Vedansh Shrivastava – 1998870
Advanced Software Engineering – Assignment 2
Exercise 2.2 – Grammar-based Fuzzing

Platform tested on: Windows (Command Prompt)

---
1) Grammar (see grammar.txt)

(See attached grammar.txt for the context-free grammar used to generate arithmetic expressions, including a few malformed productions for fuzzing.)

---
2) Example inputs

A. Valid expression (one example)

   (3+4)*2

   Explanation:
   - Well-formed arithmetic expression using parentheses, binary + and *, and integer literals.
   - This should be accepted by a correct parser and evaluate to 14.

B. Malformed expression 1 – Missing operand after operator
   ---
   5 +

   Explanation / what it reveals:
   - The input ends with a binary operator '+' and no operand following it.
   - This can expose parser issues in error recovery or incomplete-input handling.
     A robust parser should report a clear syntax error (e.g., "expected expression after '+'")
     rather than crash or attempt to read uninitialized data.

C. Malformed expression 2 – Unmatched / missing closing parenthesis
   ---
   (2 + 3 * (4 - 1

   Explanation / what it reveals:
   - There is a missing closing parenthesis for the nested subexpression.
   - This can reveal problems with recursive descent parsers (stack/recursion handling),
     or edge cases in parenthesis matching logic and error messages. It may also
     uncover off-by-one or buffer handling bugs where the parser assumes balanced parentheses.

Notes

- The grammar supports integer literals (decimal), binary operators '+' and '*', parentheses,
  and an optional unary minus (e.g., -5).
- The grammar also contains a few optional malformed productions (in grammar.txt) so a
  generator can intentionally produce slightly malformed inputs (missing operands, extra operators,
  or unmatched parentheses) to test parser robustness.
- The provided examples are short, focused, and meant to illustrate likely failure modes in a parser.