Troubleshooting:

If the given project (e.g. Q1) runs with error, you need to run "**Clean and Build Project**" (Shift+F11). If still error, try to rename or copy the project to other one, e.g. from Q1 to Q1X or Q1Y.

The given files already contain statements to implement a program for managing a fruit store. The structure of the main classes is as follows:

- Class **Fruit**: contains information about a Fruit object, including **type**, **amount**, and **price**.
- Class **Node**: includes a Fruit object and a next pointer for linking.
- Class **FruitList**: is **a singly linked list** that manages regular Nodes with complete information. This class plays the role of managing all items in the Store.
- Class **RequestQueue**: is **a queue** structure (implemented as a linked list), where the component data of the Nodes only contains the type and amount. This means that the corresponding request is to purchase an item with a specific type and amount.
- Class **MyStore**: is the main class of the program, containing a **FruitList** used to store data on the items currently available in the store and a **RequestQueue** corresponding to a queue of people wanting to make purchases.

Students are required to carefully read the provided code segments to fully understand the relationships between the classes and the functions within each class. The specific task of the test is to execute the following requirements:

a. f1(): To complete the requirement f1, students need to fulfill two specific tasks: implement the function **addLast()** in the **FruitList** structure and the function **enQueue()** (similar to addLast) in the **RequestQueue** structure. The expected output used to test your code are as follows:

   Data Fruit: (A,9,1500) (B,5,3000) (C,6,5000) (D,2,400) (E,7,9000) (F,4,7000)
   Request : (B,2) (E,5) (M,3) (F,10) (D,2)

b. f2(): To complete the requirement f2, students need to perform two specific tasks: implement the **deQueue()** function of the RequestQueue structure, then use the returned result to execute the **purchase** action for the f2 function. The purchase action can be implemented directly within the body of the f2 function or executed outside of the function by a helper function, which will then be called by f2. The purchase action consists of two steps:
   - First, **search** for the item (**type**) to be purchased. If found, proceed to step 2.
   - If the quantity (**amount**) available in the **FruitList** is **not less than** the quantity to be purchased, perform the purchase action: **update** the remaining quantity in the FruitList .

   The expected output used to test your code are as follows:

   Data Fruit: (A,9,1500) **(B,5,3000)** (C,6,5000) (D,2,400) (E,7,9000) (F,4,7000)
   Request : (B,2) (E,5) (M,3) (F,10) (D,2)
   Data Fruit: (A,9,1500) **(B,3,3000)** (C,6,5000) (D,2,400) (E,7,9000) (F,4,7000)
   Request : (E,5) (M,3) (F,10) (D,2)

   Explaination:

   Initially, the RequestQueue contains 5 elements corresponding to 5 items. After performing deQueue to remove B, the RequestQueue will only have 4 elements. When executing the purchase action, since the request B requires amount=2, while the data in FruitList has amount=5, after completing the purchase action, the remaining amount in FruitList will be 3 (highlighted in red).

c. f3(): Perform the pair of operations **deQueue()** and **purchase** for **all items** in the RequestQueue. The expected output used to test your code are as follows:

   Data Fruit: (A,9,1500) (B,5,3000) (C,6,5000) (D,2,400) (E,7,9000) (F,4,7000)
   Request : (B,2) (E,5) (M,3) (F,10) (D,2)
   Data Fruit: (A,9,1500) **(B,3,3000)** (C,6,5000) **(D,0,400)** **(E,2,9000)** (F,4,7000)
   Request : Empty

   Explaination:

   Since the purchase action is executed for all requests in the RequestQueue, at the end, the

Explaination:

Since the purchase action is executed for all requests in the RequestQueue, at the end, the RequestQueue will be empty. For the requested items, B, E, and D will be purchased and updated amount because the requested amount is less than the amount in the FruitList . However, item M does not exist in the FruitList, and item F requestes amount (amount=10) greater than the available amount in the FruitList (amount=4), so these purchases cannot be perform.

d. f4(): Perform the pair of actions, deQueue() and purchase, for all the items in the RequestQueue and **calculate** the **total revenue**. The revenue (money) from selling an item is equal to the multiplication of the amount purchased and the selling price. The expected output used to test your code are as follows:

Data Fruit: (A,9,1500) (B,5,3000) (C,6,5000) (D,2,400) (E,7,9000) (F,4,7000)
Request : (B,2) (E,5) (M,3) (F,10) (D,2)
Money : 51800

Explaination: There are 5 items in the RequestQueue, where M and F cannot be purchased. The 3 items that can be purchased are B, E, and D. The amount of B, E, and D in the RequestQueue are 2, 5, and 2. The selling prices in the FruitList are 3000, 9000 and 400, respectively. Therefore, the total money is calculated as:

Money= 2x3.000 + 5x9.000 + 2x400= 6.000 + 45.000 + 800 = 51.800