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.

Introduction

In this project, you will implement the array-based list data structures that we cover in Module 2. You will then use them in the following applications:

A Calculator System: A program that allows you to check whether a mathematical expression has balanced parentheses. Additional system functionalities will be implemented in future projects.

A Bookstore System: A program that allows you to load a product database into a catalog. It also allows you to search for items via catalog index or title infix. New items can be added to the catalog, and existing items can be deleted. The program supports adding items to a shopping cart and removing existing items in FIFO order.

Data Structure Implementations

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 must modify the files,

1. **ArrayStack.py** - contains the `ArrayStack` class which implements the `List` and `Stack` interfaces found in the module `Interfaces.py`
2. **ArrayQueue.py** - contains the `ArrayQueue` class which implements the `Queue` interface found in the module `Interfaces.py`
3. **ArrayList.py** - contains the `ArrayList` class which implements the `List` interface found in the module `Interfaces.py`

Edits to the Calculator Program

A *mathematical expression* is a sequence of numbers, letters and grouping characters. For this project, we will limit the grouping characters to open and closed parentheses. We say the parentheses are balanced if they are properly matched, i.e. if every open parenthesis has a corresponding closing parenthesis that ends the group. Examples:

Balanced parentheses:

- `a+(b*c+d)/(a-c)`
- `(a+(b*c+d)/(a-c))`
- `a+b`

Unbalanced parentheses:

- `a+(b*c+d/(a-c)`
- `)a*(b/d)(`
- `(a-d)/c)`
- `((a-d)/c`

Implement the function `balanced_parens(expr)` in the module **Calculator.py** so that it returns `True` if the parentheses in a given mathematical expression `expr` are balanced, `False`, otherwise. Your function's algorithm should run in $O(n)$ time.

Hint: Use an `ArrayStack` object in the implementation of `balanced_parens()`.

Edits to the Bookstore Program

In **BooksStore.py**, follow the directions in the `FIXME` comments to complete the implementation of the following methods:

1. `addToCatalog(i, book)` - inserts a new `Book` object, `book`, to the catalog at the given index `i`.
2. `removeFromCatalog(i)` - removes the book at index `i` from the catalog and displays its information `i`.
3. `getBookAtIndex(i)` - displays the information for the book stored at index `i` of the catalog.
4. `searchBookByInfix(infix, k)` - prints the first `k` books/DVD's in the loaded catalog whose titles contain the substring given by `infix`. If there are less than `k` titles, then it prints the max number of titles less than `k` that contain the substring.

   NOTE: The loaded catalog should be the array list object `self.bookCatalog`.
5. `addBookByIndex(i)` - adds the book at index `i` of the catalog to the shopping cart (which is an `ArrayQueue` object)
6. `removeFromShoppingCart()` - removes and displays a book from the shopping cart in FIFO order.

Testing Your Work

You should thoroughly test your work prior to submitting to CodePost. Here is how you can determine whether your implementation is working as required:

Testing Your Data Structures

The file `tester.py` contains a function called `test()` in which you can instantiate your data structure classes, store values into the objects, and then print the outcomes of calling the different methods of those objects. Some sample tests have been created for testing an `ArrayStack` object. You can build upon that example to include any other tests you think are necessary to check the correctness of your implementation.
To run your tests, begin by running `main.py`, then selecting option 3 in the main menu.

```
----------------------------------------
    MAIN MENU
    1. Calculator
    2. Bookstore System
    3. Run your own tester
    Q. Quit

Enter your selection: 3
```

Testing Your Application Systems

1. Download from Canvas the zip file called `module2_project_EXE.zip`. It contains the following files:
    - `books.txt` - a text database of 542,684 books
    - `booktest.txt` - a text database of 12 books
    - `Module 2 Array-Based Lists.exe` - an executable application of how the expected project should run once finished.
2. Extract the files, making sure that they are all saved to the same folder.
3. You can verify whether your implementation of the methods in `Calculator.py` and `BookStore.py` is correct by running `main.py` on Repl.it, and then exploring how your project behaves when you select the different menu options. Compare the outcomes of your selections against the responses that `Module 2 Array-Based Lists.exe` gives when you enter the same selections.

   Make sure to test every option on the main menu, and all options in the sub-menus of the `Calculator` and `Bookstore System`.

```
-------------------------------------
CALCULATOR MENU
1. Check mathematical expression
Q. Return to main menu
```

```
----------------------------------------
        BOOKSTORE MENU
        1. Load book catalog
        2. Edit the book catalog
        3. Access book at index
        4. Search book by infix
        5. Add a book by index to shopping cart
        6. Remove from the shopping cart
        Q. Return to main menu
```

Submission process

Submit the listed files to CodePost. You can submit an unlimited number of times without any penalty, as long as your submission is made before the due date. ***The score for the latest submission is kept.***

- `ArrayStack.py`
- `ArrayQueue.py`
- `ArrayList.py`
- `Calculator.py`
- `BookStore.py`
- `main.py`

RUBRIC

	Full Credit	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.