HOMEWORK #4:

The problem with popplers

Due Date: Friday, December the 2nd, 11:59:59 PM

For this assignment, submit any necessary files, but your `'main()'` function should be in a file called '`popplers.cpp`'. Remember to put your **name** and at the top of your program files. Your program should expect all input to come from '`cin`', and all your output should be to '`cout`'.

Problem:

The problem with **popplers** is that they are delicious! They are so delicious that people are highly likely to want to eat more and more after eating them. Bender has been supplying the ***Fishy Joe's*** restaurant with **popplers** and they are now their most popular item. There is a line of customers waiting even before the restaurant is open.

Customers are served in the order they are in the line. Customers are served as many **poppler** boxes needed to satisfy their wants. Each box has 5 **popplers** in it. Even if a customer only wants one **poppler**, he will have to buy a whole box. Customers eat **all** their **popplers** immediately once they get them, and will go back to the back of the line if they want any more. **Popplers** are so delicious that, after eating each one, there is a **one in six** chance that the customer will want to eat another one.

The ***Fishy Joe's*** restaurant has asked Galaxy Express Software to write a program to simulate the workings of one of their restaurants.

(They are delicious!)

Input:

Your program will read input that describes the line of customers waiting outside of a ***Fishy Joe's*** restaurant. The input starts with a number **N**, the number of customers to simulate. Then **N** lines follow, each with a customer's information. A customer's information consists of a **name**, and the **number of popplers** they want. Customers appear in the order they stand in line outside of the restaurant at opening time. No new customers arrive after opening time.

Output:

Simulate the behavior of the customers in the restaurant, which involves getting their **popplers**, eating them, and **going back to the line** in the case they want more. Output a message indicating when a customer gets and eats **popplers**, when they get back in line for more, and when they leave satisfied. The output **must** be formatted as in the sample output. After all the customer information messages, print the total number of **popplers** eaten, the customer that ate the most **popplers**, and the customer that ate the least. (If two customers ate the same number of **popplers**, print the one that was satisfied first).

Implementation Requirements / Details:

- Use a **queue** Data Structure to simulate the customer line.
- Your queue implementation **must** be a subclass of the provided "`AbstractQueue`" class.
- Your '`main()`' function should be inside a file called '`popplers.cpp`'
- Names of customers **do not** contain spaces.
- To determine if a customer wants another **poppler** after eating one, use the provided "`randomEngine`" class. Instantiate one object of the "`randomEngine`" class and call "`rollD(6)`" for each **poppler** a customer eats. If the returned value is a **6**, the customer will want one other **poppler**. **NOTE:** the behavior of the "`randomEngine`" class can change from computer to computer. The sample output shown here was generated in the campus Linux machines.

Testing:

In this assignment, you **will be penalized** for memory leaks. To test for leaks, you can use the "Valgrind" tool available in the UNIX systems:
After compiling with

```
fg++*.cpp -o popplers.ex
```

run your program with the command

```
valgrind --leak-check=full ./popplers.ex
```

Submission:

Submit your assignment by placing all code in this assignment's git repository in the course's GitLab server, [link] . (You should have a repository setup sometime next week).
Your program will be evaluated and graded on the **Computer Science department's Linux machines** so your program needs to be compatible with the current system, compilers and environment.

Useful Hints:

1. **Carefully** read the comments of each member function.
2. Write down an algorithm for a function before you start coding it.
3. Develop your member functions **one at a time**, starting from the simplest ones.
   Move to the next function only after the previous one has been tested.
   Trying to code the whole class and then remove the bugs may prove to be too big a task.
4. Suggestion: Implement '`operator<<`' and '`enqueue()`' first.

Sample input:

Input
6 Fry 3 Leela 1 Bender 5 Hermes 6 Amy 2 Nibbler 10

Sample output:

Output
Fry eats 5 popplers. Fry is satisfied after eating 5 popplers. Leela eats 5 popplers. Leela is satisfied after eating 5 popplers. Bender eats 5 popplers. Bender wants one more poppler! Hermes eats 10 popplers. Hermes wants 3 more popplers! Amy eats 5 popplers. Amy is satisfied after eating 5 popplers. Nibbler eats 10 popplers. Nibbler wants one more poppler! Bender eats 5 popplers. Bender is satisfied after eating 10 popplers. Hermes eats 5 popplers. Hermes is satisfied after eating 15 popplers. Nibbler eats 5 popplers. Nibbler is satisfied after eating 15 popplers. A total of 55 popplers were eaten. Hermes ate the most popplers: 15 Fry ate the fewer popplers: 5

Using the 'RandomEngine' class:

The following code uses the '`randomEngine`' class to roll a 6-sided dice 25 times:

```
#include <iostream>
#include "randomengine.h"
using namespace std;

int main()
{
  randomEngine re;
  for(int k = 0; k<25; k++){
    cout << re.rollD(6) << " ";
  }
  cout << endl;
  return 0;
}
```

If you run run these code in the campus Linux computers, the output will be:

3 3 5 2 2 2 2 5 3 1 **6** 4 5 3 3 3 1 **6** 4 1 **6** **6** 5 3 1

Which corresponds to the initial output of the simulation.

END.