CSCI3180 Assignment 1: Hellish Teemo Ramen

Image source: Wikipedia page on 地獄ラーメン by Richard (CC BY-SA 2.0)

Introduction

Teemo makes ramen for a living. He is famous (and infamous) for his super spicy ramen.

In this assignment, we are going to write a C++ program to simulate a few days at the *Hellish Teemo Ramen* restaurant proudly owned by him.

The main objectives of this relatively simple assignment are three-fold, for you:

- To have a taste of a CSCI3180 assignment, in particular, its styles, policies, and submission system.
- To practice C++ programming which will be used in the next assignment as well.
- To explore the pros and cons of explicit heap-dynamic variables (i.e. usage of heap-dynamic memory).

About academic integrity

We value academic integrity very highly.

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters.
- The penalty (for BOTH the copier and the copiee) may not be just getting a zero in your assignment. There may be additional disciplinary actions from the university, upto and including expulsion.
- Do NOT copy or look at the assignment code/work of another student/person.
- Do NOT share your assignment code/work to anyone and do NOT post it anywhere online.
- While we encourage discussion among students, you should write the code and finish the work on your own. In the discussion, you should NOT go into the details such that everyone will end up with the same code/work.
- You are responsible to double-check your own submission and do your best to avoid "accidents".

Read https://www.cuhk.edu.hk/policy/academichonesty/ for details of the CUHK policy on this matter.

Task overview & download

You have two main tasks in this assignment.

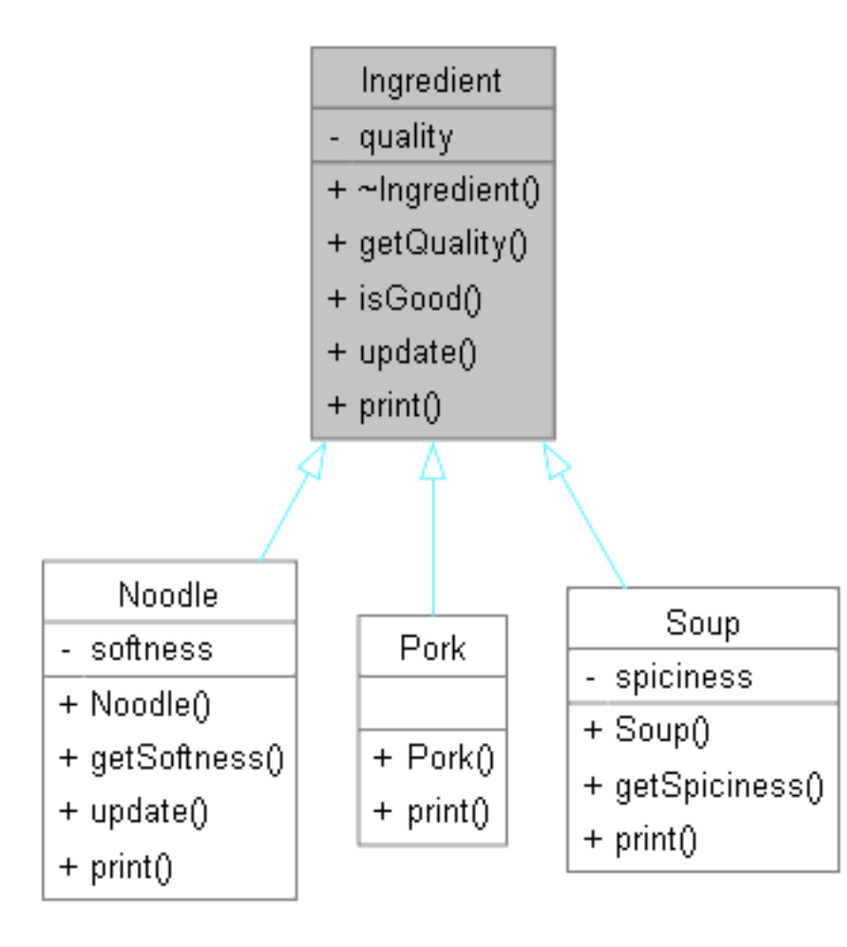
1. First, download the starting skeleton code HERE. Please download it now, as we may refer to it in the description below. Most functions have already been completed for you. Your programming task is to read the description, understand the given code, and complete the 3 incomplete functions marked with TODO in the given skeleton code files. You only need to modify and submit two files: **noodle.cpp** and **ramenRestaurant.cpp**. Basically, all other files should not be submitted or modified. We will provide the specifics in the Submission & deadline section.
2. Write a report on heap-dynamic memory pros/cons, and save it as a PDF file. We will talk about it in details after the programming task description.

Read the pinned assignment 1 FAQ post on Piazza for some common clarifications. You should check that a day before the deadline to make sure you don't miss any clarifications, even if you have already submitted your work then.

If you need further clarifications of the requirements, please feel free to post on the Piazza with the assignment1 tag. However, to avoid cluttering the forum with repeated/trivial questions, please do read all the given code, webpage description, sample output, and latest FAQ carefully before posting your questions. Also, please be reminded that we won't debug for any student's assignment for the sake of fairness and that you should not share/post your solution code anywhere.

The Ingredient class

It is the abstract base class (ABC) for ramen ingredients. We only have 4 classes in this class hierarchy and `Ingredient` sits at the top. To give you an overview of the class design, here is the class diagram:



In the diagram, + denotes the public members while - denotes the private members.

Details will be provided below. For anything that appears to be unclear to you, you should refer to the given code if they are available. First, let's look at this ABC: the `Ingredient` class.

Data members

```
int quality = 100;
```

This is the quality of the ingredient. It is always set to 100 initially and, as it degrades over time, will decrease by 1 per update which will be described later.

Member functions

```
virtual ~Ingredient();
```

This is the destructor. It doesn't really do anything in this class, but as a good practice we should always declare it virtual in an ABC. If you are curious why, think about what may happen if the base class destructor is not virtual when we delete a derived class object via a base class pointer.

```
int getQuality();
```

It returns the quality value.

```
bool isGood();
```

It returns whether the ingredient is still good to use, meaning it has a positive quality.

```
virtual void update();
```

Every hour in our simulation, the `update` function of all ingredients will be called to update their states. For all ingredients, their quality values have to each be decreased by 10 (lower-bound by 0; please check the given code for its exact meaning). For specific ingredients, the `update` function may be overridden to specify additional state updates required.

```
virtual void print() = 0;
```

It prints the information of the ingredient. It is made pure virtual in this ABC and shall be overridden in all concrete derived classes.

The Pork class

It is a piece of pork meat. It is a concrete derived class of `Ingredient`. Each Hellish Teemo Ramen requires one or two of it, depending on the customer's order.

Member functions

```
Pork();
```

It is the constructor. It doesn't do anything special. In fact, we can also choose not to implement it. However, if a declaration of it is made in the class definition, then you must provide an implementation (even an empty one) for it.

```
void print() override;
```

It prints the information of the pork meat. You should read the given implementation and the sample output to see what it prints.

The Soup class

It is a bowl of ramen soup. It is a concrete derived class of `Ingredient`. Each Hellish Teemo Ramen requires one of it.

Data members

```
int spiciness;
```

This is the spiciness of the soup.

Member functions

```
Soup(int spiciness);
```

It is the constructor. It sets the spiciness value.

```
int getSpiciness();
```

It returns the spiciness.

```
void print() override;
```

It prints the information of the soup. You should read the given implementation and the sample output to see what it prints.

The Noodle class

It is a bowl of noodle. It is a concrete derived class of `Ingredient`. Each Hellish Teemo Ramen requires one of it.

Data members

```
int softness;
```

This is the softness of the noodle. It is set by the constructor and will also decrease every update.

Member functions

```
Noodle(int softness);
```

It is the constructor. It sets the initial softness.

```
int getSoftness();
```

It returns the softness value.

```
void update() override;
```

Since a bowl of noodle is an ingredient, its quality must also be updated accordingly in an update. Additionally, its softness is decreased by 5 (lower-bound by 0), since the noodle becomes harder and harder over time.

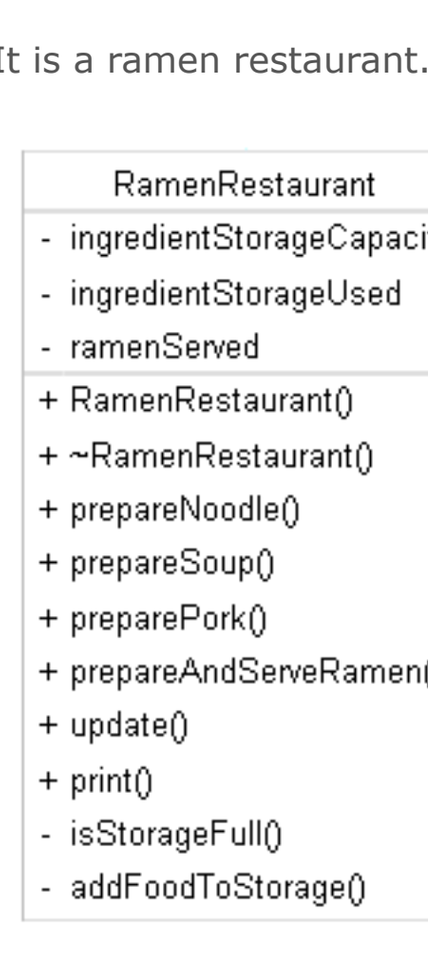
You need to implement this function.

```
void print() override;
```

It prints the information of the noodle. You should read the given implementation and the sample output to see what it prints.

The RamenRestaurant class

It is a ramen restaurant. Specifically, it models how the Hellish Teemo Ramen restaurant is run.



Data members

```
Ingredient** ingredientStorage;
```

It models the ingredient storage of the restaurant. The storage can store pork, soup, and noodle ingredients.

Implementation-wise, this is a dynamic array of `ingredient` pointers which each point to either a `Pork` dynamic object, a `Soup` dynamic object, or a `Noodle` dynamic object, or nothing (i.e. `nullptr`).

```
int ingredientStorageCapacity;
```

This is the ingredient storage capacity. That is, the maximum numbers of ingredients the storage can hold. That essentially is the size of the `ingredientStorage` array. We can also say that the ingredient storage has `ingredientStorageCapacity` storage slots.

```
int ingredientStorageUsed = 0;
```

This is the number of storage slots currently used in the ingredient storage for storing pork, soup, and noodle ingredients. Naturally, it will never exceed the `ingredientStorageCapacity`. It is always 0 initially.

```
int ramenServed = 0;
```

This counts how many bowls of ramen have been successfully prepared and served to Teemo's beloved customers. It is always 0 initially.

Member functions

```
RamenRestaurant(int ingredientStorageCapacity);
```

It is the constructor. It initializes the `ingredientStorageCapacity` and `ingredientStorage`.

```
~RamenRestaurant();
```

It is the destructor. All remaining dynamic objects created in this class should be deallocated properly.

You need to implement this function.

```
void update();
```

It updates the states of all ingredients by invoking their own `update` functions. It will be called every hour in the simulation.

```
void print();
```

It prints the information of the restaurant, in particular its storage. You should read the given implementation and the sample output to see what it prints.

```
bool isStorageFull();
```

It returns whether the ingredient storage is full. The storage is full when there is no empty storage slot left.

```
void addFoodToStorage(Ingredient* food);
```

It adds the given ingredient to the first non-empty slot in the ingredient storage. It is used in the `preparePork`, `prepareSoup`, and `prepareNoodle` functions.

```
bool preparePork();
```

It first checks if the storage is full. If yes, it prints a failure message and returns false.

Otherwise, it proceeds to create a new `Pork` object, add that to the first non-empty ingredient storage slot, print a success message, and return true.

```
bool prepareSoup(int spiciness);
```

It first checks if the storage is full. If yes, it prints a failure message and returns false.

Otherwise, it proceeds to create a new `Soup` object of the specified `spiciness`, add that to the first non-empty ingredient storage slot, print a success message, and return true.

```
bool prepareNoodle(int softness);
```

It first checks if the storage is full. If yes, it prints a failure message and returns false.

Otherwise, it proceeds to create a new `Noodle` object of the specified `softness`, add that to the first non-empty ingredient storage slot, print a success message, and return true.

```
bool prepareAndServeRamen(int requiredNoodleSoftness, int requiredSoupSpiciness, bool doublePork);
```

Well, finally, it is time to prepare the ramen!

The challenge is, our beloved customers have specific requests on exactly how their ramen should be made, as specified by `requiredNoodleSoftness`, `requiredSoupSpiciness`, and `doublePork`.

Teemo follows these rules strictly (and so will you ;)):

- A Hellish Teemo Ramen uses 1 *suitable* soup ingredient and 1 *suitable* noodle ingredient. Additionally, if `doublePork` is false, only 1 *suitable* pork ingredient is used. Otherwise, 2 *suitable* pork ingredients are used.
- Regarding what the *suitable* ingredients are:
  - They must be good ingredients. By the way, the bad ingredients just stay in the storage forever because Teemo is apparently too busy. Don't help him to remove those bad ingredients yourself.
  - A *suitable* noodle ingredient must have at least `requiredNoodleSoftness` of softness.
  - A *suitable* soup ingredient must have at least `requiredSoupSpiciness` of spiciness.
- If we cannot gather from the storage all those specific ingredients for the ramen, the function will print a failure message (which can be copied in a comment in the given code) and return false.
- When looking for each of those ingredients, Teemo simply always picks the first *suitable* ingredients - i.e. technically, the smaller the array index the better for each of them.
- After we have found the 3 or 4 *suitable* ingredients, we do the following to actually prepare and serve the ramen:
  - Remove the used ingredients from the storage. That means you should dynamically deallocate the corresponding dynamic objects of `Pork`, `Noodle`, and `Soup`. Then for each of them, set the corresponding `ingredientStorage[i]` to `nullptr` indicating that the originally-occupied storage slot is now empty and thus reusable later for new ingredients.
  - Since 3 or 4 ingredients have been used, decrease `ingredientStorageUsed` by 3 or 4.
  - Add 1 to `ramenServed` as our customer enjoys our ramen happily (or angrily if the soup is too spicy - notice the soup can be very spicy since the `requiredSoupSpiciness` doesn't specify the maximum spiciness!).
  - Print a success message (which can be copied from a comment in the given code).
  - Return true.

You need to implement this function.

Sample output & grading

Your finished program should produce the same output as our sample output. Please note that sample output, naturally, does not show all possible cases. It is part of the assessment for you to design your own test cases to test your program making sure it works in all scenarios per our description. Be reminded to remove any debugging message that you might have added before submitting your code as program grading is **automatic and purely output-based**.

For this assignment, we will also check if your program has any memory leak. We will be using the following command to compile and run your program under Linux with a gcc compiler, assuming your source files are located at the current directory.

```
g++ -std=c++20 -fsanitize=address,leak -o program *.cpp && ./program
```

It will report detected memory leak as runtime errors when you run the program (i.e. extra output indicating that there is a memory leak). If you want to learn about it, you may read this manual. With this said, we will still try to grade your program output (without leak checking, that is, with command `g++ -std=c++20 -o program *.cpp && ./program`) after applying some mark penalty even if there is a memory leak detected, so do your best in other tasks anyway if memory leak is too hard to fix for you. Please note that, on the other hand, if your program cannot even be compiled or simply crashes on our grading environment, no mark can be given. Be aware that C++ programs may behave differently in different environments. For grading, we will only consider your program output on our grading machine. Therefore, while you are free to work out your solution in any environment, **do test it in an environment similar to what we will use for grading, and by all means avoid compilation errors and crashes** before you submit your work. Program stability is king. For testing, you may use SSH to connect to a Linux machine as demonstrated in tutorial #3.

Written report

Write a report to comment on the pros and cons of heap-dynamic variables (i.e. usage of heap-dynamic memory) in one PDF file named exactly **a1.pdf**, with at most four A4 pages, containing two main sections:

1. Advantages of using heap-dynamic variables
2. Disadvantages of using heap-dynamic variables

For each of these, your explanation and examples **must be based on the design and code of this assignment**. However, you don't need to restrict yourself to use only the current assignment design and written code - you can suggest a modified/extended design or hypothetical scenarios in the context of this assignment. For example, you can do a what-if analysis on using a stack-dynamic variable (`Ingredient ingredientStorage[10];`) instead of a heap-dynamic variable (`Ingredient** ingredientStorage`) for representing the ingredient storage in this assignment, and assume that the Ingredient is not an abstract class so that stack-dynamic variable may be used.

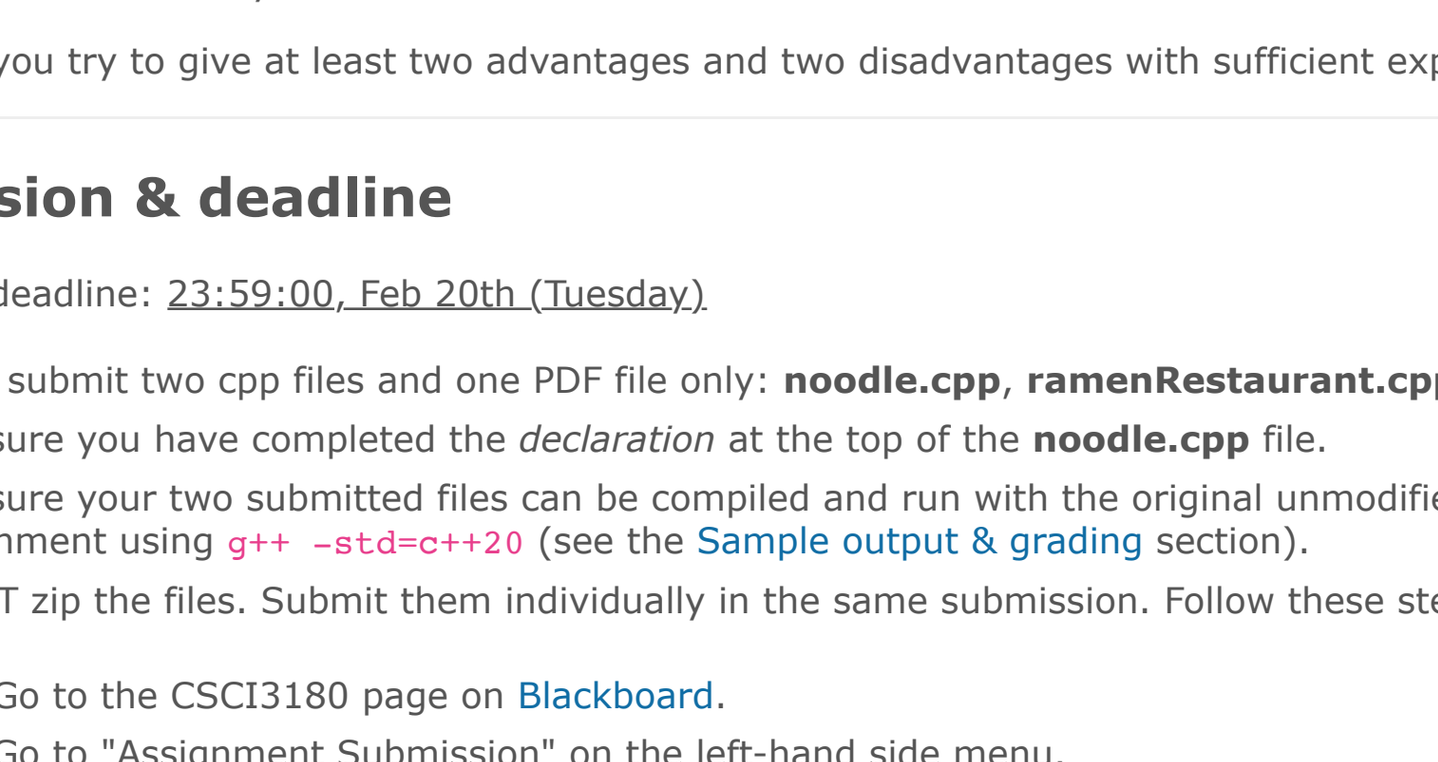
We suggest you try to give at least two advantages and two disadvantages with sufficient explanation in the context of this assignment for a good score.

Submission & deadline

Submission deadline: 23:59:00, Feb 20th (Tuesday)

- Please submit two cpp files and one PDF file only: **noodle.cpp**, **ramenRestaurant.cpp**, and **a1.pdf** ONLY.
- Make sure you have completed the *declaration* at the top of the **noodle.cpp** file.
- Make sure your two submitted files can be compiled and run with the original unmodified version of the other skeleton code files under our grading environment using `g++ -std=c++20` (see the Sample output & grading section).
- Do NOT zip the files. Submit them individually in the same submission. Follow these steps:

  1. Go to the CSCI3180 page on Blackboard.
  2. Go to "Assignment Submission" on the left-hand side menu.
  3. Pick "Assignment 1".
  4. For each of the files that you need to submit, click "Browse Local Files", and then browse for it.

  5. The files will be added to the "Attached files" list. Please **check carefully their filenames**. They must be exactly the same as the ones specified. Wrongly named files won't be graded as most grading are automatic.
  6. Click the blue "Submit" button at the bottom of the page.
  7. Now you **must verify** all of your submitted files by downloading them back and make sure they are your own and the latest version. You can download each of the files with the download button . Click "Start New" to submit again if you need to modify your submission.

- You can submit as many times as you want, but **only the latest submission attempt will be graded**. You do NOT get to choose which version we grade.
- If you submit after the deadline, late penalty will be applied according to the submission time of your last submission attempt.
- **Submit early to avoid any last-minute problem.** Only Blackboard submissions will be accepted. Again, you can submit as many times as you want, so you may make an early submission first well before the deadline, and update it later.
- Late submission penalty is 1 point per 5-minutes late. Less than 5-minutes late is considered as a full 5-minutes late. However, your score will be lower-bound by 0 after any mark deduction.