Lab2B-Sample Test Cases

The following test cases illustrate the **minimum expected coverage** of your lexer and parser. Your implementation must handle these correctly.

Note:

- These are basic testcases and totally **not exhaustive**.
- During evaluation, your code may be run on **more complex and deeply nested programs**.
- [Optional] You may consider implementing a **print** function to test the outputs of your program wherever valid and use the AST for debugging and ensure correctness.

A. Valid Test Cases [Non exhaustive list]

Test 1: Variable Declaration

```
var x;
```

Test 2: Declaration with Initialization

```
var x = 10;
```

Test 3: Assignment

```
var x;
x = 5;
```

Test 4: Arithmetic Expression

```
var x;
x = 5 + 3 * 2;
```

Test 5: Comparison Expression

```
var x = 10;
var y = 20;
x = x < y;
```

Test 6: If Statement

```
var x = 5;
if (x > 3)
  x = x - 1;
```

Test 7: If–Else Statement

```
var x = 10;
if (x < 5)
  x = 0;
```

```
else
  x = 1;
```

Test 8: While Loop

```
var i = 0;
while (i < 5)
  i = i + 1;
```

Test 9: Block with Nested Statements

```
var x = 0;
{
  var y = 1;
  while (x < 3) {
    if (x == 1)
      y = y + 1;
    else
      y = y + 2;
    x = x + 1;
  }
}
```

B. Invalid Test Cases (Must Fail Gracefully) [Non Exhaustive list]

Test 10: Use Before Declaration

```
x = 10;
```

Expected: error indicating undeclared variable.

Test 11: Missing Semicolon

```
var x = 10
x = 5;
```

Expected: syntax error with line number.

Test 12: Malformed Expression

```
var x = 5 + ;
```

Expected: syntax error near `;`.

Test 13: Unmatched Braces

```
var x = 5;
if (x > 0) {
  x = x - 1;
```

Expected: parse failure due to missing `}`.

Test 14: Invalid Assignment Target

```
var x = 5;
(x + 1) = 10;
```

Expected: syntax or semantic error.

C. Error Handling Expectations

- Parser must not crash on invalid input.
- Errors should report:
  - Line number (where possible)
  - Unexpected token
- Error recovery is optional but encouraged.

Sample complex test case

```
var i = 0;
var sum = 0;
var limit = 10;

while (i < limit) {
  if (i == 0)
    sum = sum + 1;
  else {
    if (i < 5)
      sum = sum + i * 2;
    else
      sum = sum + (i + 1) * (i - 1);
  }
  i = i + 1;
}
```

Final Note

Your implementation will be tested on **larger and more complex programs** involving deep nesting and multiple constructs. Ensure that your grammar, lexer, and AST construction are robust and general.