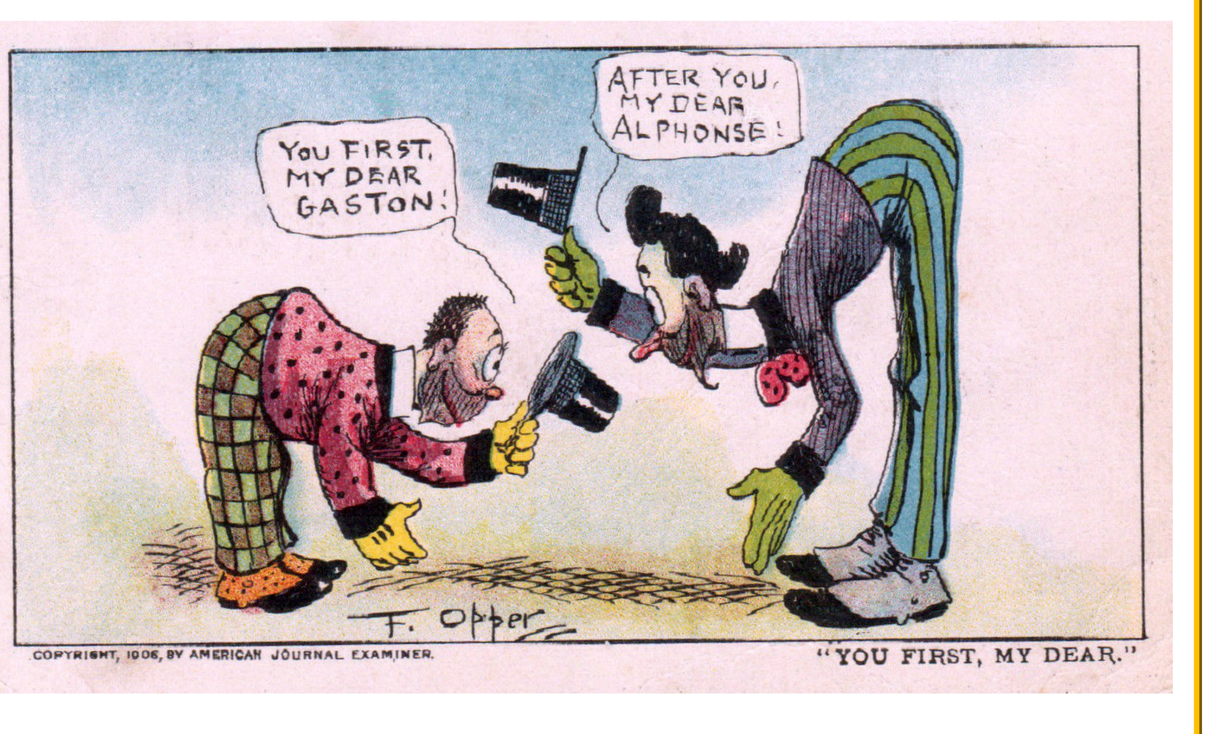
CS253 HW5: Two classes: This time it's classier!



Description

For this assignment, you will write two classes: `Event` and `Schedule`. `Event` represents a single date, per HW3, and `Schedule` is a chronologically-ordered collection of `Event` objects.

For this assignment, you will provide:

- `Event.h`, which contains the interface for `class Event`
- `Schedule.h`, which contains the interface for `class Schedule`
- the library `libhw5.a`, which contains the implementation of those classes.

To use `class Event`, the user must `#include "Event.h"`. To use `class Schedule`, the user must `#include "Schedule.h"`.

Methods

One method is forbidden:

no default ctor
: The default (no-argument) ctor for `Event` must fail to compile. This is not a run-time error; it's a compile-time error.

`Event` must have the following methods:

`Event`(C string)
`Event`(C++ `string`)
: The string must contain an event, in one of the five HW3 formats. Initialize the `Event` to that date. Throw an explanatory `runtime_error`, including the entire bad date argument string, if the date is bad in any way, syntactically (e.g., `12-cucumber`) or semantically (e.g., `2020-02-30`).

Copy ctor
Assignment operator
: Copy the information from the other `Event`.

Dtor
: Destroy.

`.set(int` *year*`, int` *month*`, int` *day*`)`
: Set the date for this `Event` to that date. Throw a `runtime_error`, containing the bad data, and leave the object unchanged, if the date is bad in any way. `.set(2020,5,11)` would set the date to today.

`.year()`
`.month()`
`.day()`
: Return the `int` year, month, or day associated with this `Event`. They have the same values that would be passed to `.set()`.

`Schedule` must have the following `public` methods:

`Schedule()`
: Create a `Schedule` containing no `Event`s.

`Schedule(istream)`
: Read `Event`s from the given stream. All the requirements of `.read()` apply.

`Schedule(C string)`
`Schedule(C++ string)`
: Read `Event`s from the given filename. Throw a `runtime_error`, including the filename, if the file can't be opened. All the requirements of `.read()` apply.

Copy constructor
: Takes another object of the same class, and deep-copies the information, replacing any previous information.

Assignment operator
: Takes another object of the same class, and deep-copies the information, replacing any previous information.

Destructor
: Destroys this object, including all the `Event`s associated with it.

`.read(istream)`
: Read all `Event`s from the `istream`, separated by whitespace, into the `Schedule`. This method does *not* replace previous contents—it adds to them.

  Upon syntactic or semantic error:
  - set the `istream` to a failed state
  - the `Schedule` must contain all the previously-encountered `Event`s in the `istream` up to the point or the error, and no `Event`s from after that
  - the `Schedule` must not contain a half-constructed `Event`.
  - **optionally**: throw a `runtime_error` describing the problem

`.clear()`
: Make this `Schedule` empty. If it's already empty, then make it as empty as a politician's heart.

`.size()`
: Return the number of `Event`s in this object, as a `size_t`.

`.empty()`
: Return `true` iff this object has no `Event`s.

`[size_t]`
: Given a zero-based index, return the corresponding `Event` by constant reference. If the index is out of range, throw a `range_error` (*not* a `runtime_error`), including the erroneous index and the number of `Event`s in this `Schedule`.

  For a `Schedule s`, a zero index must return the `Event` with the earliest date, `s[s.size()-1]` must return the `Event` with the last date, and the `Event`s in between must be in nondescending order. It is not guaranteed that the subscript operator will be called in any particular order.

Non-methods

`ostream << Event`
: Write this `Event` to the `ostream` in `YYYY-MM-DD` format. Write exactly ten characters–nothing else.

`ostream << Schedule`
: Write all the `Event`s in this `Schedule` to the `ostream` in `YYYY-MM-DD` format, each followed by a newline. Write exactly eleven characters per `Event`–nothing else. They must be written in chronological order, oldest events first, as described in the `[]` operator.

For both of the output operators above, use of I/O manipulators such as `setw` or `showpos` may add extra characters, and `hex` will break it entirely. That's ok.

Const-correctness, both arguments & methods, is your job. For example, it must be possible to call `.size()` on a `const Schedule`, or to pass a `const string` to the `Schedule` constructor.

You may define other methods, data, or classes, `public` or `private`, as you see fit. You may create other source & header files, but we will only `#include "Schedule.h"` to use a `class Schedule`, and `#include "Event.h"` to use a `class Event`. Note that it is possible to manipulate an `Event` without a `Schedule`, and it's also possible to manipulate a `Schedule` without dealing with `Event`s.

Testing

You will have to write a `main()` function to test your code. Put it in a separate file, and do **not** make it part of lib`hw5.a`. We will test your program by doing something like this:

```
mkdir a-new-directory
cd the-new-directory
tar -x < /some/where/else/hw5.tar
cmake . && make
cp /some/other/place/test-program.cc .@@
g++ -Wall test-program.cc libhw5.a
./a.out
```

We will supply a `main()` program to do the testing that we want. You should do something similar. I don't mind if your `CMakeLists.txt` builds your test program, as mine does. However, if it does, then be sure to include the source to your test program in `hw5.tar`, or your build will fail.

Sample Run

Here is a sample run, where `%` is my shell prompt:

```
% cat CMakeLists.txt
cmake_minimum_required(VERSION 3.14)
project(Homework)

# Using -Wall is required:
add_compile_options(-Wall)

# These compile flags are highly recommended, but not required:
add_compile_options(-Wextra -Wpedantic)

# Optional super-strict mode:
add_compile_options(-fmessage-length=80 -fno-diagnostics-show-option)
add_compile_options(-fstack-protector-all -g -O3 -std=c++14 -Walloc-zero)
add_compile_options(-Walloca -Wctor-dtor-privacy -Wduplicated-cond)
add_compile_options(-Wduplicated-branches -Werror -Wfatal-errors -Winit-self)
add_compile_options(-Wlogical-op -Wold-style-cast -Wshadow)
add_compile_options(-Wunused-const-variable=1 -Wzero-as-null-pointer-constant)

# add_compile_options must be BEFORE add_executable or add_library.

add_library(hw5 Event.cc Schedule.cc translate.cc)
add_executable(test test.cc)
target_link_libraries(test hw5)

# Create a tar file every time:
add_custom_target(hw5.tar ALL COMMAND tar cf hw5.tar Event.cc Event.h Schedule.cc Schedule.h translate.cc test.cc CMakeLists.txt)

% cat test.cc
#include "Schedule.h"
#include "Event.h"
#include "Schedule.h"       // I meant to do that.
#include "Event.h"
#include <exception>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>

using namespace std;

int main() {
    cout << boolalpha;  // need this for several things
    Schedule s;
    ifstream in("data");
    try {
        s.read(in);
    }
    catch (const exception &e) {
        cout << "Caught: " << e.what() << '\n';
    }
    // That stream should now be failed, but not at eof:
    cout << "fail: " << in.fail() << " eof: " << in.eof() << '\n';

    stringstream iss("   tomoRRow  ");
    s.read(iss);
    auto e = s[0];
    cout << "Oldest: |" << e << "|"
         << " year=" << e.year() << " month=" << e.month()
         << " day=" << e.day() << '\n';
    cout << "size=" << s.size() << " empty=" << s.empty() << '\n' << s;
    s.clear();
    cout << "size=" << s.size() << " empty=" << s.empty() << '\n' << s;
    s.clear();
    cout << "size=" << s.size() << " empty=" << s.empty() << '\n' << s;

    // A poor implementation might have altered cout's fill character:
    cout << left << setw(15) << "All done." << '\n';

    return 0;
}

% cat data
today            yEsTeRdAy
     000000000008.254
2020-01-01 trout 2020.366

% cmake .
… cmake output appears here …
% make
… make output appears here …
% ./test
Caught: Bad date "trout"
fail: true eof: false
Oldest: |0008-09-10| year=8 month=9 day=10
size=5 empty=false
0008-09-10
2020-01-01
2020-05-10
2020-05-11
2020-05-12
size=0 empty=true
size=0 empty=true
All done.
```

Requirements

- When a `Event` is read in HW3 format, whether from a file, stream, or string, it must follow the validity rules from HW3. For example, `23.55` is ok, `23.400` is bad, `0000-01-01` is bad, `TOMorrOw` is ok, etc. `2020-01-32` is bad—it won't get normalized to February 1.
- No method may call `exit()` or produce any output.
- It's ok for a `Schedule` to contain two `Event`s with the same date.
  - If your next thought is "Which one comes first?", well, that's a good thought—keep thinking about the subject.
- In case of a failed `.read()`, the position of the input stream is unspecified.
- You may use the `CMakeLists.txt` shown, or create your own.
- Do not put `using namespace std;` in any header (`*.h`) file. It's fine in an implementation (`*.cc`) file.
- All copies (copy ctor, assignment operator) are "deep". Do *not* share data between copies—that's not making a copy.

- You may not use any external programs via `system()`, `fork()`, `popen()`, `execl()`, `execv()`, …
- You may not use C-style I/O facilities such as `printf()`, `scanf()`, `fopen()`, `getchar()`, `getc()`, etc.
  - Instead, use C++ facilities such as `cout`, `cerr`, and `ifstream`.
- You may not use dynamic memory via `new`, `delete`, `malloc()`, `calloc()`, `realloc()`, `free()`, `strdup()`, etc.
  - It's ok to *implicitly* use dynamic memory via containers such as `string` or `vector`.
- No global variables.
- For readability, don't use ASCII `int` constants (`65`) instead of `char` constants (`'A'`) for printable characters.

- We will compile your program like this: `cmake . && make`
  - If that generates warnings, you will lose a point.
    - If you *still* don't have a `project` directive in `CMakeLists.txt`, then you're not paying attention at all, so you're not reading this.
  - If that generates errors, you will lose *all* points.
- There is no automated testing/pre-grading/re-grading.
  - Test your code yourself. It's your job.
  - Even if you only change it a little bit.
  - Even if all you do is add a comment.

If you have any questions about the requirements, **ask**. In the real world, your programming tasks will almost always be vague and incompletely specified. Same here.

Tar file

- The tar file for this assignment must be called: `hw5.tar`
- It must contain:
  - source files (`*.cc`), including `Event.cc` and `Schedule.cc`
  - header files (`*.h`), including `Event.h` and `Schedule.h`
  - `CMakeLists.txt`, which will create the library file lib`hw5.a`.
- These commands must produce the library lib`hw5.a`:
  `cmake . && make`
- Your `CMakeLists.txt` must use *at least* `-Wall` when compiling.

How to submit your homework:

- Use web checkin, or Linux checkin:

```
~cs253/bin/checkin HW5 hw5.tar
```

How to receive *negative* points:

Turn in someone else's work.