MODULE 2 PROJECT: ARRAY-BASED 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 `ArrayStack`, `ArrayQueue` and `ArrayList` data structures covered Module 2 (Array-Based Lists). All the data structures should be fully functional and must follow the logic presented in the lecture.

   You have to modify the files,

   - **ArrayStack.py**
   - **ArrayQueue.py**
   - **ArrayList.py**

   Test your data structures by doing the following using the following tests.

   - Remove one element from an empty ArrayStack/ArrayQueue/ArrayList. This should result in an `IndexError`.
   - Stack: Add 5 elements and remove them checking that they are in opposite order of insertion, e.g., Inserting the sequence 5,4,3,2,1 should result in the sequence 1,2,3,4,5 when removing.
   - Queue: Add 5 elements and remove them checking that they are in the same order of insertion, e.g., Inserting the sequence 1,2,3,4,5 should 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)` should result in the array `[1, 2, 3, 4, 5]`. Remove 2 elements, e.g., index 2 and 3 and the final array should be `[1, 2, 4]`.

2. Implement `RandomQueue` so that the `remove()` function removes an element at random from amongst the elements currently in the queue. The `add(x)` and `remove()` operations in a RandomQueue should run in amortized constant time per operation.

   You have to modify the file **RandomQueue.py.**

   Hint: Use the random method `randint()` from the module `random` to return random numbers.

   Test your `RandomQueue` using the following tests.

   - Remove one element from an empty `RandomQueue`. This should result in an `IndexError`.
   - Add 5 elements then remove them all. Check that the `remove()` function returns random value.

Edits to the Calculator Application

A *mathematical expression* is a sequence of numbers, letters and grouping characters that are properly matched. For example, `a+(b*c+ d)/(a-c)` is a matched expression, but `a+(b*c+d/(a-c)` is not.

Implement the function `matched_expression()` in the module **Calculator.py** so that it returns `True` if a given string expression contains a matched mathematical expression, `False`, otherwise. Your function's algorithm should run in $O(n)$ time.

Test your function by running `main.py` and then selecting option 1 of the calculator menu. Try entering different matched and unmatched expressions such as `(3+x)(2(x-1)+7)`, `)+3(x+2)`, `((x-1)`, etc. The empty expression is considered to be a matched expression.

Hint: Use an `ArrayStack` object in the implementation of `matched_expression()`. Consider the invariant that for every closed parenthesis, there must be one open parenthesis.

Edits to the BookStore Application

1. In **BooksStore.py**, verify that:

   (a) the attribute `self.shoppingCart` is initialized as an empty `ArrayQueue` object in the constructor.

   (b) the `load_catalogue()` function loads the attribute `self.bookCatalog` as an `ArrayList` object.

2. In **BooksStore.py**, modify the function `searchBookByInfix(infix)` so that it takes an additional parameter `cnt`. This parameter should be given an integer value when the function is called, i.e., the function should become

   ```
   def searchBookByInfix(infix : str, cnt : int):
   ```

   Then, implement this function so prints the first `cnt` books/DVD's (or less if there are not at least `cnt`) in the loaded catalog that contain the substring given by `infix`. The loaded catalog should be the array list object `self.bookCatalog`.

   Hint: Use a for loop and the `in` operator to test whether the infix is found in a book title.

You can test your bookstore system by running main.py and selecting option 2 (bookstore system). Then, in the bookstore application submenu, test the following options:

Option 1: Load the `booktest.txt` catalog. This is a subset of 12 books from the larger database.

Option 3: Add a book by index to the shopping cart. You must enter the index i of the book in the catalog. The index corresponds to the position that the `Book` was stored in the array list. It is NOT the same thing as the book key. Since there are only 12 books, you should enter an index in the range 0 to 11, inclusive. This should be fully functional if your implementations of `ArrayList` and `ArrayQueue` are complete and correct. Observe that this operation emulates adding a book to a shopping cart in any online store. Repeat this process to add 5 more books.

Option r: Select this option to transfer your current shopping cart to a random shopping cart. You will be able to verify if this worked in the next step.

Option 4: Remove a book from the shopping cart. Repeat the process 5 more times. If `RandomQueue` has been implemented correctly, the books will be displayed in random order (i.e., not in the queue order that you added them).

Repeat option 3 and option 4 but do not transfer the books to a random shopping cart. This time the books should display in queue order, i.e. in the order you added them to the cart.

Option 5: Search a book by title: Give an infix of an existing title and the max number of matches that you want displayed. For example, if you perform a search with the infix “Ma" and count 10, the application should display 2 titles.

SUBMISSION PROCESS

---

1. Modify Some Files

Since you will not be uploading all files from the projects template to CodePost, you must modify some lines of code prior to submitting. Modify the following modules:

- `Calculator.py`

    1. Comment out these unnecessary imports:
        - `import BinaryTree`,
        - `import ChainedHashTable`
        - `import DLList`
        - `import operator`.

        Only the `numpy` and `ArrayStack` imports should be left un-commented.
    2. Assign the attribute `self.dict` to be `None`

    Your `Calculator` module should look like this after you have made the edits listed above:

```
# ------------------- COMMENT OUT THESE IMPORTS -----------------#
# import BinaryTree
# import ChainedHashTable
# import DLList
# import operator
# ---------------------------------------------------------------#
import numpy as np
import ArrayStack

class Calculator:
```

```
    def __init__(self) :
        self.dict = None #ChainedHashTable.ChainedHashTable(DLList.DLList)
```

- BookStore.py

  1. Comment out these unnecessary imports:
     - import DLList
     - import SLLQueue
     - import ChainedHashTable
     - import BinarySearchTree
     - import BinaryHeap
     - import AdjacencyList

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

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



class BookStore:
    '''
    BookStore: It simulates a book system such as Amazon. It allows  searching,
    removing and adding in a shopping cart.
    '''
    def __init__(self) :
        self.bookCatalog = None
        self.shoppingCart = ArrayQueue.ArrayQueue()

                    ...
```

2. Submit to CodePost

- `ArrayStack.py`
- `ArrayQueue.py`
- `ArrayList.py`
- `RandomQueue.py`
- `Calculator.py` - with modifications above
- `BookStore.py` - with modifications above
- `main.py`

RUBRIC

	Full Credit 2 pts.	Partial Credit pts. vary; See CodePost.	No Credit 0 pts.
ArrayStack implementation	Implementation is correct and passes all CodePost tests.	Implementation is partially correct. Fails one or more CodePost tests	Implementation is incorrect/incomplete and fails all CodePost tests.
ArrayQueue implementation	Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
ArrayList implementation	Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
RandomQueue implementation	Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
Validating mathematical expression	Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.
Searching books by infix	Implementation is correct and passes all CodePost tests.	Implementation is partially correct; fails one or more CodePost tests.	Implementation is incorrect/incomplete and fails all CodePost tests.