2020/11/18 - Score Database

17 Tháng Mười Một 2020 9:52 SA

Synopsis

- Go over Lab 8.

- Ok... now we're at the final lab.

Lab 8

- As always, on Canvas, go to the "Lab 8" assignment and read the "lab8.html" file attached. "All" lab details will be there.

Submission Command

- tar -cvf lab8.tar Rank_byname.cpp Rank_byscores.cpp

Rank_byname

- Given a file of first names, last names, and scores, print them out lexicographically sorted by name.

  - std::string does have an operator<, by the way. Seriously. Try it.

  - The name sorting is lastname, then firstname. Sure, you can do this with 2 variables. But you will want to do it in one. And it will save you from some stress when printing (since you are going to use setw and setfill too).

- This is excellent practice for the std::map data structure. My favourite.

- Two classes:
  1. name_t - Stores name information.
  2. scores_t - Stores score information.

- These are thrown into a map, NS, which uses name_t as the key and scores_t as the value.
  - In C++ term: map<name_t, scores_t>.

- WHAT THEY DON'T TELL YOU: A map is a balanced BST (most often, a RED-BLACK TREE). This means it's sorted by key.
  - map<name_t, scores_t>
    - name_t: KEY
    - scores_t: VALUE
  - So, how does it know how to sort a class? It doesn't. You must manually tell it how. It's actually simple. Just implement operator<.
    - Why only operator<? You can get > and == (as well as <=, =>, and !=) from just <. And map takes advantage of this to save you the hassle.
    - It must be const. Thus:

```
bool operator<(const T&) const;
```

- My page should have an introduction to std::map if you are stuck.

★ ANYWAYS, THE LAB...

- You will need these includes: stdlib.h, iostream, iomanip, string, fstream, sstream, numeric (for accumulate), map, vector.

- Behold the following class structure:

```
class name_t {
    public:
        name_t()
        name_t(const string &, const string &);

        bool operator<(const name_t &) const;
        void print(int = 0) const;

    private:
        string name;
};

class scores_t {
    public:
        scores_t();
        void insert(int);
        void insert_done();
        void print();

    private:
        vector<int> scores;
        float mean;
};
```

★ NAME_T

- Default constructor just sets name blank.
- Second constructor takes 2 strings, firstname and lastname and concatenates them into a single string, separated by a comma and space.

  (Ex. name_t("Clara", "Nguyen");

  name = "Nguyen, Clara"

- The operator< just compares the name with another.
  - Remember. Above, I said std::string has an operator< overload. Don't over-complicate a one-liner.
- print(int W) is tricky...
  - Set width to W + 3.
  - Set fill to '.'. Yes, a dot.
  - Set alignment to left.
  - Print name and an additional space.
  - These options are "sticky"... AKA, they persist in future cout calls. This will be a problem...

★ SCORES_T

- Default constructor sets mean to 0.0 i guess.
- insert just pushes a number into scores.
- insert_done just computes mean via scores vector.
  - When doing division, remember scores can be empty. Let's NOT divide by zero...
  - Use std::accumulate if you want.
- print, again, is tricky.
  - Set align to right.
  - Set fill to ' ' (a space).
  - Loop through scores and print.
    - Do a space first. Then set the width to 2. THEN print scores[i].
  - Print the mean.
    - ' : ' (Space-colon-space). Then print the mean with one decimal place.
  - Newline.

No coffee that day...

NGUYEN, CLARA ..... 90 95 92 40 89 : 81.2

NAME_T::PRINT SCORES_T::PRINT

★ MAIN FUNCTION

- The structure is given via comments pretty well.
- Read a file with ifstream.
  - Go by line. Put line in istringstream.
    - BY THE WAY, stop using "stringstream" (without the "i" or "o"). I see it too much in broken code.
  - Extract firstname and lastname from the line.
  - Make a name_t out of those names.
  - The rest of the stuff in the istringstream is scores. While-loop » them in to a scores_t.
  - Insert into the NS mentioned earlier... but only if the name isn't already in NS.
    - map<name_t, scores_t> NS;
    - map class has its own find function. Simply chuck your name_t in it and see if it returns NS.end() to tell if name was inserted before.
    - When inserting, use make_pair:

      (Ex. NS.insert(make_pair(n, s));

      NAME_T ↑ ↑ SCORES_T
- Done reading lines? Close the file.

- Use an iterator to loop through NS.
  - Map iterators have "->first" to access key and "->second" to access value.
  - Call each name_t and scores_t print function.

RANK_BYSCORES

- Welp. no more std::map. =/
- Instead, now going to use a vector of namescores_t (a new class storing a name_t/scores_t pair) as a max_heap.
  - Conveniently, #include <algorithm> has make_heap and pop_heap. Hmm...
- As the name suggests, we are reporting by score (mean) this time. Use name if there is a tie in means.
- Additional command line argument K. This indicates how many people to print out.

(Ex. ./Rank_byscores 20 5 test.txt
w K FILE.TXT

Prints top 5 scoring students

- Copy your code from Rank_byname.cpp to Rank_byscores.cpp.
- Add a function to scores_t which simply returns the mean. Make this public.

- Behold the following new class namescores_t:

```
class namescores_t {
    public:
        namescores_t();
        namescores_t(const name_t &, const scores_t & );

        bool operator<(const namescores_t &) const;

        void print_name(int = 0);
        void print_scores();

    private:
        name_t    n;
        scores_t  s;
};
```

★ NAMESCORES_T FUNCTION IMPLEMENTATION

- Default constructor does nothing.
- Second constructor simply copies arguments over to n and s.
- The two print functions simply call the print function in either n or s respectively.
  - It should be obvious which is which...
- Operator< compares s.mean with rhs.s.mean (use that function I told you to write which returns mean, as that variable is private).

Two scenarios are possible:

1. The means ARE NOT the same. Simply return whether current mean is less than rhs's.

2. The means ARE the same. Compare names instead. name_t.name is private, but you can use name_t.operator<.

   - The return condition is REVERSED for name printing. Thus, return TRUE if rhs is less than this's name.

     This is because we want scores printed highest-to-lowest, which a max-heap does. But, that means names would go Z, Y, X, ..., C, B, A. We want tied mean names still lexicographical, so flip to force it.

★ MAIN FUNCTION

- Same as before, but obviously with small adjustments.

- Account for new argument order:
  ARGV[1]: W, width of name field
  ARGV[2]: K, up to K people printed
  ARGV[3]: file, scores file to read in

- Replace map<name_t, scores_t> with a vector<namescores_t>.

- Obviously, this means change how the insertion is done while reading lines. Simply use push_back.

- After reading in file and closing it, we will utilise STL to make a max-heap.

  1. Use make_heap on your vector.

  2. This max-heap implementation guarantees that the first element is always the maximum element. Thus, when looping, use vector[0], not vector[i].

     Loop i to K, quit early if vector is empty. Print name + scores.

  3. Post-printing, use both pop_heap and vector.pop_back to pop off the current max and move the next one up to vector[0].

- HUGE HINT: Stuck? Look at the example written on cplusplus.com for make_heap. It's 1-to-1 what you need here.