEvent pointer as input. You'll check this HEvent against the other HEvents in the HCal to see if something is scheduled on this date.
- To perform the search, implement the binary search algorithm from class. Be sure to add the private helper function (binarySearchPart) to the HCal class too.
- Test your search function by creating a few events (some already on the calendar, some not). Use the search in main to verify you can find the existing events. You should also verify that the missing events generate a detectable error and don't just quit your program.

Part 6: User interface

- We have all the pieces together now to help the happy couple find an appropriate day for their wedding. Now we need to implement a user interface for our program.
- Read in all the data files and create a merged calendar for 2017 for all 3 countries.
- Create a new HEvent object and use the search function in HCal to see if an event occurs the date in question.
- Inform the user of what holiday occurs on the selected date (or if the date is available).
- If the user enters "q" or "Q" quit the program.

Sample output

Below is sample output for a full run-through of the program. User input is in **red**.

```
Welcome to the wedding planner!
Please enter a date (MM/DD/YYYY): 7/4/2017
7/4/2017 is Independence Day (National holiday)

Please enter a date (MM/DD/YYYY): 12/15/2017
That's a great day for a wedding!

Please enter a date (MM/DD/YYYY): q
Quitting!
```

A Note on Style

Style will be a small portion of the overall homework grade (5%). If you write spaghetti code, don't provide comments for function declarations, and so on, you will lose some points. We will similarly make sure you use the separate files correctly.

Deliverables

1. You must submit a ZIP file to Blackboard called **HW4Submit.zip**, using the designated Homework 4 submission link in the "Assignments" section. It should include all .cpp and .h files you've created or modified.
2. Follow the same procedure to create the ZIP file as in Homework 1.

Grading

Item	Points
Part 1: HEvent class	15
Part 2: HCal class	15
Part 3: loadCal function	15
Part 4: merge function	20
Part 5: search and binarySearchPart functions	20
Part 6: User interface	10
Comments, style, and correct use of separate files	5
Total	**100**