Lab 10: Optional Extra Credit
Compiler for a Simple Prefix Language

Points

Completing this optional assignment will add **10 points** to your final exam score (out of 100).

Instructions & Deliverable

Develop a compiler for the fictitious programming language described in any language of your choice. Submit your source code / project as well as examples (screen shots) of your testing output.

Introduction & Background

LUK is a simple prefix language. A sample expression in LUK looks like:

```
mul 3 sub 2 sum 1 3 4
```

For simplicity, there are a few rules we need to follow:

1. We have only the functions: mul, sub, sum, div.
2. Each individual string token is surrounded by whitespace (spread operator).
3. We support only natural numbers.

mul accepts multiple operands, passed with the spread operator. The function just multiplies all of them, so for instance: mul(2, 3, 4) = 24. sub respectively subtracts the passed arguments and sum sums them.

The expression above can be translated to the following expression:

```
mul(3, sub(2, sum(1, 3, 4)))
```

or…

```
3 * (2 - (1 + 3 + 4))
```

Grammar

```
num := 0-9+
op := sum | sub | div | mul
expr := num | op expr+
```

This translated to plain English, means:

- `num` can be any sequence of the numbers between 0 and 9.

- `op` can be any of `sum`, `sub`, `div`, `mul`.

- `expr` can be either a number (i.e. `num`) or an operation followed by one or more `expr`s.

Notice that `expr` has a recursive declaration.

Additional Example

The LUK expression:

```
sub 2 sum 1 2 3
```

Produces the following mathematical expression:

```
sub(2, sum(1, 2, 3)))    or 2 - (1 + 2 + 3)
```

Produces the following tree:

```
   sub
   /   \
  2    sum
         /|\
        1 3 4
```