MTH 4300
Spring 2025
E. Fink

Homework 5

---

1) student.cpp

Create a class definition for a class called `Student`. Each `Student` has 10 homework assignments they have to submit.

The objects of this class should have the following *public member variables*:

- `string name`, representing the *name* of the student;
- `int num_submitted`, the number of assignments the student has currently submitted;
- `int scores[10]`, an array which holds the ten homework grades.

The class definition should also contain the following *public member functions*:

- a default constructor (that means it takes no parameters), which sets `name` to be `"?"`, `num_submitted` to be `0`, and doesn't initialize `scores`
- a constructor which takes one `string` parameter, which initializes `name` (and the other members are initialized as before)
- `void add_score(int)`, which takes an `int` as an outside argument; if `num_submitted` is less than 10, this function should increment this variable, and insert the new score into `scores`. Be on the lookout for off-by-one errors here
- `double average()`, which takes no outside arguments, and returns the average of the number of submitted scores (or 0 if `num_submitted` is 0)
- `void display()`, which takes no outside arguments and returns nothing, but prints out the name and average of a `Student` – this function should call upon `average()`. (To call upon a member function inside of another member function, you can write something like `x = this->average()`, or better yet, just `x = average()`; these would both assign the return value of `average()` to `x`.)

Your class implementation should make my client code work in `main()` work. It should be written in the style that we have introduced: the class definition should only provide member function declarations, with member function implementations outside. Also, access-only member functions should be marked as `const`.

Specifications: your program must

- contain a declaration and definition of the class `Student` that includes all the members listed above, which allow my test code to run properly.
- follow the style we have set forth in class, with the class definition containing only declarations of functions, not implementations, and with functions marked as `const` if appropriate.
- have `display()` call upon the `average()` method.

---

2) spinner.cpp

Before ChatGPT, if you wanted to plagiarize your English papers effectively, a common tool was a *text-spinner*: a program that would take a paper that you had received from the internet, and replace a random sampling of words with synonyms, to make it hard to find the source material (and hence hard to discover that it is plagiarized at all). Of course, the documents that get produced often read like nonsense.

Let's write a simple text-spinner. As part of this, you will build two classes: `SynGroup` (which will represent a single group of words that are synonyms for one another) and `Thesaurus` (which will contain a list of `SynGroup`s).

You have been provided with 50 files, whose names are contained in the variable `string list_of_group_files[];` contained in the `main()` function in `spinner.cpp`. In each file, you have a group of words and phrases that can be synonyms for one another, one on each line.

First, write the declaration and definition for the `SynGroup` class in the file `syngrouptest.cpp`. There is code in the `main()` function in that file that you can use to test this class.

The objects of the `SynGroup` class should have the following *PRIVATE member variables*:

- `string words[20]`, an array which holds up to 20 words/phrases which are synonyms;

- `int length`, representing the actual number of words in the group (which should equal the actual number of entries in `words` that will be filled).

The class definition for `SynGroup` should also contain the following *public member functions*:

- a default constructor (that means it takes no parameters), which sets `length` to be `0`, and doesn't initialize `words`.
- a constructor which takes one `string` called `filename` as a parameter. This constructor should open a file, contained in the working directory of your running program, whose name is contained in the variable `filename`. This constructor should read that file line-by-line, and each line should be stored in its own entry in `words`. The member `length` should also be initialized appropriately. You may want to also include code that sends out some sort of alert if a file is not opened successfully.
- `bool has_word(string)`, which takes a `string` as an outside argument; if this argument is contained in `words`, this should return `true`, otherwise it should return `false`.
- `string replacement(string)`, which takes a `string` called `w` as an outside argument. First, it should be checked if `w` is contained in `words` – if not, the input should be return unchanged. Otherwise, a DIFFERENT word than `w` should be returned, chosen at random. For this, it will help to note that

  ```
  std::uniform_int_distribution<> distr(0, n);
  ```

  can be used in place of our old `uniform_real_distribution` to generate a random integer between 0 and n, inclusive.
- `void display()`, which takes no outside arguments and returns nothing, but prints out all the words/phrases in the group.

Once the test code in `main()` of `syngrouptest.cpp` passes, you should copy and paste your class definition to the next file, `thesaurustest.cpp`.

Next, write the declaration and definition for the `Thesaurus` class in the function `thesaurustest.cpp`. There is code in the `main()` function in that file that you can use to test this class.

The objects of the `Thesaurus` class should have the following *PRIVATE member variables*:

- `int entries`, representing the actual number of `SynGroup`s that will be stored in the Thesaurus – in this assignment, that number will be exactly 50, but we'll make it a member variable anyway;
- `SynGroup group_list[50]`, an array which will holds exactly 50 `SynGroup`s. (Note that this illustrates that just like you can have arrays of `int`s or `string`s, you can also have arrays of user-defined classes!)

The class definition for `Thesaurus` should also contain the following *public member functions*:

- a constructor which takes an array of `string`s of length 50 called `filenames` as a parameter. This constructor should go through this array, and create `SynGroup`s for each filename contained in this list. This constructor should also set `entries` to equal 50.
- `string produce_syn(string)`, which takes a `string` named `w` as an outside argument. The function should return a synonym for `w` if `w` appears in at least one `SynGroup` in the `Thesaurus`; if `w` does not appear in any `SynGroup`, the function should just return `w`.
- `void display()`, which takes no outside arguments and returns nothing, but prints the entire contents of the `Thesaurus` in a reasonable way.

Once the test code in `main()` of `thesaurustest.cpp` passes, you should copy and paste your class definition to the final file, `spinner.cpp`.

Finally, in the `main()` function of `spinner.cpp`, write code which opens up the file `TERM_PAPER.txt`. Your program should read through this file word-by-word; use a `Thesaurus` object to replace each word with a synonym if possible, or leave the word unchanged if not; and write the (possibly unchanged) words into to a file named `SPUN.txt`. The `Thesaurus` object should be created using the supplied files, whose names are contained in `list_of_group_files`.

It is ok if your output file has all the text in the output file on one long line, although there should at least be spaces between each word. Your spinner is allowed to ignore words that begin or end with punctuation, and it does not need to recognize words with capital letters. Also, some of the phrases in our thesaurus will have multiple words; your program does not have to recognize these phrases in the `TERM_PAPER.txt` file, but it should sometimes produce these phrases in `SPUN.txt`.

Specifications: your programs must

- contain a declaration and definition of the class `SynGroup` in `syngrouptest.cpp`, `thesaurustest.cpp`, and `spinner.cpp` that includes all the members listed above, which allow my test code in `syngrouptest.cpp` run properly.

- contain a declaration and definition of the class `Thesaurus` in `thesaurustest.cpp` and `spinner.cpp` that includes all the members listed above, which allow my test code in `thesaurustest.cpp` run properly.
- in `spinner.cpp`, write code which opens `TERM_PAPER.txt`, reads through it word-by-word; replaces each word with a synonym if possible, using a `Thesaurus` object; and writing the (possibly unchanged) words `SPUN.txt`, with spaces between words. The `Thesaurus` object should be created using the `list_of_group_files` array.
- follow the style we have set forth in class, with the class definition containing only declarations of functions, not implementations, and with functions marked as `const` if appropriate.