Programming Exercises: ChessBoard Array & Polynomial

Exercise 1: ChessBoard Array

Objective: Implement a chessboard-style array where only "white squares" can store non-zero values.

Requirements:

- **Class**: `ChessBoardArray` with nested helper classes `Row` and `ConstRow`.
- **Indexing**: Throw `out_of_range` for invalid indices or black squares.
- **Printing**: Use `setw(4)` for alignment.

Example Code:

```
int main() {
    ChessBoardArray a(4, 1); // 4x4 array, indices start at 1
    a[3][1] = 42;
    a[4][4] = 17;
    try { a[2][1] = 7; }  // Black square → exception
    catch(out_of_range &e) { cout << "a[2][1] is black" << endl; }
    cout << a;
}
```

Expected Output:

```
a[2][1] is black
   0    0    0    0
   0    0    0    0
  42    0    0    0
   0    0    0   17
```

Exercise 2: Polynomial as a Sorted Linked List

Objective: Represent polynomials using a sorted linked list (descending exponents).

Requirements:

- **Class**: `Polynomial` with nested `Term` nodes.
- **Rules**: No duplicate exponents or zero coefficients.
- **Operations**: Addition, multiplication, evaluation.

Example Code:

```
int main() {
    Polynomial p;
    p.addTerm(1, 3);  // 3x
    p.addTerm(2, 1);  // x²
    p.addTerm(0, -1); // -1
    Polynomial q(p);
```

```
    q.addTerm(1, -3); // q = x² - 1
    cout << "P(x) = " << p << endl;
    cout << "P(1) = " << p.evaluate(1);
}
```

Expected Output:

```
P(x) = x^2 + 3x - 1
P(1) = 3
```
