Homework 3 - CS146

Frank Mock

March 4, 2014

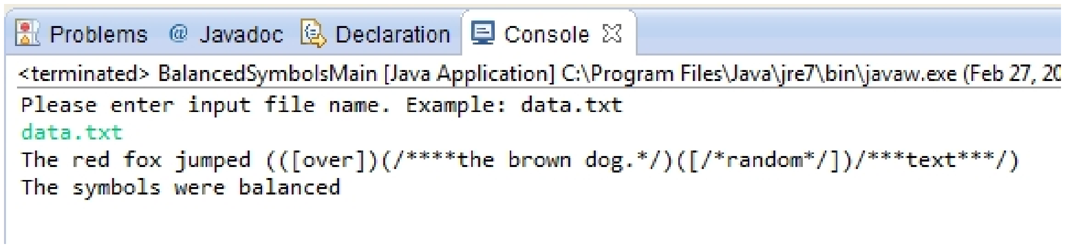
3.7

O(n^2) because the trimToSize() method is causing extra work. It is reducing the capacity of the Arraylist after an integer is added to the array which doesn't leave any room for the next integer insertion so memory must be re-allocated.

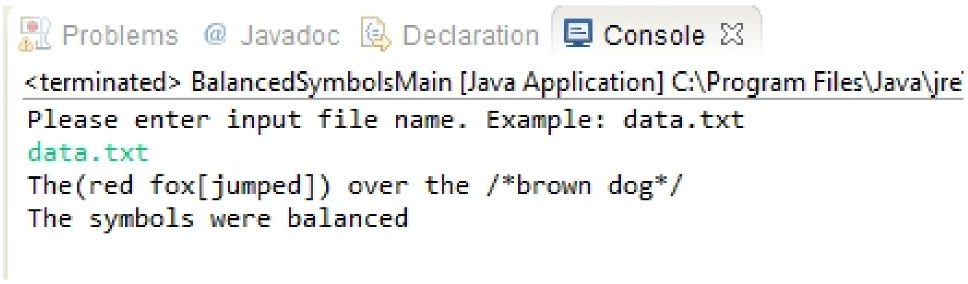
3.21

```
import java.util.Scanner;
import java.util.Stack;
import java.io.File;
import java.io.FileNotFoundException;

/**
 * Frank Mock - CS146 - spring 2014
 * Reads text from a file and checks if the symbols are balanced.
 * This program does not tell you where or what caused an un-
 * balanced condition, only that the symbols in the text are
 * unbalanced. It does this by pushing symbols onto a stack and
 * removing them as appropriate. If the stack is not empty after
 * all the text has been read then there is an unbalanced symbol
 * condition.
 */

public class BalancedSymbolsMain
{
    public static void main(String[] args) throws FileNotFoundException
    {
        Stack<Character> symbolStack = new Stack<Character>();
        Stack<Character> commentStack = new Stack<Character>();

        Scanner in = new Scanner(System.in);
        //Get the file name from the user
        System.out.println("Please enter input file name. Example: data.txt ");
        String fileName = in.next();

        //Read the file
        File file = new File(fileName);
        Scanner input = new Scanner(file);

        while(input.hasNextLine())
        {
            String text = input.nextLine();
            for(int i = 0; i < text.length(); i++)
            {
                if(text.charAt(i) == '('
                || text.charAt(i) == '{'
                || text.charAt(i) == '[')
                {
                    symbolStack.push(text.charAt(i));
                }
                else if(text.charAt(i) == ')'
                     || text.charAt(i) == '}'
                     || text.charAt(i) == ']')
                {
                    if(symbolStack.empty())
                        symbolStack.push(text.charAt(i));
                    else
                        symbolStack.pop();
                }
                else if(text.charAt(i) == '/')
                {
                    if(text.indexOf("/*") == i)
                    {
                        commentStack.push('/');
                        commentStack.push('*');
                    }
                }
                else if(text.charAt(i) == '*')
                {
                    if(text.indexOf("*/") == i)
                    {
                        if(commentStack.empty())
                        {
                         commentStack.push('*');
                         commentStack.push('/');
                        }
                        else
                        {
                            commentStack.pop();
                            commentStack.pop();
                        }
                    }
                }
            }
            System.out.println(text);
        }//end while loop

        if(symbolStack.isEmpty() && commentStack.isEmpty())
            System.out.println("The symbols were balanced");
        else
            System.out.println("The symbols did not balance!");
        in.close();
        input.close();
    }
}
```

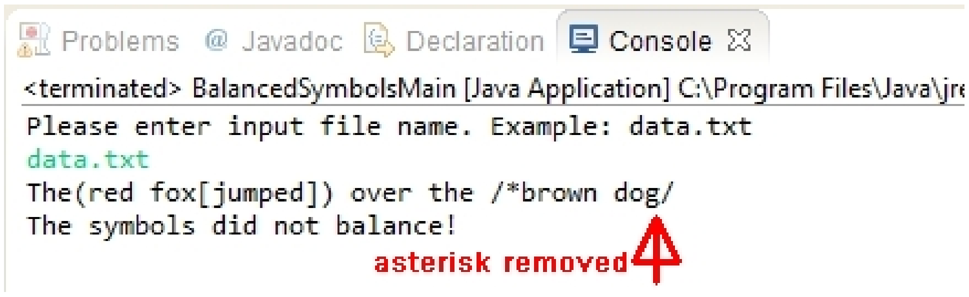
Here are some screen shots showing the results of both balanced and unbalanced text input.



```
<terminated> BalancedSymbolsMain [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Feb 27, 20
Please enter input file name. Example: data.txt
data.txt
The red fox jumped (([over])(/****the brown dog.*/)([/*random*/])/***text***/)
The symbols were balanced
```

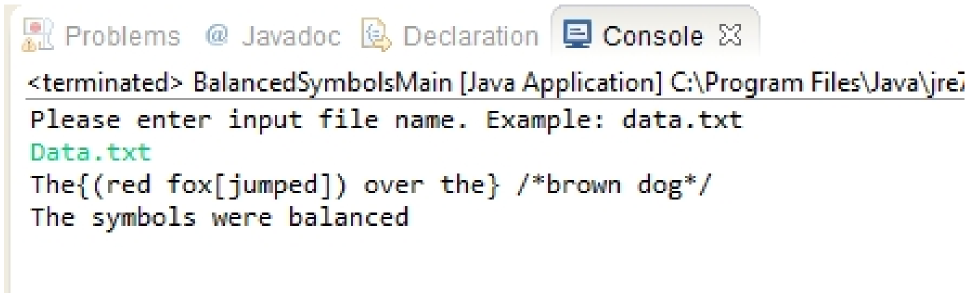


```
<terminated> BalancedSymbolsMain [Java Application] C:\Program Files\Java\jre
Please enter input file name. Example: data.txt
data.txt
The(red fox[jumped]) over the /*brown dog*/
The symbols were balanced
```

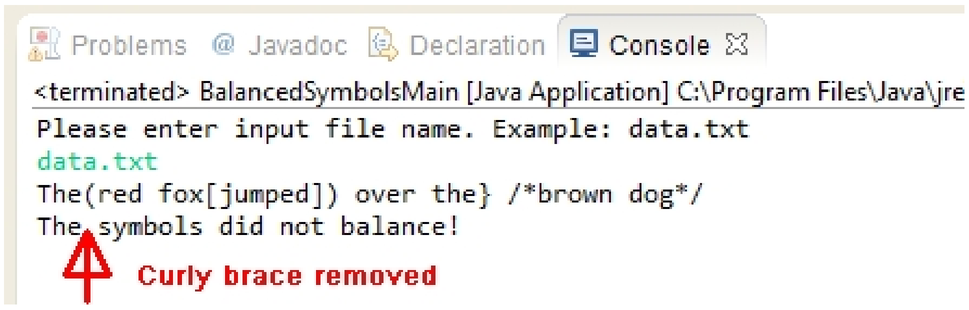


```
<terminated> BalancedSymbolsMain [Java Application] C:\Program Files\Java\jre
Please enter input file name. Example: data.txt
data.txt
The(red fox[jumped]) over the /*brown dog/
The symbols did not balance!
```

asterisk removed

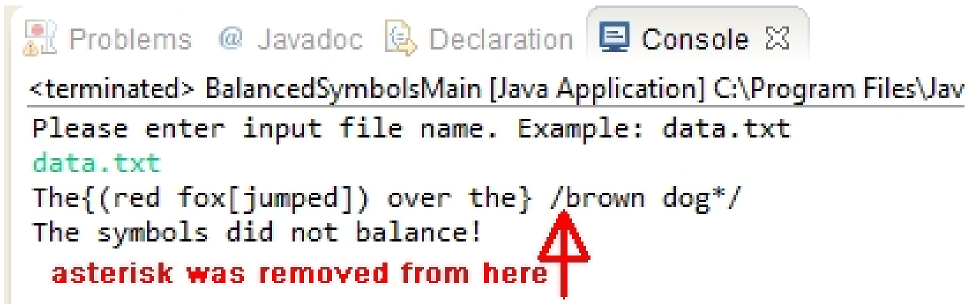


```
<terminated> BalancedSymbolsMain [Java Application] C:\Program Files\Java\jre7
Please enter input file name. Example: data.txt
Data.txt
The{(red fox[jumped]) over the} /*brown dog*/
The symbols were balanced
```

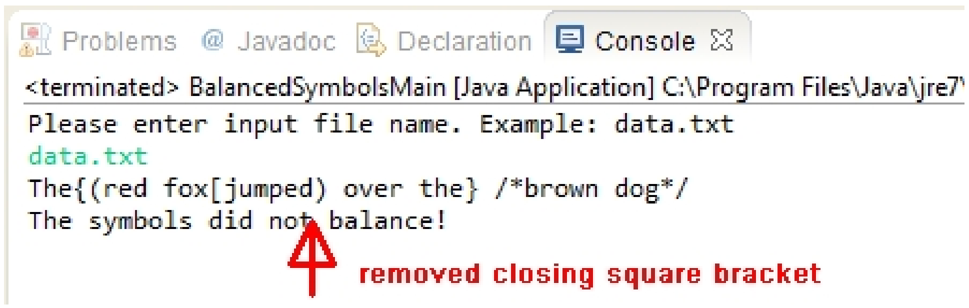


```
<terminated> BalancedSymbolsMain [Java Application] C:\Program Files\Java\jre
Please enter input file name. Example: data.txt
data.txt
The(red fox[jumped]) over the} /*brown dog*/
The symbols did not balance!
```

Curly brace removed



```
<terminated> BalancedSymbolsMain [Java Application] C:\Program Files\Jav
Please enter input file name. Example: data.txt
data.txt
The{(red fox[jumped]) over the} /brown dog*/
The symbols did not balance!
```



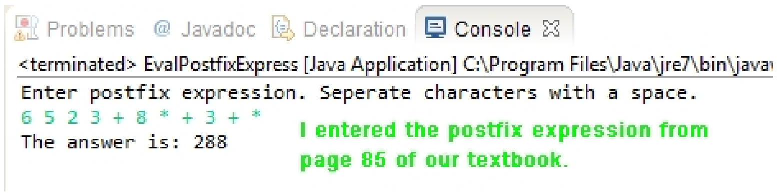
```
<terminated> BalancedSymbolsMain [Java Application] C:\Program Files\Java\jre7
Please enter input file name. Example: data.txt
data.txt
The{(red fox[jumped) over the} /*brown dog*/
The symbols did not balance!
```

3.22

```
import java.util.ArrayList;
import java.util.Scanner;
import java.util.Stack;

/*
 * Frank Mock - CS146
 * This program evaluates a postfix expression. When the user
 * enters the expression at the commandline, operands and
 * operators must be separated by a space. Also, this program
 * assumes a correctly entered postfix expression. Use only
 * +, -, *, / for addition, subtraction, multiplication and
 * division respectively.
 */
public class EvalPostfixExpress
{
public static void main(String[] args)
{
      //the working stack
      Stack<Integer> theStack = new Stack<Integer>();
      //list of operators that will be used
      ArrayList<String> operators = new ArrayList<String>();
      operators.add("+");
      operators.add("-");
      operators.add("*");
      operators.add("/");

      //Get a postfix expression from the user
      System.out.println("Enter postfix expression. Seperate characters with a space.");
      Scanner in = new Scanner(System.in);
      String expression = in.nextLine();
      expression.trim();

      //separate expression at spaces and put pieces into array
      String[] characters = expression.split("\\s+");

      //evaluate expression using appropriate stack operations push or pop
      for(String s : characters)
      {
       if(!operators.contains(s))//its a number
        {
         int num = Integer.parseInt(s);
         theStack.push(num);
        }
        else //its an operator
        {
         int index = operators.indexOf(s);
         int b = theStack.pop();
         int a = theStack.pop();

         //determine which type of operator
         switch(index)
         {
            case 0:
            theStack.push(a + b);
            break;
```

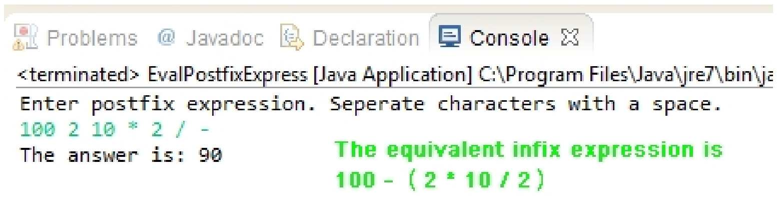
```
                case 1:
                theStack.push(a - b);
                break;
                case 2:
                theStack.push(a * b);
                break;
                case 3:
                theStack.push(a / b);
                break;
             }
            }
        }//end loop
        System.out.println("The answer is: " + theStack.pop());
        in.close();
    }
}
```

The following are some screen captures showing the program works correctly.



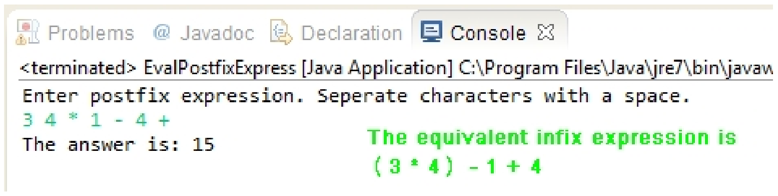
```
<terminated> EvalPostfixExpress [Java Application] C:\Program Files\Java\jre7\bin\java
Enter postfix expression. Seperate characters with a space.
6 5 2 3 + 8 * + 3 + *
The answer is: 288
```

I entered the postfix expression from page 85 of our textbook.



```
<terminated> EvalPostfixExpress [Java Application] C:\Program Files\Java\jre7\bin\ja
Enter postfix expression. Seperate characters with a space.
100 2 10 * 2 / -
The answer is: 90
```

The equivalent infix expression is 100 - (2 * 10 / 2)



```
<terminated> EvalPostfixExpress [Java Application] C:\Program Files\Java\jre7\bin\javaw
Enter postfix expression. Seperate characters with a space.
3 4 * 1 - 4 +
The answer is: 15
```

The equivalent infix expression is (3 * 4) - 1 + 4

3.28

The following methods that perform an insertion use a method called reallocate that doubles the capacity of the circular array when it becomes full. Since the allocation is amoritized across each insertion, these methods are still considered a constant time operation O(1). I coded a class called QueueArray to ensure that the methods that follow are functional.

- front : an integer that stores the index of the front pointer
- back : an integer that stores the index of the back pointer
- size : an integer that keeps track of the number of elements
- capacity : an integer that keeps track of the array max capacity
- theData : a generic array to store data

```
/*
 * Adds an item to the front of the dequeue
 */
public boolean push(Type item)
{
      if(size == capacity)
      reallocate();

      front--;

      if(front < 0)
      {
         front = capacity - 1;
         if(front == back)
            reallocate();

         theData[capacity - 1] = item;
      }
      else if(front == back)
      {
         reallocate();
         theData[capacity - 1] = item;
      }
      else
      {
         theData[front] = item;
      }

      size++;
      return true;
  }
```

```
/*
* generic method that removes item at the front
*/
public Type pop()
{
   if(size == 0)
      return null;
   else
   {
      Type frontEntry = theData[front];
      front = (front + 1) % capacity;
      size--;
      return frontEntry;
   }
}
```

```
/*
*generic method that inserts item at the rear end
*/
public boolean inject(Type item)
{
  if(size == capacity)
     reallocate();

  size++;
  back = (back + 1) % capacity;
  theData[back] = item;
  return true;
}
```

```
/*
* Remove the item from the rear of the queue
*/
public Type eject()
{
  if(size == 0)
    return null;
  else
  {
     size--;
     back--;
     return theData[back];
   }
 }
```

3.34a.

```
1 Place a reference to the current node in a queue
2 While the next node doesn't equal null move to the next node
3 Compare the current node to the first item in the queue
4 If they match a cycle exists
5 GOTO 1
```

3.34b.

```
• Place two iterators at the first node (call them begin and current)
• While current doesn't equal NULL
          Advance to the next node
          If current equals begin a cycle exists
```