Preview of Project 4: Minimum Overlap Coverage

Instructions: For this project we are trying to find the Minimum Overlap of sets that still cover all points. For example, say we have numerous group projects for our class that needed to be presented at the end of the semester. Your professor must grade every student at least once, but does not mind grading multiple projects. Due to the nature of the projects, students are members of multiple projects, but only want to present once. How can we minimize the number of times a student has to present? Naturally, everyone in a group has to present (no partial group presentations.)

Files:

- Each file in the data directory contains groups of sets, which represent the groups that *may* present.
- src/main.cpp is a general purpose test program for running test cases on your code. It also produces timings.
- src/MinimumOverlap.hpp is the header file for your class that will compute the minimum overlap.
- src/MinimumOverlap.cpp is the source file for your class that will compute the minimum overlap.

Set/Group File: Each set/group file contains multiple sets. Each set is contained on one line and is space delimited list of numbers. Each number represents a student's ID. For Example:

```
0 2 5
0 2 3
1
2 3 6
1 4 3
5 6
```

```cpp
#ifndef MinimumOverlap_HPP
#define MinimumOverlap_HPP

#include <string>
#include <vector>

class MinimumOverlap {
  public:

      // Passes in a string pointing to the set/group file.
      // Make sure you store all the sets/groups!
      MinimumOverlap(const std::string &setFile);

```

```
        // Returns the minimum number of overlaps possible given a cover.
        // If print is true, it prints the set(s).
        unsigned int findMinimumOverlap(const std::vector<int> &cover, const bool &print) co
};

#endif
```

Algorithm Setup:

- SOFAR represents the sets you have chosen
- INPUT represents the student/IDs that still need to be chosen
- print is for your convenience. I recommend printing debug information with print = true.
- min represents the minimum number of overlap found so far in a cover
- RESULT is what is left over from removing the evaluating set

```
unsigned int findMinimumOverlap (SOFAR, INPUT, const bool &print) const {

    unsigned int count = 0;
    unsigned int min = UINT_MAX;

    for each set S in the set file {
        RESULT = INPUT subtract S
        if (RESULT has 0 members left) {//found a cover
            if (RESULT has fewer sets than min) {
                min = the size of RESULT.
            }
        } else {//didnt find a cover yet, so try with what is left
            add (append) S to the end of SOFAR
            // Recursively call with S
            int cmin = findMinimumOverlap(SOFAR, RESULT, print);
            // If the new min that was found is smaller, then update min.
            if (cmin < min) {
                min = cmin;
            }
            remove S off the end of SOFAR so we can evaluate the next S.
        }
    }
    return count;
}
```

As usual, you may not change the public interface but may add any private data and methods. Since we are using recursive backtracking you **must** add at least one additional method for the recursion to work. Keep in mind the above is pseudo-code (I never defined SOFARs type.)

Notice that data stored in the private section SHOULD NOT depend on a particular call to find-MinimumOverlap but only on the set/group file. Other data should be declared locally within findMinimumOverlap and passed as parameters when needed. Pass by reference is faster than pass by value for any data items bigger than a single integer so consider using that if you dont change the value. Keep in mind that const does give some performance improvement, but not much.

You may create as many helper classes as you would like. The public interface of those classes is completely up to you as well, subject to good design criteria. In particular, you will need have a representation (possibly as a class or struct) to represent a set/group. You may use anything from the STl or roll your own.

Example Runs:
To be written...

Example Timings:
To be written...

Memory Management:
Now that we are using new, we must ensure that there is a corresponding delete to free the memory. Ensure there are no memory leaks in your code! Please run Valgrind on your tests to ensure no memory leaks!

STL:
You may use anything from the STL.

How to turn in:
Turn in via GitHub. Ensure the file(s) are in your directory and then:

- $ git add <files>
- $ git commit
- $ git push

Due Date: December 2, 2017 2359

Teamwork: No teamwork, your work must be your own.