1. Goal
The goal of this assignment is to build an expression tree and then use
the tree to compute Bikini Bottom Residents taxes.

2. Problem Statement [client’s statement of their need]
The Bikini Bottom Civic Association (BCCA) needs an expression evaluator so they can correctly compute Sponge Bob’s taxes for him. Also the program needs to output the expression in three different formats to satisfy various accounting standards.

3. Requirements Analysis [What is the client’s problem?]
Q: What kinds of expressions?
A: Expression involve integers, and the operators + (addition), − (subtraction), * (multiplication), / (division), and neg (negation).
Q: What needs to be done with each expression?
A: 1) for Sponge Bob, evaluate the expression
2) for Patrick’s accountant print it out the expression prefix notation
3) for Squidward’s accountant print it out the expression postfix notation
4) for Mr. Krabs’s accountant print it out the expression infix notation

For example given an expression that adds 1 and 2, the outputs would be:
1) “3”
2) “+ 1 2”
3) “1 2 +”
4) “(1 + 2)”

4. Design [How]
[This I’ll leave up to you. I’m happy to look at your design and corresponding build plan if you like.]
I will also encourage some upfront OO thought, which will pay off in the end. To this end, I’m happy to look at your UML diagram.

What to hand in
(1) A print out of your analysis and design work.
(2) A well-formatted 2-up printout of your source code.
(3) An email with a compressed tarball of your analysis, design, code, and spreadsheet named <your-name-no-spaces>.tar.gz (send me this and only this file!)
attached and (optionally this time) the URL of your published JavaDoc. Something like
tar -cvf - <files> | gzip > <your-name-no-spaces>.tar.gz.

Assignment Requirements [part of being a course rather than software development]
• You must implement an expression tree and include appropriate node kinds.
• You can hard code several expressions into your program for testing.

• Create your code to be part of a package named `asn5`.

• Use clear documentation and careful formatting. Be consistent in indentation and alignment of braces. Each open brace “{” **must** be on its own line. Intend 2 or 4 spaces as the most.

• Each source code file must start with

  ```
  // This is my code
  // <Your Name>
  // CS312
  ```

**Notes**

• To get full credit output only required parenthesis. For example, `(2 + 3) * 4` is more naturally written `2 + 3 * 4` and `(4 * 5) + 6` is more naturally written `4 * 5 + 6`.

• In class we may write code to find a specific operator using either a DFS or a BFS. We may also write an iterator that iterates through the integers contained in a tree.