Bryan Arnold

10/16/16

CSE 2100

Programming Assignment 4

Description

This program is designed to take the values of numbers 1-13, make as many 4 number combinations with the numbers with no repeating numbers, then take 3 operators (+-*/) and find all combinations of them with repeats, then combine the two using postfix notation to determine how many combinations are equal to 24 and display them in a step by step format. How my program goes about this is first, it finds all the permutations of the numbers and the operators. After those are found, they are then converted into the same type to easily combine them into all possible postfix combinations. Finally, once those possible combinations are found, utilizing a stack implementation, my program evaluates each postfix notation in proper mathematical order and displays a step by step computation of each postfix, as well as the total number of postfixes that equal 24.

Tradeoffs

First, I considered how to go about getting the permutations of postfix combinations. This would require all the combinations or operators and numbers, that are within their parameters of repeats or no repeats. I could find all possible for the numbers, and eliminate repeats with another method, but this adds runtime and code so I opted to do that with one simple if statement within the card permutations method. The operator permutations were with repeats, so I didn't have to consider another method for repeats for the operators, making that method simple. Next, I decided to take these permutations and convert them into strings. This allowed for easy combination of the two permutations into valid postfix notations, and overall easier postfix evaluation later on. Lastly, by finding all the valid permutations of postfix before and evaluation was done, I could cross check with math and testing to ensure all the permutations were valid and without repeats and followed postfix rules. This ensured no mess ups in the postfix permutations later on during evaluations and display.

Extensions

A could have done a few things better. First, I could've made the operators' storage strings to begin with instead of characters. This would eliminate an entire method and in turn create less code and runtime, making my program cleaner and more efficient. I could easily fix this by making the storage of operators an array of string instead of characters. Next, I could've done the above to my storage of number values. This shares the same reasoning as the operators' switching to a string array as well, and how to improve it. Lastly, there were a few spots that were slightly hardcoded. This was due to time crunch and that I couldn't quite figure out a way to do it without slight hardcoding. To fix these spots, I could have spent more time really analyzing what needed to be done as well as find an actual non hardcoding method.

Test Cases

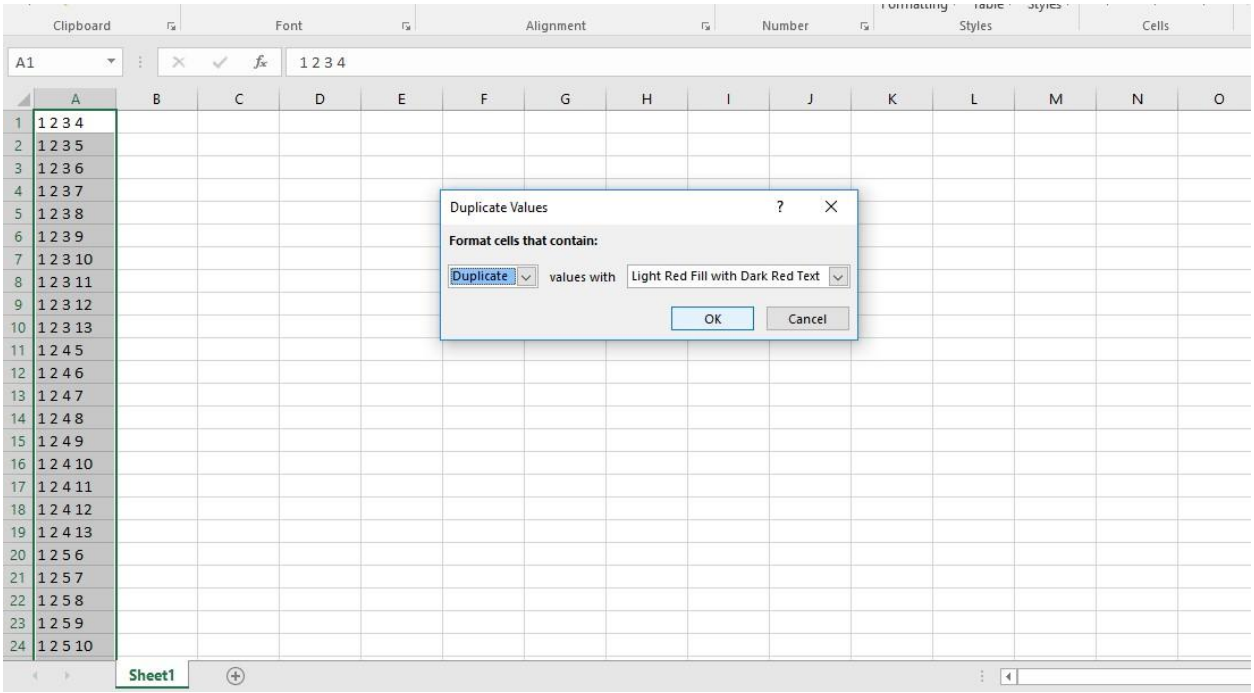
First, the values of cards and operators were my test case values for each method of this assignment. I chose these values because not only is it the end goal, it allows me to make extra sure that my program works correctly. So the card values were 1-13, and the operators were +-*/. This also helped prove the correct implementation of my stack class later on in the evaluations.

To begin, I tested that each of the permutations of the card values and operators were without repeats of the same permutation. This was the test output for the card value permutations:

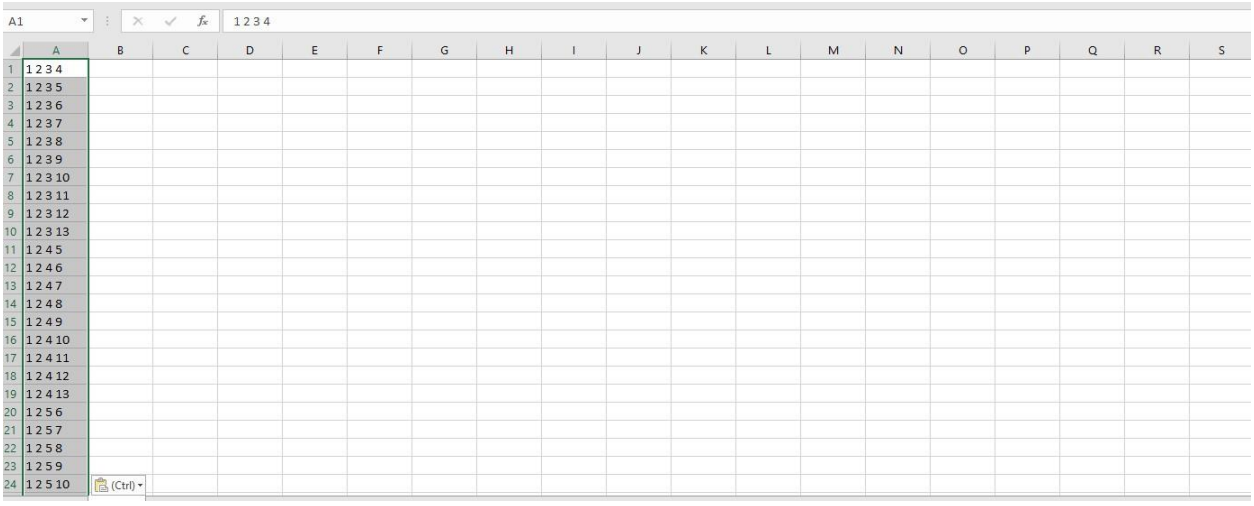
```
Valid permutations: 658
6 7 11 13
Valid permutations: 659
6 7 12 13
Valid permutations: 660
6 8 9 10
Valid permutations: 661
6 8 9 11
Valid permutations: 662
6 8 9 12
Valid permutations: 663
6 8 9 13
Valid permutations: 664
6 8 10 11
Valid permutations: 665
6 8 10 12
Valid permutations: 666
6 8 10 13
Valid permutations: 667
6 8 11 12
Valid permutations: 668
6 8 11 13
Valid permutations: 669
6 8 12 13
Valid permutations: 670
6 9 10 11
Valid permutations: 671
6 9 10 12
Valid permutations: 672
6 9 10 13
Valid permutations: 673
6 9 11 12
Valid permutations: 674
6 9 11 13
Valid permutations: 675
6 9 12 13
Valid permutations: 676
6 10 11 12
Valid permutations: 677
6 10 11 13
Valid permutations: 678
6 10 12 13
Valid permutations: 679
6 11 12 13
Valid permutations: 680
7 8 9 10
Valid permutations: 681
7 8 9 11
Valid permutations: 682
```

Note: total permutations displayed at end, which is 715.

As you can see, it displays each permutation of 4 card values, then the total number of valid permutations that have no repeats. No repeat card values in the same permutation is already accounted for. To ensure further uniqueness of each permutation, I put the values in excel and used conditional formatting and duplicate highlight tool. This checks for duplicates of highlighted cells and indicates if there are repeats, as seen below:



After pressing ok, duplicates should turn red and a notification at the top of the screen should appear to notify me of a repeat. This is after ok was pressed:



As you can see, there was no notification or color change of the cells, meaning each combo is unique and without repeats. This same method was used for operator's permutations and postfix permutations, and both had no repeats indicated by excel. These are some example screenshots from test runs of both the operator permutations and postfix permutations:

Operator Permutations (can have repeat values in each permutation):

```
Permations: 28
-/+
Permations: 29
-/-
Permations: 30
-/*
Permations: 31
-//
Permations: 32
*++
Permations: 33
*+-
Permations: 34
*+*
Permations: 35
*+/
Permations: 36
*-+
Permations: 37
*--
Permations: 38
*-*
Permations: 39
*-/
Permations: 40
**+
Permations: 41
**-
Permations: 42
***
Permations: 43
**/
Permations: 44
*/+
Permations: 45
*/-
Permations: 46
*/*
Permations: 47
*//
Permations: 48
/++
Permations: 49
/+-
Permations: 50
/+*
Permations: 51
/+/
Permations: 52
```

Note: real total displayed at end of test code, which is 64.

PostFix Permutations:

```
10 11 12 13 / - *
10 11 12 / 13 - *
10 11 12 / - 13 *
10 11 / 12 13 - /
10 11 / 12 - 13 /
10 11 12 13 / - /
10 11 12 / 13 - /
10 11 12 / - 13 /
10 11 / 12 13 * +
10 11 / 12 * 13 +
10 11 12 13 / * +
10 11 12 / 13 * +
10 11 12 / * 13 +
10 11 / 12 13 * -
10 11 / 12 * 13 -
10 11 12 13 / * -
10 11 12 / 13 * -
10 11 12 / * 13 -
10 11 / 12 13 * *
10 11 / 12 * 13 *
10 11 12 13 / * *
10 11 12 / 13 * *
10 11 12 / * 13 *
10 11 / 12 13 * /
10 11 / 12 * 13 /
10 11 12 13 / * /
10 11 12 / 13 * /
10 11 12 / * 13 /
10 11 / 12 13 / +
10 11 / 12 / 13 +
10 11 12 13 / / +
10 11 12 / 13 / +
10 11 12 / / 13 +
10 11 / 12 13 / -
10 11 / 12 / 13 -
10 11 12 13 / / -
10 11 12 / 13 / -
10 11 12 / / 13 -
10 11 / 12 13 / *
10 11 / 12 / 13 *
10 11 12 13 / / *
10 11 12 / 13 / *
10 11 12 / / 13 *
10 11 / 12 13 / /
10 11 / 12 / 13 /
10 11 12 13 / / /
10 11 12 / 13 / /
10 11 12 / / 13 /
Total permutations: 228800
```

Note: 228800 is total possible postfix permutations.

Finally, in regards to permutations, I used math to cross check the total amount of permutations that I got. First, for cards, these following rules must be applied. There are 13 total possible numbers, there can only be 4 cards per permutation, there cannot be repeat number values in each permutation, and each permutation must be unique. Taking these rules and using some old math equations for permutation calculations, you get the following equation:

$$Total\ Permutations = \frac{n!}{r!\,(n-r)!}$$

In this equation n = 13, total possible numbers, and r = 4, total cards per permutation. Finally, we get the total permutations:

$$Total\ Permutations = \frac{13!}{4!\,(13-4)!} = 715\ permutations$$

Now, following the test code, the final count of permutations was 715, meaning I found all the valid permutations and they were all correct.

Next, for operators, these following rules must be applied. There are 4 possible operators, and only 3 possible operators per permutation. This equation is much simpler to calculate and find and by apply the above rules we get the equation:

$$Total\ Permutations = n^r$$

In this equation n = 4, total possible operators, and r = 3, total operators per permutation. Finally, we get the total permutations:

$$Total\ Permutations = 4^3 = 64\ permutations$$

Now, following the test code results, the final count of permutations was 64, indicating that I found all the possible permutations that are all valid and unique.

Lastly, the amount of postfix permutations is simple to find. We simply have to multiply the amount of card permutations by the operator permutations, since each card permutation has each operator permutation, then multiply that result by the number of possible postfix notations, which is 5. The total number of postfix notations was found in discussion during class (see announcements for the course). This is the equation and calculations for the total postfix permutations:

$$Total\ Permutations = card\ permutations * operator\ permutations * postfix\ possibilities$$

In this equation, card permutations = 715, operator permutations = 64, and possible postfixes = 5. Now, for the calculations:

$$Total\ Permutations = 715 * 64 * 5 = 228800\ permutations$$

Finally, as seen in the test code run, the final amount of postfix notations was 228800, meaning I got the number of postfixes right and in uniqueness and correctness, as it doesn't allow for repeats. On to the evaluation methods.

For the final two methods of my class, I simply took the card values 1-13 and operators +-*/ and put them into all possible postfix notations. Then, using my evaluation method, with the T comments commented out (explained in the comments, basically test code sysout calls that aren't called in the final program and are commented out. These were commented out, and can be uncommented as mentioned in code, to allows for 24 postfix solutions to be printed, as the total permutations calculations won't all fit on the console window) I checked all, not quite but a ton, of the results of displayed computation to make sure my methods worked correctly, which also in turn ensured that my stack class was implemented correctly since this method wouldn't work if they were implemented incorrectly. Below is the output for all postfix permutations calculations:

```
Postfix Notation: 3 6 7 12 - + *
Step by step computation
7.0 - 12.0 = -5.0
6.0 + -5.0 = 1.0
3.0 * 1.0 = 3.0

Postfix Notation: 3 6 7 - 12 + *
Step by step computation
6.0 - 7.0 = -1.0
-1.0 + 12.0 = 11.0
3.0 * 11.0 = 33.0

Postfix Notation: 3 6 7 - + 12 *
Step by step computation
6.0 - 7.0 = -1.0
3.0 + -1.0 = 2.0
2.0 * 12.0 = 24.0

Postfix Notation: 3 6 - 7 12 + /
Step by step computation
3.0 - 6.0 = -3.0
7.0 + 12.0 = 19.0
-3.0 / 19.0 = -0.15789473684210525

Postfix Notation: 3 6 - 7 + 12 /
Step by step computation
3.0 - 6.0 = -3.0
-3.0 + 7.0 = 4.0
4.0 / 12.0 = 0.3333333333333333

Postfix Notation: 3 6 7 12 - + /
Step by step computation
7.0 - 12.0 = -5.0
6.0 + -5.0 = 1.0
3.0 / 1.0 = 3.0

Postfix Notation: 3 6 7 - 12 + /
Step by step computation
6.0 - 7.0 = -1.0
-1.0 + 12.0 = 11.0
3.0 / 11.0 = 0.2727272727272727

Postfix Notation: 3 6 7 - + 12 /
Step by step computation
6.0 - 7.0 = -1.0
3.0 + -1.0 = 2.0
2.0 / 12.0 = 0.16666666666666666
```

I checked all of these computations, including many more, to ensure that my method worked and the proper order of operations was done in postfix manner. As you can see, all of these are equal to what they should be, if you do the math out yourself. The next screenshot is all the permutations that equal 24, my second method of the PostFix class, and their step by step computations:

```
PostFix Notation: 3 4 - 12 13 + +
Steps to get solution
3.0 - 4.0 = -1.0
12.0 + 13.0 = 25.0
-1.0 + 25.0 = 24.0

PostFix Notation: 3 4 - 12 + 13 +
Steps to get solution
3.0 - 4.0 = -1.0
-1.0 + 12.0 = 11.0
11.0 + 13.0 = 24.0

PostFix Notation: 3 4 12 - - 13 +
Steps to get solution
4.0 - 12.0 = -8.0
3.0 - -8.0 = 11.0
11.0 + 13.0 = 24.0

PostFix Notation: 3 4 12 - 13 - -
Steps to get solution
4.0 - 12.0 = -8.0
-8.0 - 13.0 = -21.0
3.0 - -21.0 = 24.0

PostFix Notation: 3 5 + 6 9 - *
Steps to get solution
3.0 + 5.0 = 8.0
9.0 - 6.0 = 3.0
8.0 * 3.0 = 24.0

PostFix Notation: 3 5 6 + * 9 -
Steps to get solution
5.0 + 6.0 = 11.0
3.0 * 11.0 = 33.0
33.0 - 9.0 = 24.0

PostFix Notation: 3 5 6 - 9 + *
Steps to get solution
5.0 - 6.0 = -1.0
-1.0 + 9.0 = 8.0
3.0 * 8.0 = 24.0

PostFix Notation: 3 5 6 9 - - *
Steps to get solution
6.0 - 9.0 = -3.0
5.0 - -3.0 = 8.0
3.0 * 8.0 = 24.0
```

I also checked all of these computations, including many more, to ensure that my method worked and the proper order of operations was done in postfix manner. As you can see, all of these are equal to what they should be, if you do the math out yourself. These two last test cases confirm that my final methods are functional, as well as my stack class interface and implementation are working correctly. Lastly, the total number of 24 postfix evaluations, not displayed in the screenshot but displayed at the end of test run, is displayed and it is equal to 778. Granted that all my permutations were correct, which they are, and that my evaluations were correct, I'm mostly certain due to that I didn't individually solve each permutation as 228800 is way too much to do, I'm confident that the total number of possible postfix notations that equal 24 is 778.