MODULE 4 PROJECT: HASH TABLES

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.

CLO 5. Design and analyze composite data structures.

Data Structure Implementation

Files to Modify: ChaineHashTable.py.

Directions: Implement the `ChainedHashTable` data structure with operations,

- `add(k, v)` - adds an item with key `k` and `v` if the key does not already exist in the table (i.e. no items with duplicate keys are allowed). Returns `True` if the item was added successfully, `False` otherwise.
- `find(k)` - returns the value of the item with key `k` if it exists in the table, `None` otherwise
- `remove(k)` - removes the item with key `k` and returns the value, if the key `k` exists in the table. Otherwise, returns `None`
- `size()` - returns the number of items in the table

Recall that each bin in the hash table stores a list of elements that share in common the same hash value. You can use the already-implemented `_hash(key)` method to determine which bin each element must be added to.

IMPORTANT: You will need to decide whether each bin will hold an `ArrayList` or a `DLList` to store the elements with the same hash code. You should consider run time efficiency to help you decide.

(Optional) Test your program:

- Remove one element from an empty `ChainedHashTable`. Should return `None`
- Search in an empty `ChainedHashTable`: `find(2)` should return `None`
- Add 3 elements: `add(1,"first")`, `add(2, "second")`, `add(3, "fourth")`
- Attempt to add an element with existing key 3: `add(3, "third")`. Print the outcome, it should be `False`
- Check that `size()` returns 3
- Remove one element: `remove(3)` and check that `size()` returns 2.
- Find one element: `find(3)` should return `None`
- Add 3 elements: `add(3, "third")`, `add(4, "fourth")`, `add(5, "fifth")`
- Check that `size()` returns 5
- Find one element: `find(3)` should return `"third"`

Changes to the BookStore System

Files to Modify: BookStore.py.

Directions:

1. Modify the constructor in `BookStore` so that it includes an additional attribute called `self.bookIndices`. Initialize it to an empty `ChainedHashTable` object. This hash table will be used to map `Book` keys to their corresponding index in `self.BookCatalogue` (i.e. it will allow us to keep track of the index where each `Book` object is located based on its key).
2. Recall that `Book` objects have attributes `key`, `title`, `group`, `rank`, and `similar`. Modify the `loadCatalog()` method so that when a `Book` gets loaded into the catalogue, a new element consisting of the `Book` key and the book's index in the catalogue is stored to the hash table. The pseudocode is as follows:

   ```
   For each row i in books.txt
        b = Book(key, title, group, rank, similar)
        bookCatalog.append(b)
        bookIndices.add(key, bookCatalog.size()-1)
   ```

   The attribute `self.bookCatalogue` should be a `DLList` as in the module 3 project.
3. Add a method to `BookStore` called `addBookByKey(self, key)` that adds the book with the given `key` to the shopping cart. *HINT*: First search the hash table `bookIndices` for the index corresponding to the given key. If the index is not `None`, add the `Book` at the given index in the `bookCatalogue` to the cart.

Changes to the Calculator System

Files to Modify: Calculator.py
Directions:
In the Module 2 project you created a method that validates if a mathematical expression is properly matched (i.e., if for every open parenthesis there exists a matching closing parenthesis).

In this assignment, you will use a `ChainedHashTable` to transform a valid expression that contains variables to an expression that contains the constant values defined for specific variables. For example, if the user specifies the values $a = 1.3,\ b = 4.2$, and $c = 2.7$ for the valid expression `((a+(b*c+d))/(a-c))`, then your calculator system will have an option to display the expression `((1.3+(4.2*2.7+d))/(1.3-2.7))`.

1. In the constructor of the `Calculator` class, initialize `self.dict` to a `ChainedHashTable` object. The hash table `self.dict` will contain the key-value pairs consisting of variables as keys and constant values for those variables as values (for example, a key `"a"` corresponding to a value `1.3`).
2. Add a new method to the `Calculator` class named `print_expression(self, exp)`. The method `print_expression(self, exp)` takes in string `exp` representing a valid mathematical expression (i.e. parentheses are matched), and returns a string where the variables in `exp` have been replaced by the values contained in `self.dict`. For example, if `calculator` is an instance of the `Calculator` class, and the user has set $a = 1.3,\ b = 4.2$, and $c = 2.7$, then

   ```
   calculator.print_expression("((a +(b * c + d))/(a - c))")
   ```

   must return the string `"((1.3 + (4.2 * 2.7 + d))/(1.3 - 2.7))"`.

Changes to Main

Files to Modify: main.py

Directions:

1. Modify `menu_bookstore_system()` so that it includes a new option to add a book to the shopping cart by key. Your new list of options should read as follows:

```
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
7 Add a book by key to shopping cart
0 Return to main menu
```

   If the user chooses option 7, the system should prompt for a key with the message:

```
Enter book key:
```

   If the key exists, the system will add the corresponding book to the cart and display the message,

```
Added title: <Insert book title>
```

   If the key does not exist, it should display the message `Book not found.`.

2. Modify `menu_calculator()` so that it includes two new options: one to introduce variable values, and another to print an expression with given variable values. Your new list of options should read as follows:

```
1 Check mathematical expression
2 Store variable values
3 Print expression with values
0 Return to main menu
```

- If the user chooses option 2, the system should...
  I. prompt the user to enter a variable and value.

```
Enter a variable: <user input>
Enter its value:  <user input>
```

  II. ask the user if they wish to enter another variable-value pair:

```
Enter another variable? Y/N
```

    (a) If the user selects `Y`, step I. is repeated.
    (b) Otherwise, the user is taken back to the calculator menu
- If the user chooses option 3, the system should...
  I. prompt the user for an expression using the message

```
Introduce the mathematical expression:
```

  II. check if the entered expression is valid (i.e. all parentheses are matched).
    (a) If the expression is valid continue to step III.
    (b) If the expression is invalid, display the message `Invalid expression`, and return to the top of calculator menu.
  III. display the expression with variables replaced by existing values. If no variables have been set, then the original expression without replacement is displayed.

SUBMISSION PROCESS

Submit to CodePost

- `ChainedHashTable.py`
- `DLList.py` or `ArrayList.py` depending on what you used to implement
- `ChainedHashTable`
- `BookStore.py`
- `Calculator.py`
- `main.py`

RUBRICS

	Full Credit 7 pts.	Partial Credit Pts. vary; See CodePost	No Credit 0 pts.
`ChainedHashTable` 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.

	Full Credit 2 pts.	Partial Credit Pts. vary; See CodePost	No Credit 0 pts.
`Calculator` 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.
`menu_calculator()`: 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.
`menu_bookstore_system()` 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.