Programming assignments 3  20 Points

(Do NOT work in groups for programming assignments.)

A cryptarithm is a puzzle in which letters are substituted for numbers in an equation. We assume that no two letters have the same value. The term was coined by Maurice Vatriquant in 'Sphinx' magazine in 1931. The following are examples of cryptarithms:

\[
\begin{array}{c}
S E N D \quad 5 4 7 8 \\
M O R E \quad 1 6 2 4 \\
G O L D \quad 9 6 3 8 \\
M O N E Y \quad 1 6 7 4 0 \\
\end{array}
\]

\[
\begin{array}{c}
V E N U S \quad 5 4 7 3 9 \\
- E A R T H \quad 4 6 1 2 0 \\
M A R S \quad 8 6 1 9 \\
\end{array}
\]

In this assignment, you are to use Ruby to find the solution for cryptarithms. Our puzzles will be stated as a string. For example, puzzle = “SEND+MORE+GOLD==MONEY”; You will write the solution as: 5478+1624+9638==16740

**Part 1 Test a specific solution**

Input: (test for five problems shown above)
- puzzle: “SEND+MORE+GOLD==MONEY”;
- solution String: YMRLESONDG

Output:
- Original Puzzle: “SEND+MORE+GOLD==MONEY”
- Solution String: YMRLESONDG
- Puzzle with substitution: "5478+1624+9638==16740"  CORRECT (or possibly NOT CORRECT)

Note: The solution string is the ordering of the letters such that the first character in the string is 0, the second is 1, etc. So, for “SEND+MORE+GOLD==MONEY”, the solution string is "YMRLESONDG", as Y is a 0, M is a 1, R is a 2, etc

**Hints:**

Use the ruby command tr to replace the characters of the puzzle with numbers.
The syntax of `tr` is:
```
str.tr(from_str, to_str) \rightarrow new_str
```
Returns a copy of `str` with the characters in `from_str` replaced by the corresponding characters in `to_str`. If `to_str` is shorter than `from_str`, it is padded with its last character.

Use the Ruby command `eval` to evaluation the puzzle (after substitution)
```
res = eval(puzzle)
```

A number that begins with a zero causes a problem in `eval` (likely because it thinks it is an octal number), so we'll ignore those cases.

If `eval` returns true, we have found a correct solution.

**Part 2 Computer finds the solution**

Finish the assignment by finding a solution to a provided puzzle. You can do this any way you like. The following is only a suggestion. You do not need to solve the problem this way.

**Suggestion:**
Find all the unique letters of your puzzle.

Write a recursive function to permute the unique letters of the puzzle so that all possible permutations are found. Try them (one at a time) as a solution string.

Note, that you do not expect that finding all permutations will run quickly as the number of permutations is ten factorial! Show that your permutation routine works by showing all permutations of “ABCD”. (This is a good testing technique. Try with something small so you can follow the recursion and it doesn’t take forever.)

The following is a C++ version of the permute algorithm.

```
// v is the array of integers to be arranged
// start is the current location in the array (you are only to worry about permuting from start to the end)
// n is the length of the array
//Given an array of v with the elements before start fixed, print out all copies of v with the
//remaining elements permuted.
void permute(int v[], int start, int n) {
    if (start == n-1) {
        print(v, n);
    }
    else {
        for (int i = start; i < n; i++) {
            int tmp = v[i];
            v[i] = v[start];
            permute(v, i, n);
            v[i] = tmp;
        }
    }
}
```
v[start] = tmp;
permute(v, start+1, n);
v[start] = v[i];
v[i] = tmp;
}
}

Try each permutation of your unique letters until a solution is found or no solution is possible. This is slow, so put your feet up and take a nap while you churn out the answer. During debugging, I found myself wanting some feedback from the brute force trials. I did this via a statement which printed some feedback every 500 times.