Retnuh's Restaurant

Background Information

Retnuh's Restaurant is the hottest new eatery in town. However, Chef Ayzman is worried that his chefs aren't making food as fast as he would like. He needs your help. Your mission—should you choose to accept it—will require you to use your knowledge of data structures and algorithms up until this point to simulate and analyze the operation of Chef Ayzman's kitchen.

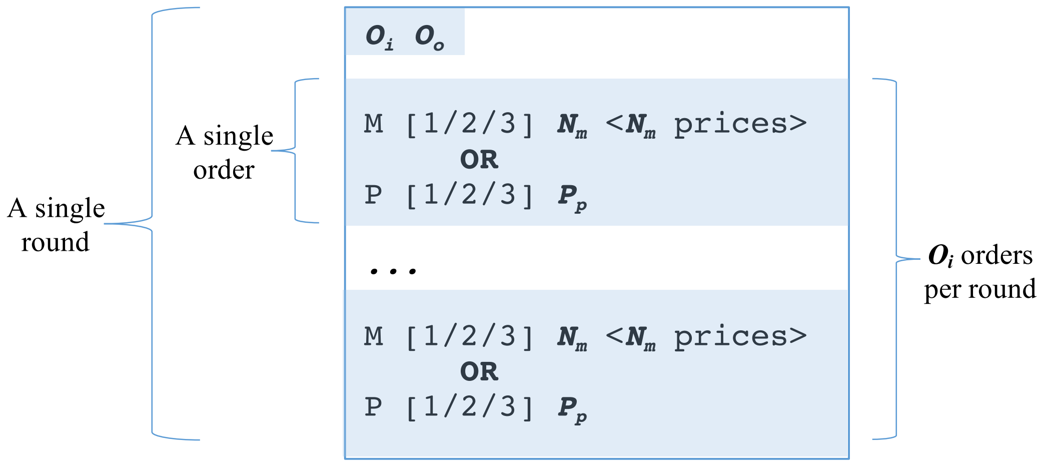
Order Processing

The kitchen processes **orders** from a file. Each order has a **priority**—designated by a value of **1**, **2**, or **3**, with a higher value indicating a higher priority. Each order has a **total price**. Customers can order a **prix fixe selection** or order by **menu item**. When customers order a prix fixe selection, the price of the entire order is one flat fee. This restaurant offers multiple different prix fixe selections, so the prices between each one may vary. When customers order by menu item, the total price of the order is the sum of all the prices of the individual menu items. Each menu items order can contain as many menu items as the customer chooses. Each order has an **id**. The first order in the file has an id of **1**, the second order has an id of **2**, and so on.

The kitchen processes orders in **rounds**. During each round, the kitchen receives a new set of incoming orders and the chef decides how many outgoing orders the kitchen will fulfill before the end of the round. If an order is unable to be fulfilled within a round, it is deferred to the next round, thereby increasing the order's **wait time**. When the kitchen fulfills orders, it cooks the one with the highest priority first. When multiple orders have the same priority, the kitchen chooses the more expensive one first. When multiple orders have the same priority and the same price, you may decide the one to cook first however you wish. Each round has an **id**. The first round in the file has an id of **1**, the second round has an id of **2**, and so on. The id of the round has no impact on the ids of its orders. They increment independently.

Input File Structure

Input files will contain an unknown number of rounds. Rounds conform to the following structure:



```
Oi Oo

M [1/2/3] Nm <Nm prices>
     OR
P [1/2/3] Pp

...

M [1/2/3] Nm <Nm prices>
     OR
P [1/2/3] Pp
```

where O_i represents the number of incoming orders for the current round, and
where O_o represents the number of outgoing orders the kitchen fulfills in the current round, and
where a menu item order begins with an 'M' and a prix fixe selection order begins with a 'P', and
where the type of order is immediately followed by a priority value of 1, 2, or 3, and
where N_m represents the number of menu items for the current menu item order, and
where <N_m prices> represents N_m menu item prices separated by whitespace, and
where P_p represents the price of the current prix fixe selection.

The following input file gives a concrete example of this structure:

```
4 2

P 3 32.00

M 2 2 6.25 3.95

M 3 3 2.25 8.75 3.95

P 1 20.00

3 2

P 2 28.00

P 1 32.00

M 2 1 28.00

0 1

1 1

M 3 3 1.95 1.95 1.95
```

First round: 4 incoming orders and 2 outgoing orders
(1) Prix fixe selection order with a priority of 3, priced $32.00
(2) Menu items order with a priority of 2, containing
2 menu items priced $6.25 and $3.95
(3) Menu items order with a priority of 3, containing
3 menu items priced $2.25, $8.75, and $3.95
(4) Prix fixe selection order with a priority of 1, priced $20.00

Second round: 3 incoming orders and 2 outgoing orders
(1) Prix fixe selection order with a priority of 2, priced $28.00
(2) Prix fixe selection order with a priority of 1, priced $32.00
(3) Menu items order with a priority of 2, containing
1 menu item priced $28.00

Third round: 0 incoming orders and 1 outgoing order

Fourth round: 1 incoming order and 1 outgoing order
(1) Menu items order with a priority of 3, containing
3 menu items priced $1.95, $1.95, and $1.95

Expected Output

Your program must process the input file and run a simulation of the kitchen's activities for each round. You must display diagnostic information about what is occurring in the kitchen. As each round of incoming orders is sent to the kitchen, you must display the id of the current round and the specified number of incoming and outgoing orders. You must display details about each incoming order in the round—its id, its type, its priority, and its total price. After the kitchen processes the incoming orders, you must display the total price of the round's incoming orders. You must also display details about each order that will be fulfilled by the kitchen during the round—its id, its type, its priority, its total price, the id of the round it originated in, and the number of deferred rounds until it was fulfilled. After the kitchen fulfills the outgoing orders, you must display the kitchen's backlog: the number of unfulfilled orders remaining and their order ids. The order ids must be printed in the order that they will be fulfilled in future rounds from left to right, with the rightmost order predicted to be fulfilled first. (It's only a prediction because an order might come in during the next round that supersedes it.)

The example input file yields the following output:

```
ROUND #1
The kitchen will process 4 incoming orders and fulfill 2 outgoing orders.
*** Incoming Orders ***
ORDER #1 (Prix Fixe). Priority: 3. Total price: $32.00.
ORDER #2 (Menu Items). Priority: 2. Total price: $10.20.
ORDER #3 (Menu Items). Priority: 3. Total price: $14.95.
ORDER #4 (Prix Fixe). Priority: 1. Total price: $20.00.
The total price of this round's incoming orders is $77.15.
*** Outgoing Orders ***
ORDER #1 (Prix Fixe). Priority: 3. Total price: $32.00. Round ordered: #1.
Wait time: 0 rounds.
ORDER #3 (Menu Items). Priority: 3. Total price: $14.95. Round ordered: #1.
Wait time: 0 rounds.
The kitchen has 2 orders (#4, #2) in its backlog.

ROUND #2
The kitchen will process 3 incoming orders and fulfill 2 outgoing orders.
*** Incoming Orders ***
ORDER #5 (Prix Fixe). Priority: 2. Total price: $28.00.
ORDER #6 (Prix Fixe). Priority: 1. Total price: $32.00.
ORDER #7 (Menu Items). Priority: 2. Total price: $28.00.
The total price of this round's incoming orders is $88.00.
*** Outgoing Orders ***
ORDER #7 (Menu Items). Priority: 2. Total price: $28.00. Round ordered: #2.
Wait time: 0 rounds.
ORDER #5 (Prix Fixe). Priority: 2. Total price: $28.00. Round ordered: #2.
Wait time: 0 rounds.
The kitchen has 3 orders (#4, #6, #2) in its backlog.

ROUND #3
The kitchen will process 0 incoming orders and fulfill 2 outgoing orders.
*** Outgoing Orders ***
ORDER #2 (Menu Items). Priority: 2. Total price: $10.20. Round ordered: #1.
Wait time: 2 rounds.
ORDER #6 (Prix Fixe). Priority: 1. Total price: $32.00. Round ordered: #2.
Wait time: 0 rounds.
The kitchen has 1 order (#4) in its backlog.

ROUND #4
The kitchen will process 1 incoming order and fulfill 1 outgoing order.
*** Incoming Orders ***
ORDER #8 (Menu Items). Priority: 3. Total price: $5.85.
The total price of this round's incoming orders is $5.85.
*** Outgoing Orders ***
ORDER #8 (Menu Items). Priority: 3. Total price: $5.85. Round ordered: #4.
Wait time: 0 rounds.
The kitchen has 1 order (#4) in its backlog.
```

Extra Credit

1. **How did we do?** [3 points]. After your program finishes displaying all of the required diagnostic information, you must display the following information:

- the total number of rounds and orders
- the total revenue from all the rounds
- the average revenue per round
- the average revenue per order
- the number of prix fixe orders processed
- the number of menu items orders processed
- the average number of incoming orders per round
- the average number of prix fixe orders processed per round
- the average number of menu items orders processed per round
- the average number of outgoing orders per round
- the order ids as they were fulfilled chronologically throughout the simulation
- the number of orders and the order ids that took longer than 2 rounds to fulfill
- the number of orders and the order ids that remained unfulfilled at the end of all the rounds

The following is an example output (independent of the previous input file):

```
The kitchen went through 8 rounds and received 25 orders.
The total revenue was $418.50.
The average revenue per round was $52.31.
The average revenue per order was $16.74.
The kitchen processed 10 prix fixe orders.
The kitchen processed 15 menu items orders.
The kitchen processed an average of 3.13 incoming orders per round.
The kitchen processed an average of 1.25 prix fixe orders per round.
The kitchen processed an average of 1.88 menu items orders per round.
The kitchen fulfilled an average of 2.25 outgoing orders per round.
Orders fulfilled (chronologically): #2, #5, #3, #7, #13, #12, #14, #16, #15,
#8, #18, #6, #19, #11, #20, #22, #21, #17.
The kitchen took longer than 2 rounds to fulfill 4 orders (#8, #6, #11, #17).
The kitchen did not fulfill 7 orders (#10, #4, #24, #9, #1, #25, #23).
```

2. **Output to file** [3 points]. In addition to printing output to the screen, modify your program to send output to a file. The file should be named according to the format '***LastName_InputFileName*.out**' where ***LastName*** is your own last name and ***InputFileName*** is the name of the input file provided as an argument to your program. If no input file name is specified, the default value for ***InputFileName*** is **RetnuhTestInput**.

Program Design Requirement

After your personal assignment repository is created, I will open up a "GitHub issue" in the repository asking you to define the design of your project. You must specify what classes your program will define, what functionality these classes provide, what data structures you will use to store information, how your classes interact with one another, etc. Your design explanation should feature little to no code; instead, it should focus on the macro-elements of your proposed program. I will give you feedback on your design choices so as to help you in the right direction.

Things to Think About

From an architectural perspective, there is no one perfect way to do this assignment. Nevertheless, good design choices can make your solution modular, extensible, efficient, and understandable. Endeavor to strike a balance that makes sense and document your choices—whether in the README file, code, or both. Here are some general principles to think about or take note of:

- From a high level, think about the structures that you'll need and the distinct steps involved in accomplishing the given task. You'll begin to see how some classes need to act mostly as containers of related data and how others need to take responsibility for one or more major steps in the restaurant's workflow. Be modular in your design! A single class should handle a single logical set of responsibilities. Don't make any one class do too much heavy lifting itself. It may seem that having all your actions take place in one class is faster and easier. However, functions ultimately become too long and too confusing over time, making them more difficult to change and debug when necessary. You'll find that separating responsibilities into many classes actually makes function implementations relatively short, readable, and understandable! If you write five or more classes in the course of completing this assignment, you're probably doing something right.
- Certain components in your design will probably share common data and functionality. You should leverage inheritance (and polymorphism) in order to reuse code where it makes sense! Code that properly uses inheritance is quite beautiful for its conciseness and readability; the subclasses are usually quite short, since they let the superclasses do all of the heavy lifting.
- Implementing certain class capabilities can make writing code more fluid, thereby making it more readable/understandable, e.g., overloading the comparison and I/O operators.
- Try to make the format and style of your output conform as much as possible to the sample output's format and style. It makes grading significantly easier and allows me to focus more on reviewing the code itself.
- You may use any data structure that we have covered so far in class, before trees.

Here are some project specific questions you should ask yourself:

- Will you immediately process rounds from the file or store them in some data structure first?
- What data structure makes deciding the next order to be fulfilled easier and/or more efficient?

Submission Details

Your submission must include all the header (*.h) and source files (*.cpp) required for your program to properly compile. Do not submit any generated executables or object files (*.o). You must use/submit the provided makefile—updated only to include the executable file name and the sources to compile. You must properly document your code. You must also update the README file with any guidance that someone looking at and/or running your code needs to (and might like to) know; this includes the program's purpose, compilation instructions, general design patterns, interactions between classes, problems overcome, etc. Self-documenting code is ideal. In instances where there is tricky functionality, specific assumptions, or a disambiguation, a comment is appropriate and necessary.

See the **GitHub Submission Guide** on Piazza for information about how to submit your assignment via git and GitHub. Be sure to ask questions on the Q&A board if you have any issues.

Due Dates

There are two due dates for this assignment. The due date for the program design requirement is Wednesday, March 23rd, before the start of our class (7pm). If you submit the program design late, there is a high chance that I will end up giving you feedback late. If you start coding before I give you feedback, your implementation may suffer as a result. I will not provide feedback to any program designs submitted after March 27th. The due date for the coded portion of the assignment is April 11th, before the start of our class (7pm). These are hard deadlines, meaning that there will be little leniency with regard to lateness. For every day that you miss the deadline for the coded portion, I will deduct five points from the final grade. A sample solution will be provided after April 11th.

Grading

Grades are comprised of the following:

Components	**Percentage**	**Relevant Questions**
Correctness	40%	Does the simulation provide output as expected based on the input? Are orders fulfilled in the correct order? Does the diagnostic information display correct information?
Design	35%	Do you separate out logical units into classes? Is your codebase modular or is functionality too clustered together? Do you use data structures that make your solution efficient?
Documentation	15%	Does your README document provide users with adequate info? Have you adequately commented your class interface files and function implementations (when necessary)?
Style	10%	Is your coding style readable and consistent? Do you follow (to the best of your ability) the modified Google Style Guide?

If the program does not compile, you will receive no points for the **Correctness** component. If you submit the program design after March 27th, you will receive at most half credit for the **Design** component.

Final Words

This assignment might seem overwhelming! It may feel like the whole Republic of CSCI 235 is conspiring against you. Don't feel alone in this endeavor, my padawans. Feel free to discuss design with your fellow students. But please don't be a Rebel; plagiarism is not acceptable. Start early. Good luck, and may the Force be with you.