Day 16 - Recursive Processing

Chapter 03, 3.3 – Recursive Processing of Arrays, Review Problem(s): ***10, 11***

Chapter 03, 3.4 – Recursive Processing of Linked Lists, Review Problem(s): ***15***

Chapter 03, 3.7 – Removing Recursion

Chapter 03, 3.8 – When to Use Recursion

"Because of their nested calls to themselves, recursive methods can be confusing to debug."

Binary search example 1, "Guess which number."

Binary search example 2, "Guess which page."

Chapter 03 PPT, pg 19-36.

Then, Chapter 03 PPT, pg 50-64.

Consider the mathematical expression: $3 + 5$

It is of the form ***operand*** ***operator*** ***operand***

We typically refer to this as "in-order" or ***Infix*** notation.

What if we were to instead, write this expression as ***Prefix*** notation, with
operator ***operand*** ***operand***

What would it look like?

$$+\,3\;\;5$$

Now, what if we consider that each operand could itself be another Prefix expression?

$$+\;/\;10\;\;2\;*5\;\;3$$

What does this expression evaluate to?

It looks like the subexpressions $/\;10\;\;2$ and $*\;5\;\;3$ have to be evaluated first, and then added.

How would you evaluate the expression, when one of the operators is moved?

$$+ / * 10 \;\; 2 \; 5 \;\; 3$$

It would be nice to have an algorithm to recursively evaluate a Prefix expression.

```
public int result()
{
     String token = expr.next();
     // is token an operator?
     if ( token is not an operator ) // it's a number
     {
          return String token converted to a number
     }

     //token must be an operator
     int a = result();
     int b = result();
     // convert token to char variable called op,
     // so the switch statement can be used
     switch(op)
     {
          case '+':
           calc = a + b;
           break;
          case '-':
           calc =  a - b;
           break;
          case '*':
           calc =   a * b;
           break;
          case '/':
           calc =   a / b;
           break;
     }
     return calc;
}
```

Try working through the algorithm with the first expression:

$+ / 10\ 2\ * 5\ 3$

a = `result(`$/ 10\ 2\ * 5\ 3$`)`

    a = `result(`$10\ 2\ * 5\ 3$`)`

    will return 10, a=10

    b = `result(`$2\ * 5\ 3$`)`

    will return 2, b=2

    The operator at this level of recursion is /, so 10 / 2 will be evaluated, as 5, which is returned, a =5

b = `result(`$* 5\ 3$`)`

    a = `result(`$5\ 3$`)`

    will return 5, a=5

    b = `result(`3`)`

    will return 3, b=3

    The operator at this level of recursion is $*$, so 5 * 3 will be evaluated, as 15, which is returned, b =15

Finally, all the recursive calls have been returned from, and we have the last calculation, a + b, where a is 5 and b is 15, which results in 20.

Now, work through the algorithm with the second expression:

$+ / * 10 \;\; 2\, 5 \;\; 3$

a = `result(`$/ * 10 \;\; 2\, 5 \;\; 3$`)`

a = `result(`$* 10 \;\; 2\, 5 \;\; 3$`)`

a = `result(`$10 \;\; 2\, 5 \;\; 3$`)`

will return 10, a=10

b = `result(`$2 \;\; 5 \;\; 3$`)`

will return 2, b=2

The operator at this level of recursion is $*$, so 10 * 2 will be evaluated, as 20, which is returned, a =20

b = `result(`$5 \;\; 3$`)`

will return 5, b=5

The operator at this level of recursion is /, so 20 / 5 will be evaluated, as 4, which is returned , a=4.

b = `result(`3`)`

will return 3, b=3

Finally, all the recursive calls have been returned from, and we have the last calculation, a + b, where a is 4 and b is 3, which results in 7.

HW 04