Lab 1 Analysis-Haoyu Qi

Description

In this project, I create my own program to realize the ADT stack in creatstack.java.

Stack

Stack is a kind of first in last out (FILO) ADT. The first element will be pushed at the bottom of the stack and will be the last one be pop off the stack. In this project, I realize the following functions of stack: push(make the element into the stack and it will be placed at the top of the stack), pop(take out the top element of the stack and do not place it back), peek(show the top element and the former stack structure is not changed), isEmpty(check whether the stack is empty).

Justification of Utilizing Stack

In this scenario, we choose stack to converse the prefix and postfix is mainly because of its FILO property. Between the prefix and postfix, the operate sequence is conversed but the operons do not. Using stack to store the operons then rebuild the output string is a good way to converse.

Appropriateness of Application

In my application, I firstly realize the conversion of prefix to postfix, this is the basic work. Plus, I also realize the input format mistake checking to make sure only the correct input prefix expression will be conversed to postfix expression. Also, I apply some modular design in the application, separate functions in different class file, this makes the advance and modification easier.

Description and justification of your design decisions

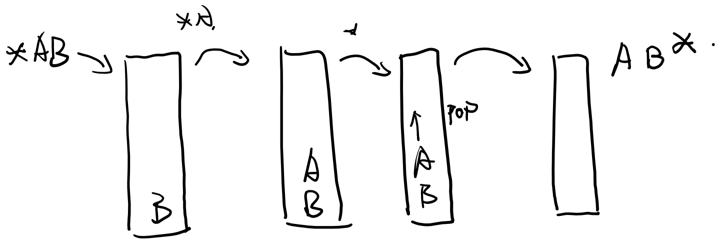
1. I used array to realize the stack design and I have realized the functions of stacks as I described above. The stack, top, bottom are artificially defined in the coding.
2. The I/O is realized by BufferedRead and BufferedWrite as we shown in the Project0.
3. I wrote a distinct class to distinguish the character type because it is needed in both conversion and reading.
4. Also, I designed a methods to check the input mistake, the mistake includes containing invalid elements like"@, # ,&" and the operons in this lab contains characters and integers.

The application can check 2 kinds of mistake input format, the incorrect numbers of operons and operaters, and the bad prefix format unable to convert to postfix.

5. I apply some user friendly designed in the application, like file input guild, error warning text or a as compact as possible output format.

Efficiency with time and space

Because the prefix to postfix conversion is liner and no circulation, as the figure shown below, it only depends on the input character sizes, so the time complexity is O(n) for conversion. The stack usage in space is also liner, so the space complexity is also O(n).



What I have learned

This lab project gives me a very concrete impression of the stack and its usage. I may try to imply stack for some specific problems similar to the prefix to postfix in the future. The codes I wrote to realize the stacks may be very useful in my future work. Also, because my java is not skilled, the use of BufferedRead is also a progress for me.

What you might do differently next time

The input format is still seems too strick for the users, maybe I will try to imply the function of process the keyboard input in DOS window or PDF/xls input. Also the users may care about the infix of the expression, I may add the infix output in the future. Plus, the application can only realize the prefix to postfix convers, maybe I will realize the prefix-infix-postfix mutual conversion and automatically recognize the input format.

Recursive Solution

```
Public recursiveS()
String input, chara, str1, str2
For i=0, i<input.length(), i++
Chara= input.charAt(i)
if reader == -1
    return output;
else
if Chara is ValidOperons
```

```
       str1=recursiveS (input.substring(1))
       str2=recursiveS (input.substring(1))
       output=str1+str2+ Chara;
       return output
  else
                     return recursiveS(chara)
}
```

Comparison

The recursion method has shorter coding. But I think the stack has less time complexity and more easy to understand.