Assignment 3 Part 2

NASA Launch Analysis
You were given a CSV file "`Space_Corrected.csv`" which contains space launch data dating back to the beginning of NASA. Write a program `NASALaunchAnalysis.cpp` that reads the data from this file, extracts the time of day (UTC values), computes the average, and prints it out.

Your program should use the `TimeCode` class to store the time of day values (a `vector<TimeCode>`). The day should be ignored. Only gather the time. Your program should then compute the average time of day that launches take place using the `TimeCode`'s + - * and / operations.

Joke: For motivation you can imagine that someone really important is asking you to do this and if you don't do it then you might receive a warning. Also imagine that after two warnings you'll receive a formal reprimand and after three formal reprimands you may have your annual bonus with-held. If you don't know someone important you can just picture Queen Elizabeth II.

There are some launch instances which are missing any exact time data (only a day). For those you should ignore them completely in your calculation.

If you've implemented `TimeCode` correctly, and tested things well, this program should be relatively easy. The most difficult job will be parsing strings in C++. You may want to write a helper functions such as

`parse_line(string)` Takes a line from the file (a string). Returns the `TimeCode` object for the time embedded in that line.

`vector<string> split(string, delim)` takes a string and a single character (delimiter). Returns a vector containing the sub-strings between any / all occurrences of the delimiter. For example:
`split("some+ random+ words+here", "+") → {"some", " random", " words", "here"}`

Sample Output:
```
user@machine$ nasa
4198 data points.
AVERAGE: 12:7:56
```

Paint Dry Timer
~~Imagine you are working at a toy company that produces balls of various kinds; ball-pit balls for children's playrooms, rubber bouncy balls, basketballs, kick-balls, etc. The balls are produced using different materials and many of them need to be painted. Depending on the size of the ball the paint takes a different amount of time to dry. Your task is a write a program to help the paint technicians keep track of the balls currently drying since they work on many different batches at once.~~

Imagine you're living in a hypothetical situation in which it is necessary to time things.

Write a program `PaintDryTimer.cpp` which computes *and tracks* the time for batches of ~~painted balls to dry~~ things. One ~~ball~~ thing takes `n` seconds to dry, where `n` is the surface area of the ~~ball~~ thing (assume a sphere) in centimeters. Of course an entire batch dries at once, so a batch of one ~~ball~~ thing or ten-thousand will take the same amount of time.

Your program will use `time(0)` to get the current time. A duration can be calculated as shown below.

```
time_t start = time(0);

...
// a lot of code / time passes here
...

time_t end = time(0);
time_t duration = (end - start); // gives duration from start to end in seconds.
```

Your program will use this trick (as well as your `TimeCode` object) to display the drying status in *real time*. Measuring the time of the program like this is called "profiling." Although this is a crude implementation of profiling. Anyway, see the program output below.

```
user@machine$ ./pdt
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: a
        radius: 2
        Batch-1804289383 will dry in: 0:0:50
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        Batch-1804289383 (takes 0:0:50 to dry) time remaining: 0:0:41
        1 items being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: a
        radius: 2.5
        Batch-846930886 will dry in: 0:1:18
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: a
        radius: 1.75
        Batch-1681692777 will dry in: 0:0:38
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        Batch-1804289383 (takes 0:0:50 to dry) time remaining: 0:0:26
        Batch-846930886 (takes 0:1:18 to dry) time remaining: 0:1:8
        Batch-1681692777 (takes 0:0:38 to dry) time remaining: 0:0:33
        3 items being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        Batch-1804289383 (takes 0:0:50 to dry) time remaining: 0:0:7
        Batch-846930886 (takes 0:1:18 to dry) time remaining: 0:0:49
        Batch-1681692777 (takes 0:0:38 to dry) time remaining: 0:0:14
        3 items being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        Batch-846930886 (takes 0:1:18 to dry) time remaining: 0:0:37
        Batch-1681692777 (takes 0:0:38 to dry) time remaining: 0:0:2
        2 items being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        Batch-846930886 (takes 0:1:18 to dry) time remaining: 0:0:31
        1 items being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: q
```

User waited real-world time (~20 seconds) before entering "v" again.

Starter code has been provided to make the job go faster.

A key aspect of this assignment is to ensure that the `TimeCode` objects aren't leaked. They should be allocated on the heap (one for each batch) and they should be de-allocated (`delete`'d) when the time remaining for the batch reaches 0. At that point the batch should also be removed from the tracker.

You can tell you don't have any memory leaks using `valgrind` (see below). You can imagine in the aforementioned hypothetical situation that this is important for some reason that is personal to your hypothetical identity.

```
user@machine$ valgrind ./pdt
==54479== Memcheck, a memory error detector
==54479== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==54479== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==54479== Command: ./pdt
==54479==
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: a
        radius: 2.0
        Batch-1804289383 will dry in: 0:0:50
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: a
        radius: 2.5
        Batch-846930886 will dry in: 0:1:18
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        Batch-1804289383 (takes 0:0:50 to dry) time remaining: 0:0:40
        Batch-846930886 (takes 0:1:18 to dry) time remaining: 0:1:16
        2 batches being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        Batch-846930886 (takes 0:1:18 to dry) time remaining: 0:0:24
        1 batches being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: v
        0 batches being tracked.
Choose an option: (A)dd, (V)iew Current Items, (Q)uit: q
==54479==
==54479== HEAP SUMMARY:
==54479==     in use at exit: 0 bytes in 0 blocks
==54479==   total heap usage: 31 allocs, 31 frees, 75,848 bytes allocated
==54479==
==54479== All heap blocks were freed -- no leaks are possible
==54479==
==54479== For lists of detected and suppressed errors, rerun with: -s
==54479== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Actually memory leaks *are* an important problem. One example is the web browser. When you use your web browser, memory is allocated for each tab. When you use it for a long time you might close tabs and that memory should be freed (`delete`'d). But, if the programmers forgot to do the free-ing then your browser will slowly consume more and more system memory (RAM) over time. This is a complete waste because the tabs are gone, but the memory for them is still reserved! Eventually the computer will slow down, the browser may even crash. Young people today rarely if ever close tabs, so the problem is less of an issue. After all, why close a tab when you might need it again in a few months or years?

Some Details

- Your program should have a Makefile that compiles all three programs as such:
  - `TimeCodeTests.cpp -o time-code`
  - `NasaLaunchAnalysis.cpp -o nasa`
  - `PaintDryTimer.cpp -o pdt`
- The batch ID numbers are just `rand()` numbers. The system might give two batches the same ID number. I don't care :)
- You should consider writing more test code in `tests()`, of course, I will when grading.
- Bonus points may be awarded for submissions which win national or international awards.

Submission: Your program will be submitted using canvas and git + github.

1) You should already have a private "HW3" repo with `fmresearchnovak` as a collaborator.

2) Make a new commit to the repo before the submission deadline for part 2.

3) On canvas you should upload a link to this github repository
`https://github.com/fmresearchnovak/HW3.git`

I will grade the final commit made before the due-date. This means that you can actually submit on canvas at any time before the deadline, and still make changes (new commits) to the code.