Project 3: Bill Payer Problem
CPSC 131 Fall 2019

Introduction

Let's assume that you have a fixed income and on the first of every month you receive your paycheck, the same amount every month. You have not sign up for electronic delivery of your bills so you get bills in the mail from time to time, and you have to pay them. The bills arrive in the mail and you collect them on the first of the month. But you don't pay bills the moment you receive them: instead, you place them on your desk, where they become unfulfilled obligations. When you decide to pay a bill, you take one from your desk and pay it.

There are two strategies you can use when you start paying the bills:

- *The stack strategy*: When you get a bill, you place it on the top of your pile of bills. When you decide to pay a bill, you take the one on top, i.e., the one you got most recently, and attempt to pay it. You can pay the bill if you have enough money left in your checking account. If not, then you postpone paying it until later, possibly with penalty. Meanwhile you continue to add bills to the stack as they are received in the mail.

- *The queue strategy*: The bill you pay is the one that's been on your desk the longest. One way to do this is to insert every new bill at the bottom (rear) of the pile, and take from the top (front) of the pile. If not, then you postpone paying it until later, possibly with penalty. Meanwhile you continue to add bills to the queue as they are received in the mail.

Penalty on overdue bills: When a bill is overdue, you have to pay $35 plus 0.1% of the original bill times the number of days is overdue, rounded up to the nearest penny. So if you have a bill of $100 that is due February 10 and you pay it only on March 6, on March 6 you will have to pay $100 + $35 + 0.001*24 days *$100= $137.40. We do not consider leap years, so February has 28 days.

If the bill is not overdue on the payment day, the penalty is zero.

We assume that we do not receive multiple bills for overdue payments. So if we are late in paying a bill, then we pay it with penalty but no overdue bills will be received. If we receive

another bill from the same company, it will not contain the overdue amount, but the amount we owe as the previous bill or bills have been paid.

For each strategy you keep track on how much money you've spent on penalties since the beginning of the year.

Each bill contains the name of the payee, the amount due, and the date when the bill is due. The bills are read from a text file, with the data separated by ‘,’. For example:

```
bill,Chase credit card, 535.49, 1/17
bill,Southern California Edison, 57.89, 1/10
bill,Citibank credit card, 1009.12, 1/18
bill,Republic Services trash, 62.75, 1/13
```

Please note that there is no space before the name of the payee.
The text file may also contain the amount of money deposited when the paycheck arrived:

```
paycheck, 1000.00
```

and the day when we pay bills:

```
pay, 2/1
```

For example, if a file contains the lines below:

```
paycheck, 1000.0
bill,Chase credit card, 535.49, 1/17
bill,Southern California Edison, 57.89, 1/10
bill,Citibank credit card, 1009.12, 1/18
bill,Republic Services trash, 62.75, 1/13
pay,2/1
```

It means that the banking account started with $0, then the paycheck of $1000 made the balance of the account to become $1000. Then four bills arrived. Then you start paying all the bills you could on February 1 (date 2/1) using some strategy, as long as you have enough money to cover a bill and its overdue penalty.

All four bills are overdue on February 1: the first bill by 15 days, the second bill by 22 days, the third bill by 14 days, and the last bill by 19 days.
The fee (aka penalty) for the first bill is computed as $(35 + 0.001 * 15 * 535.49) = $43.04.
The fee for the second bill is $(35 + 0.001 * 22 * 57.89) = $36.28.
The fee for the third bill is $(35 + 0.001 * 14 * 1009.12) = $49.13
The fee for the fourth bill is $(35 + 0.001 * 19 * 62.75) = $36.20.

If we use the stack strategy, then on 2/1 we succeed on paying the last (the fourth bill) plus its fee, but we do not have enough money to pay the next bill (the third bill) and its fee.

If we use the queue strategy, then on 2/1 we succeed on paying the first bill plus its fee, the next (the second) bill plus its fee, and we stopped but we do not have enough money to pay the next (the third) bill and its fee.

Objective

The goal of this assignment is to increase queue concepts and proficiency by making a class that wraps around the standard queue (std::queue) class. Do not implement the queue class, just use the standard library's implementation. Refer to std::queue for a list of class methods.

There are three major domain objects:

- **Bank account** – Encapsulates the following attributes: bank name, the starting amount in your checking account, the amount you have it deposited on the first of every month, and the amount left after all bills who could have been paid have been paid

- **Bill (to be paid)** – Encapsulates the following attributes: the name of the payee, the amount due, and the date when the bill is due.

- **Strategy** – There are two strategies, `QueueStrategy` and `StackStrategy`. Each strategy stores and pays the bill accordingly, using the bank account, and keeps track of how much money you have spent on penalties since the beginning of the year.

Complete the implementation of these classes, and make sure all tests pass. Your code is tested in the provided `main.cpp`.

Initially the given code may not compile. As you complete the code, the tests should start to pass in `main.cpp`.

Source Code Files

You are given complete header and "skeleton" source files with declarations that may be incomplete and without any implementation. Implement the code marked with "To be completed" in the following files and ensure that all tests in `main.cpp` pass successfully.

- `Bank.hpp`: contains the class definition for the class `Bank`; you need to implement all the public functions in a separate file called `Bank.cpp`.

- `Bill.hpp`: contains the class definition for the class `Bill`; you need to implement all the public functions in a separate file called `Bill.cpp`.

- `Strategies.hpp`: contains two classes, `QueueStrategy` and `StackStrategy`; Since these are templated classes, you need to implement all the public functions in the same header file `Strategies.hpp`.

- `main_project3.cpp`: The main function tests your functions. You are encouraged to add additional tests but keep in mind that during auto-grading, your `main.cpp` file will be replaced with the one you were provided with in the starter code.

- `README.md`: You must edit this file to include your name and CSUF email. This information will be used so that we can enter your grades into Titanium.

- Several text files to be used for testing purposes.

Hints

Make sure your code compiles, and then try and solve the logic. Focus on solving one test at a time. It's recommended that you start with Checking.hpp first, then Bill.hpp.

Additional hints are included within sections marked with "To be completed".

Obtaining and submitting code

We will be using GitHub Classroom to distribute the skeleton code and collect your submissions.

Click the assignment link to fork your own copy of the skeleton code to your PC.

Do not fork your repository to your personal github account (instructors have admin access to private repositories under https://github.com/CSUF-CPSC-131-Fall2019/). Your code should have a URL like https://github.com/CSUF-CPSC-131-Fall2019/project3-brians, NOT https://github.com/brian/project3-brians.

https://classroom.github.com/a/9-ZdjrtM

Then edit your code locally as you develop it. As you make progress, commit and push your changes to your repository regularly. This ensures that your work is backed up, and that you will receive credit for making a submission. Don't wait until the deadline to learn how to push code!

Testing

Since you include separate implementation files, your command will need to include the separate implementation files as well (but never the header files):

```
clang++ -g -std=c++14 main.cpp Bank.cpp Bill.cpp -o test
```

To attempt to run the compiled test program, use the following command:
`./test`

Grading rubric

Your grade will be comprised of two parts, Form and Function.
Function refers to whether your code works properly as tested by the main function (80%).
Form refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

Deadline

The project deadline is **Friday, November 8**, 11:55pm.

You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.

Your code must compile/build for it to be tested and graded. If you only complete part of the project, make sure that it compiles before submitting.