COMP 2404 -- Assignment #1

Due: Tuesday, October 9, 2018 at 12:00 pm (noon)

Goal

You will be working with an existing event management program throughout the term. The base code that you must start with is posted in *cuLearn*. For this assignment, you will modify the base code to improve the design and add new entity classes.

Learning Outcomes

With this assignment, you will:

- understand an existing object-oriented program
- add code to an existing code base with simple C++ classes
- work with dynamically allocated memory and pointers

Instructions

1. Understand the base code:

- Read and make sure that you thoroughly understand the existing event management program provided in the base code.
- Build the program and run it several times with different input. Use pipelining for standard input redirection from the given `in.txt` file to test the program, as you did in the tutorials. Add your own data to this file to test the program more thoroughly.

2. Modify the Date class

You will modify the `Date` class as follows, reusing existing functions everywhere appropriate:

- add a data member that is a `Time` object; the `Time` class is provided as part of the base code
  - modify the constructor and `set()` member function accordingly, so that they initialize the hours and minutes from parameters; the seconds can be set to zero
  - modify the `print()` member function to print out the time with the date
- add a `lessThan()` member function that compares the given parameter with the `Date` object on which the function is called; comparing the date must involve comparing the time as well; the function will have the following prototype:

```
bool lessThan(Date&)
```

3. Implement the Event class

You will create a new class called `Event`. This class will correspond to a new event that will be managed in the program. The `Event` class will contain the following:

- a data member for the event name, which can be a string
- a data member that's a `Date` object to represent the date of the event
- a constructor to initialize all contained data members
- a `print()` member function to print out the event details
- getter and setter functions for the event date

4. Implement the Calendar class

You will create a new class called `Calendar`. This class will contain all the events in your personal calendar. The `Calendar` class will contain the following:

- a data member for the calendar name, which can be a string; in future work, we could be managing several calendars, for example a work calendar and a school calendar
- a collection of events, represented as an array of `Event` pointers
- the number of events currently in the calendar
- a constructor
- a destructor to clean up the dynamically allocated `Event` objects
- an `add(Event*)` member function that inserts a new event into the array so that it stays in ascending order by date; this will require shifting some events towards the back of the array to make room for the new one
  - the events will be compared using the `Date` class's `lessThan()` member function
- a `print()` member function that prints out all the events in the array to the screen

5. Modify the main() function

You will modify the program so that the `main()` function:

- doesn't declare a date array anymore; instead, it will declare a `Calendar` object
- uses the `Calendar` object and its functions, instead of manipulating the dates or the array directly
- prompts the user to enter all required event information, including the event name and time
- dynamically allocates an `Event` object once the user enters the information for an event, and adds the event to the calendar using functions implemented in previous steps

6. Test the program

- You will modify the `in.txt` file so that it provides sufficient datafill for a minimum of 15 events. The ordering of event dates and times in the file must be such that the program is thoroughly tested. Do **not** order the events in the file already in ascending order!
- Check that the event information and the order is correct when the calendar is printed out at the end of the program.
- Make sure that all dynamically allocated memory is explicitly deallocated when it is no longer used. Use valgrind to check for memory leaks.

Constraints

- your program must follow the existing design and organization of the base code
- do not use any classes, containers, or algorithms from the C++ standard template library (STL)
- do not use any global variables
- do not use structs; use classes instead
- objects must always be passed by reference, not by value
- your classes must be thoroughly documented in every class definition
- all basic error checking must be performed
- existing functions must be reused everywhere possible

Submission

You will submit in *cuLearn*, before the due date and time, the following:

- one **`tar`** or **`zip`** file that includes:
  - all source, header, and data files, including the code provided
  - a Makefile
  - a readme file that includes:
    - a preamble (program and revision authors, purpose, list of source/header/data files)
    - compilation. launching, and operating instructions

Grading (out of 100)

Marking components:

- 15 marks: correct modifications to `Date` class
  - 5 marks: correct definition and initialization/setting of `Time` object
  - 7 marks: correct implementation of `lessThan()` function
  - 3 marks: correct implementation of `print()` function
- 15 marks: correct implementation of `Event` class
  - 5 marks: correct definition and initialization of event name
  - 5 marks: correct definition and initialization/setting of `Date` object
  - 5 marks: correct implementation of `print()` function
- 45 marks: correct implementation of `Calendar` class
  - 3 marks: correct definition and initialization of calendar name
  - 7 marks: correct definition and initialization of `Event` array
  - 10 marks: correct implementation of destructor
  - 20 marks: correct implementation of `add()` function
  - 5 marks: correct implementation of `print()` function
- 25 marks: correct modifications to `main()` function
  - 5 marks: correct prompting for event information
  - 10 marks: create creation and initialization of `Event` object
  - 5 marks: correct addition to calendar
  - 5 marks: correct printing of calendar

Execution requirements:

- all marking components must be called, and they must execute successfully to receive marks
- all data handled must be printed to the screen for marking components to receive marks

Deductions:

- Packaging errors:
  - 10 marks for missing Makefile
  - 5 marks for a missing readme
  - 10 marks for consistent failure to correctly separate code into source and header files
  - 10 marks for bad style or missing documentation

- Major programming and design errors:
  - 50% of a marking component that uses global variables, or structs
  - 50% of a marking component that consistently fails to use correct design principles
  - 50% of a marking component that uses prohibited library classes or functions
  - 50% of a marking component where unauthorized changes have been made to the base code
- Execution errors:
  - 100% of a marking component that cannot be tested because it doesn’t compile or execute in VM
  - 100% of a marking component that cannot be tested because the feature is not used in the code
  - 100% of a marking component that cannot be tested because data cannot be printed to the screen