CS2400 Project#1 (100 Points)
Fall - 2019
Instructor: Tannaz R.Damavandi
Due Date: Friday - 10/04/2019 at 11:59 pm.

**Purpose:**

1. Understand the interface and application of ADT Bags.
2. Understand the interface and application of ADT Stack.

**Task #1 (50 Pts.)**

Class rosters include primary information of students. In this task you will learn how to use ADT Bag to implement a class roster with unspecified size.

- **Step 1:** Use a **resizable array** to implement ADT Bag and write a java program to let the user add, drop and search for a specific student using student ID **(Duplication is not allowed)**. Your program should also provide the user with the following options:
    - Class size
    - If class is full or empty
    - The number of students in the same academic level.
  - In your program, define a java class called *Student* with the following data fields: *Student ID, first_name, last_name and academic level* with: *int , String, String* and *String* data types respectively. Note that **academic level can be: freshman, sophomore, junior** or **senior only.**
- **Step 2:** Repeat step 1, but use a **chain of linked nodes** instead of a resizable array.

**Task #2 (50 Pts.)**

We are used to write arithmetic expressions with the operator between the two operands: a + b or c % d. If we write a + b * c, however, we have to apply precedence orders to avoid ambiguous evaluation. This type of expression is called **infix** expression. There are other two types of different but equivalent ways of writing expressions.

- **Infix**: X + Y: Operators are written in-between their operands. Infix expression needs extra information to make the order of evaluation of the operators clear: precedence and associativity, brackets ( ). For example, A * (B + C) / D.
- **Postfix**: X Y +: Operators are written after their operands. The above infix expression should be written as A B C + * D /
- **Prefix**: + X Y: Operators are written before their operands. The above infix expression should be written as / * A + B C D

Implement the ADT stack using a **resizable array**, a **linked chain**, and a **vector** to implement two methods that can convert an *infix* expression entered by the user to its equivalent *postfix* and *prefix* expressions. Your program should ask the user to enter the *infix* expression. Once the user hits ENTER; your program should:

- **Step1:** Check whether the infix expression is balanced or not (use the algorithm *checkBalance* at the end of this file).
- **Step 2:** If step 1 is successful, return both *prefix* and *postfix* expressions of the *infix* expression; otherwise throw an error and ask the user for a balanced infix expression.

**What to Submit?**

1. Java source codes for each task combined with executable.jar files. **(Please comment your code properly)**
2. A detailed report and explanation together with your program's output.
3. Readme.txt (Please describe how to run your code)
4. Please zip all documents as yourname_project1.zip and submit it on blackboard.

***Discussion among students is encouraged, but I expect each student to hand in original work.***

**The algorithm *checkBalance***

```
Algorithm checkBalance(expression)
// Returns true if the parentheses, brackets, and braces in an expression are paired correctly.

isBalanced = true
while ((isBalanced == true) and not at end of expression)
{
   nextCharacter = next character in expression
   switch (nextCharacter)
   {
      case '(': case '[': case '{':
         Push nextCharacter onto stack
         break

      case ')': case ']': case '}':
         if (stack is empty)
            isBalanced = false
         else
         {
            openDelimiter = top entry of stack
            Pop stack
            isBalanced = true or false according to whether openDelimiter and
                         nextCharacter are a pair of delimiters
         }
         break
   }
}

if (stack is not empty)
   isBalanced = false
return isBalanced
```