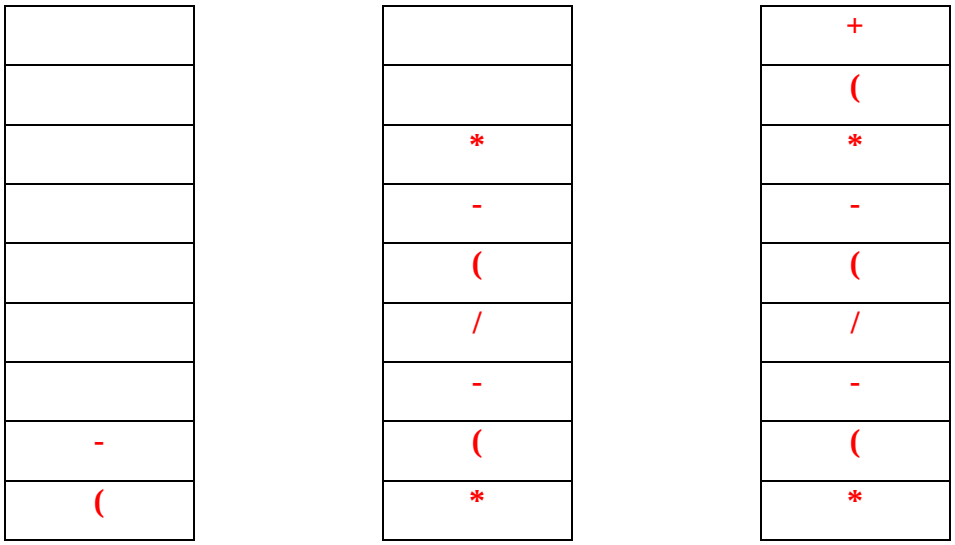
Spring 2017 COP 3502 Final Exam: Part B Solutions

Date: 4/27/2017

13) (10 pts) Convert the following infix expression to postfix, showing the contents of the operator stack at the three points A, B and C indicated during evaluating the expression and provide the converted postfix expression.

```
                    A                   B       C
( 3 + 6 ) * ( 7 -   27 / ( 19 - 2 *  ( 4 +  1 ) ) )
```



A B C

Resulting postfix expression:

3 6 + 7 27 19 2 4 1 + * - / - *

Grading: 1 pt first stack, 2 pts second stack, 2 pts third stack, 5 pts expression

14) (5 pts) Show the result of insertion sort after each iteration when sorting the array shown below:

Array	13	6	9	3	12	5	7
After 1st iteration	**6**	**13**	**9**	**3**	**12**	**5**	**7**
After 2nd iteration	**6**	**9**	**13**	**3**	**12**	**5**	**7**
After 3rd iteration	**3**	**6**	**9**	**13**	**12**	**5**	**7**
After 4th iteration	**3**	**6**	**9**	**12**	**13**	**5**	**7**
After 5th iteration	**3**	**5**	**6**	**9**	**12**	**13**	**7**
After last iteration	3	5	6	7	9	12	13

Grading: 1 pt per row, point is only earned if row is perfect, no exceptions.

15) (3 pts) Consider a Merge Sort running on an array of size n = 16, using the code shown in class (the base cases are arrays of size 0 and 1, otherwise recursive calls are made to Merge Sort). How many times will the Merge function get called during the sort?

Solution
On an array of size 2, Merge gets called 1 time. On an array of size 4, it gets called 3 times (once for the recursive call to Merge Sort on the left, once on the right, and once to put the two sides together). Similarly, on an array of size 8, Merge gets called 3 (left) + 3 (right) + 1 (together) = 7 times. Finally, on an array of size 16, Merge gets called 7 (left) + 7 (right) + 1 (together) = 15 times. In general, Merge gets called 2^n - 1 times on a Merge Sort of 2^n items.

15

Grading: 3 pts for 15, 2 pts for 7, 14 or 16, 1 pt for anything else within 8, 0 pts otherwise

16) (6 pts) Consider the following game. You are given a positive integer, *n*. Your goal is to change that integer to a number 0 or less. You are allowed two possible operations to reduce your number: (1) subtract 10 from it, (2) integer divide it by 3. Write a ***recursive*** function that calculates the minimum number of operations you have to apply in sequence to reduce your number to 0 or less. (For example, if your number was 19, you could divide it by 3 to obtain 6 and then subtract 10 to obtain -4, to obtain the goal in 2 steps. There is no way to do this in 1 step, so the correct answer is 2 for this case.) **Hint: recursively try both moves, see which one "wins" the game for you more quickly and build off that move.**

My Initial Intended Solution
```
int minMovesToWinSlow(int n) {
    if (n == 0) return 0;
    if (n <= 10) return 1;
    int div3 = minMovesToWinSlow(n/3) + 1;
    int sub10 = minMovesToWinSlow(n-10) + 1;
    return min(div3, sub10);
}
```

My Better Solution, Still Recursive
```
int minMovesToWin(int n) {
    if (n == 0) return 0;
    if (n <= 10) return 1;
    return minMovesToWin(n/3) + 1;
}
```

Grading: 4 pts base cases, 6 pts recursive call or calls.

17) (1 pt) What type of food is served at Blaze Pizza? **Pizza** **(Give to All)**