Update README.md
Everything TA authored 4 weeks ago

Name	Last commit	Last update
README.md	Update README.md	4 weeks ago

README.md

OOP Homework #6

> This homework was created by 黃漢軒 (109590031), please feel free to ask me if you have any questions.
>
> Email: t10950031@ntut.org.tw / MS Teams 黃漢軒

⚠️ Due: 11:59 p.m., 22 / 12 / 2022 ⚠️

Goal

This homework has these goals:

- Know how to apply the polymorphism.
- Know the details concept about virtual.
- Know how to solve the problem with the concept.
- [Optional] Know how to implement anonymous function.

Folder Structure Tree

- You should finish the unit test written by you.
- You can split the unit test into multiple files, just remember to include all of it into `ut_main.cpp` (see course repo).

While your project has been built by `makefile`, the structure tree should be the same as the following section.

```
bin/
├─ ut_all
src/
   ├─ alcohol.h
++ ├─ algorithm.h
++ ├─ order.h
   ├─ sour.h
   ├─ tequila.h
test/
   ├─ <some test file>
   ├─ ut_main.cpp
makefile
```

Problem Content

Ah-ha, another Christmas is coming.

Ah-ha, yet another drunkard-love problem coming too.

The bar which run by Uriah will need to handle a lot of people, which want to buy a drunk in Christmas.

So Uriah want to add some algorithm to help him sort the order, or get the min-max of the order.

He want to add 4 function into his system, for example:

- The sort function, sort the alcohol by alcohol content
- The sort function, sort the alcohol by alcohol name. (with lexicographical order)
- A max function, return the alcohol with greatest alcohol content.
- A min function, return the alcohol with smallest alcohol content.

Also, you should complete `Order` class, to let it describe the order.

Task

In this task, you should complete the `algorithm.h` and `order.h`, and use `sour.h`, `tequila.h`, `alcohol.h` in pervious homework.

- `algorithm.h`
  - `void sort_alcohol_by_content(vector<Alcohol*> &alcohol)`
    - Sort the alcohol vector by alcohol content.
    - If the alcohol content is equal when compare, name will be the second priority (with lexicographical order).
  - `void sort_alcohol_by_name(vector<Alcohol*> &alcohol)`
    - Sort the alcohol vector by alcohol name. (with lexicographical order)
  - `Alcohol* get_greatest_content_of_alcohol(vector<Alcohol*> alcohol)`
    - Get the alcohol with greatest alcohol content.
    - If the alcohol content is equal when compare, name will be the second priority (with lexicographical order).
  - `Alcohol* get_smallest_content_of_alcohol(vector<Alcohol*> alcohol)`
    - Get the alcohol with smallest alcohol content.
    - If the alcohol content is equal when compare, name will be the second priority (with lexicographical **reverse order**).
- `order.h` with `class Order`
  - `void append_alcohol(Alcohol* alcohol)`
    - Append the alcohol to the alcohol list.
  - `int get_alcohol_count()`
    - Return the count of alcohol list.
  - `int get_total_price()`
    - Return the total price of alcohol list.
- More details of lexicographical order:
  - lexicographical order: `[AB, ABC, DEF, ZXC, ZZZ]`
  - lexicographical **reverse order**: `[ZZZ, ZXC, DEF, ABC, AB]` (Just reverse the lexicographical order.)

Test

In this homework, you should use `gcovr` tool to make sure your *code coverage* in `/src` is all above 90%.

- If your lines of *code coverage* are below 90%, you will receive `FAILURE` in the HW Job.

	Exec	Total	Coverage
Lines:	85	85	100.0%
Functions:	32	32	100.0%
Branches:	42	43	97.7%

File	Lines		Functions		Branches	
alcohol.h	100.0%	17 / 17	100.0%	7 / 7	100.0%	15 / 15
algorithm.h	100.0%	17 / 17	100.0%	6 / 6	100.0%	6 / 6
order.h	100.0%	15 / 15	100.0%	5 / 5	83.3%	5 / 6
sour.h	100.0%	18 / 18	100.0%	7 / 7	100.0%	8 / 8
tequila.h	100.0%	18 / 18	100.0%	7 / 7	100.0%	8 / 8

**You will get the 35% score if HW Job passed, otherwise, you will lose the 35% score if HW Job failed. **

See the course slide (`OOP_gcovr.pptx`) to know how to install and how use it.

Notice

- Use nullptr if you want to have a null pointer, which is a special pointer that doesn't point to anything.
- Use `ASSERT_EQ` to test integers, `ASSERT_NEAR` to test floating-point numbers, and `ASSERT_THROW` to test exceptions.
- You should neither add a bin folder to your git nor add a file with the name '.gitignore' in the bin folder (see our class repo).
- In some situations you will lose score:
  - **You lose 5 points for each test that has a memory leaks. You can check memory leak with `valgrind` cmd.**

    ```
    valgrind --track-origins=yes --leak-check=all <executable_file>
    ```

  - **You will lose 10% if your bin folder contains compiled ut_all in the git repo.**

Meme