Feb 12, 2024

Assignment – 1

[100 Points]

CSC-413-02
Spring 2024

San Francisco State University
Computer Science Department

Assignment Goal

The purpose of this assignment is to practice object oriented design to create an object that evaluates mathematical expressions.

1. For this part of the assignment, you will build a customized iterator for a linked implementation of an AbstractDataType(ADT) List. You will need to refresh your skills from the Data Structures class where you implemented ADT List [50 pts]

    A. Create a class *MyLList* that represents ADT List. You may use the code from your DataStructures class where you implemented linked version of the ADT List
    B. Create a class *LinkedListWithIterator* with the requisite hierarchy, along the lines of *ArrayListWithIterator* as discussed in the class. This iterator will iterate through contents of your *MyLList* object rather than the array-based list that was discussed in class.
    C. You will need to also create the *ListInterface* class that outlines all methods. This class should include all methods as defined in the ListInterface, as shown below. Your *MyLList* class created above will need to implement all the methods of the *ListInterface* as shown below

```
/** Adds a new entry to the end of this list.
    The list size is increased by 1.
    @param newEntry  The object to be added as a new entry. */
 public void add(E newEntry);

 public boolean add(int newPosition, E newEntry);


 public Comparable remove(int givenPosition);

 /** Removes all entries from this list. */
 public void clear();


 public E replace(int givenPosition, E newEntry);

 /** Retrieves the entry at a given position in this list.
    @param givenPosition  An integer that indicates the position of
                          the desired entry.
    @return  A reference to the indicated entry.
    @throws  IndexOutOfBoundsException if either
             givenPosition < 1 or givenPosition > getLength(). */
 public Comparable getEntry(int givenPosition);

 /** Retrieves all entries that are in this list in the order in which
    they occur in the list.
    @return  A newly allocated array of all the entries in the list.
             If the list is empty, the returned array is empty. */
 public Comparable[] toArray();

/** Sees whether this list contains a given entry.
    @param anEntry  The object that is the desired entry.
```

```
   @return  True if the list contains anEntry, or false if not. */
public boolean contains(E anEntry);

/** Gets the length of this list.
   @return  The integer number of entries currently in the list. */
public int getLength();

/** Sees whether this list is empty.
   @return  True if the list is empty, or false if not. */
public boolean isEmpty();
```

D. In addition to the *ListInterface* methods implemented in your *MyLList* class, you will also need to implement the *getIterator()* method, as discussed during code walkthrough of code in class, that returns an initialized instance of

E. Once all the above classes are implemented:
   - Create an instance of *MyLList* class, populating it with requisite data. To keep it simple, you may just add string objects to it.
   - Get the iterator object for this instance
   - Use the Iterator methods like *hasNext()*, *next()* etc., to iterate and print objects in your list
   - Your test should print all data in the list object in the console

2. Build an application called ExpressionEvaluator. The primary responsibility of this application will be to evaluate a predefined Infix and Postfix algebraic expressions with different identifier values [50 pts]

   A. ExpressionDriver class
      This will be the client adaptor class that will initiate the test for evaluation of both infix and postfix expressions

   B. InfixEvaluator class
      - This class will be responsible to run the logic to evaluate an infix expression
      - This class will have a method as follows:
        public static evaluateInfix(String str), which will take a string as an argument and run the logic using 2 stacks named *operatorStack* and *valueStack*, as discussed in class, to compute the infix value

   C. PostFixEvaluator class
      - This class will be responsible to run the logic to evaluate a postfix expression
      - This class will have a method as follows:
        public static evaluatePostfix(String str), which will take a postfix string as an argument and run the logic as discussed in class, to compute the postfix value

   D. The Expressions that will be processed using the methods defined above will be as below
      - Infix expression: (a+b)*(c+d)

- Postfix Expression: ac-b^d+

These above two string expressions should be defined as private static final.

E. Reading in values for each identifier in the expressions above

(a) While the infix and postfix expressions are as shown above, the values for the identifiers will be provided at tun time by the user from the console

(i) The ExpressionDriver adaptor class stated 2(A) above should have a *while* loop that will:

- Ask user to enter values, either integer or double, for the 4 identifiers a, b, c and d, which should be stored in the application
- Call the *evaluateInfix* and *evaluatePostfix* methods which in turn will use the user provided values for each of the 4 identifiers(a, b, c, d), assign the values to each identifier, compute the final value for that expression and print the computed value in the following format for the two expression types:

  "Value of infix string (a+b)*(c+d) with a = 10, b = 3, c = 3, d = 4 is 52"
  "Value of postfix string ac−b^d+ with a = 10, b = 3, c = 3, d = 4 is 68"

- The while loop, as stated above, in each of its iteration should ask the user if computing is needed. If the user responds "yes", identifier values will be solicited from user and both expressions are computer, If, however, the user chooses to answer "no", the program should be ended.