MODULE 3 PROJECT: LINKED-LISTS

Learning objectives:

CLO 1. Identify fundamental data structures in computer science and their use in real-life applications.

CLO 3. Develop problem solving skills by implementing data structures.

CLO 4. Compare advantages and disadvantages of different data structure implementations.

Data Structure Implementations

1. Implement the data structures `SLLStack`, `SLLQueue` and `DLList` covered in the Module 3 (Linked-Lists). All the data structures should be fully functional and must follow the logic presented in the lecture.

   You have to modify the following files:

   - **SLLStack.py**
   - **SLLQueue.py**
   - **DLList.py**

   (Optional) Test your data structures:

   - Remove one element from an empty Stack, Queue, List. This should produce an `IndexError`.
   - Stack: Add 5 elements and remove them and check that they are in opposite order of insertion, e.g., Inserting the sequence 5,4,3,2,1 result in the sequence 1,2,3,4,5 when removing
   - Queue: Add 5 elements and remove them and check that they are in the same order of insertion, e.g., Inserting the sequence 1,2,3,4,5 result in the sequence 1,2,3,4,5 when removing
   - List: Add 5 elements in different positions (including the first and last) and check that they are in order, e.g., `add(0,4)`, `add(0,1)`, `add(1,3)`, `add(1,2)`, and `add(4,5)`. Then, `get(i)` should return `i+1`. Remove 2 elements, e.g., index 2 and 3 and the final list should be "1,2,5".

2. *Background:* A palindrome is a word or expression that reads the same when the characters are reversed (case and punctuation are not considered). Some examples are,

   "Toot" $\rightarrow$ "tooT"

   "Madam i'm adam" $\rightarrow$ "madam 'im adaM"

   Directions: Implement the method `isPalindrome()` in the class `DLList` so that it returns `True` if the elements stored in the linked list form a palindrome, i.e., the element at position `i` is equal to the element at position `n-1-i` for all `i = 0, ..., n-1`. Your code should run in $O(n)$ time.

   You must modify the file **DLList.py**.

   HINT: Traverse the list forward and backward simultaneously.

3. Implement the method `reverse()` in the class `DLList` so that it reverses the whole list. Your code should run in $O(n)$ time.

   You must modify the file **DLList.py**.

   Test your program:

   - An empty list.
   - A list with one element.
   - A list with n elements. For example, "5,4,3,2,1" should return "1,2,3,4,5"

4. Implement a `MaxQueue` data structure as a singly-linked list, that can store comparable elements. `MaxQueue` supports the usual FIFO `Queue` operations `add(x)`, `remove()`, and `size()`. In addition, it supports a `max()` operation, which returns the maximum value currently stored in the data structure. You have to modify the file **MaxQueue.py** that is included in the same Beachboard folder as these instructions.

   Hint: Consider storing elements into a singly-linked list and into a doubly-linked list using the following algorithms:

   - **Adding an element:** When a new element is added to the `MaxQueue`, it is added to the tail of the singly-linked queue, regardless of its value. Moreover,
     - if the element is larger than the current max (which is stored at the head of the doubly-linked list) then the doubly-linked list is emptied out, and the new element gets placed in the head as the new the max.
     - if the element is smaller than the current max, the element gets placed in the doubly-linked list in decreasing order. Discard any elements existing in the doubly-linked list that are smaller than this new element. This is because they will be removed from the singly-list queue before the newly-added element, and thus will never have the chance to be the max element of the queue.

   - **Removing an element:** When an element is removed, it is removed from the head of the singly-linked queue. Moreover, if the removed element was the maximum element, the head of the doubly-linked list must also be removed.

   Example:

Function Call	Queue Contents	DLList Contents	`max()` Returns
`add(3)`	[3]	[3]	3
`add(4)`	[3, 4]	[4]	4
`add(1)`	[3, 4, 1]	[4, 1]	4
`add(2)`	[3, 4, 1, 2]	[4, 2]	4
`remove()`	[4, 1, 2]	[4, 2]	4
`remove()`	[1, 2]	[2]	2
`add(8)`	[1, 2, 8]	[8]	8
`add(6)`	[1, 2, 8, 6]	[8, 6]	8
`add(5)`	[1, 2, 8, 6, 5]	[8, 6, 5]	8
`add(7)`	[1, 2, 8, 6, 5, 7]	[8, 7]	8

   Notice that by implementing it in this manner, the run-time of the `max()` is at worst $O(n)$ but on average, performs a little better than finding the max by having to visit every element of the queue to find the max.

Changes to the BookStore System

In **BooksStore.py**, make the following changes:

1. In the constructor, initialize the attribute `self.shoppingCart` as an empty `MaxQueue` object.
2. In the `loadCatalogue()` function, load the attribute `self.bookCatalog` as a `DLList` object.
3. Add a method called `getCartBestSeller()` that prints the title of the book that is the best-seller amongst the rest of the books in the cart. HINT: Observe that `shoppingCart.max()` returns the best-selling book in the shopping cart, because `Book` objects are compared based on their best-seller ranks. The expected print statement should be:

   ```
   print(f‘‘getCartBestSeller returned \n{best_seller} \nCompleted in {elapsed_time} seconds’’)
   ```

   where `best_seller` is the title of the book in the cart that is the best-seller, and `elapsed_time` is the number of seconds it took for the algorithm to return the title. You can use existing methods in Bookstore.py as an example for how to calculate `elapsed_time`.

Changes to `main.py`

1. Modify the function `main()` in the file **main.py** so that it offers the option to test a palindrome. When running **main.py**, the program should offer the options:

   ```
   1 Calculator
   2 Bookstore System
   3 Palindrome Test
   0 Exit/Quit
   ```

   When the user selects your newly-added option, namely, option 3, they should be prompted to enter a word/phrase. The program should react in the following manner:

   ```
   Enter a word/phrase: <user enters word/phrase here>
   Result: <Your program displays either Palindrome or Not a palindrome>
   ```

2. Modify the function `menu_bookstore_system()` so that it offers the options

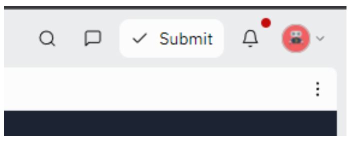
   ```
       s FIFO shopping cart
       r Random shopping cart
       1 Load book catalog
       2 Remove a book by index from catalog
       3 Add a book by index to shopping cart
       4 Remove from the shopping cart
       5 Search book by infix
       6 Get cart best-seller
       0 Return to main menu
   ```

   If the user selects option 6, the function should call `bookStore.getCartBestSeller()`.

SUBMISSION PROCESS

1. Submit your project for review on Repl.it.

You can find the “Submit" button on the top right-hand corner of your Repl.it workspace.



2. Modify Import Statements:

In `BookStore.py`, modify the following:

1. Comment out these unnecessary imports:
   - `import ChainedHashTable`
   - `import BinarySearchTree`
   - `import BinaryHeap`
   - `import AdjacencyList`
2. Import `MaxQueue`

Your `BookStore` module should begin like this after you have made the edits listed above:

```
import Book
import ArrayList
import ArrayQueue
import RandomQueue
import DLList
import SLLQueue
import MaxQueue
import time
#------------------------ COMMENT OUT THESE IMPORTS ------------------------#
# import ChainedHashTable
# import BinarySearchTree
# import BinaryHeap
# import AdjacencyList
#---------------------------------------------------------------------------#

...
```

3. Download and submit to CodePost.

- `SLLStack.py`
- `SLLQueue.py`
- `DLList.py`
- `MaxQueue.py`
- `BookStore.py`
- `main.py`

RUBRIC

	Full Credit 2 Pts	Partial Credit Pts. vary; See CodePost	No Credit 0 Pts
SLLStack implementation	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.
SLLQueue implementation	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.
DLList implementation	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.
Palindrome Method	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.
Reverse Method	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.
MaxQueue implementation	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.
`getBestSeller()` Implementation	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.
Palindrome Test Option in Main	Implementation is correct	Implementation is partially correct.	Implementation is incorrect, incomplete, or results in an unexpected `Error`.