Tarea Independiente 08/09/2025

David Núñez Franco

September 9, 2025

Inventario de Conceptos Claves

- Ambiguedad en expresiones
- Precedencia
- Asociatividad
- Caso de verbo a sustantivo
- Visitable + Visitor

(

Ejercicio 0)

Haga AstToString un visitor de Node que retorna lo mismo que el toString

Solución

```
        /*Ejercicio 0 T.I 8/9*/
        export class AstToString extends Visitor {
            visit(node) {
                if (node == null) return "";

                //Hojas
                if (node instanceof Num) return String(node.value);
                if (node instanceof Id) return String(node.name);
                if (node instanceof Oper) return String(node.name);

                // Unary: op + expr
                if (node instanceof UnaryOp) {
                    const op = node.oper.accept(this);
                    const e = node.expr.accept(this);
                    return op + e;
```

```
            }

            // Binary: L O R
            if (node instanceof BinaryOp) {
                const L = node.left.accept(this);
                const O = node.oper.accept(this);
                const R = node.right.accept(this);
                return `(${L} ${O} ${R})`;
            }

            // Classic operation: (op, arg1, arg2...)
            if (node instanceof Operation) {
                const items = [node.oper, ...node.args].map((x)
                    =>
                x instanceof Node ? x.accept(this) : String(x)
                );
                return `(${items.join(",")})`;
            }

            // Generic Node: just showing head and children
            if (node instanceof Node) {
                const items = [node.head, ...node.children].map
                    ((x) =>
                x instanceof Node ? x.accept(this) : String(x)
                );
                return `(${items.join(",")})`;
            }
        }
    }

    function test_ast_to_string() {
        const v = new AstToString();

        const n = new Node("add", 1, 1, 2, 3);
        console.log("AstToString(Node)      =", n.accept(v)); //
            (add, 1, 1, 2, 3)

        const num = new Num(666);
        console.log("AstToString(Num)       =", num.accept(v));
           // 666

        const id = new Id("x");
        console.log("AstToString(Id)        =", id.accept(v));
           // x

        const plus = new Oper("+");
        const addOp = new Operation(plus, id, num);
```

```
58              console.log("AstToString(Operation) =", addOp.accept(v))
                    ; // (+, x, 666)
59
60              const minus = new Oper("-");
61              const negx = new UnaryOp(minus, id);
62              console.log("AstToString(UnaryOp)   =", negx.accept(v));
                    // -x
63
64              const addBin = new BinaryOp(plus, id, num);
65              console.log("AstToString(BinaryOp)  =", addBin.accept(v)
                    ); // (x + 666)
66          }
```

Listing 1: AstToString and TestCase

Ejercicio 3

3. Usando un Visitor genere un string que representa
a) La notación prefija de Node
b) La notación postfija de Node

Solución

```
1           export class PrefixNotation extends Visitor {
2               visit(node) {
3                   if (node == null) return "";
4
5                   if (node instanceof Num) return String(node.value);
6                   if (node instanceof Oper) return String(node.name);
7                   if (node instanceof Id) return String(node.name);
8
9                   if (node instanceof UnaryOp) {
10                      const op = node.oper.accept(this);
11                      const e = node.expr.accept(this);
12                      const opname = op === "-" ? "neg" : op === "+" ?
                            "pos" : op;
13                      return `(${opname} ${e})`;
14                  }
15
16                  if (node instanceof BinaryOp) {
17                      const op = node.oper.accept(this);
18                      const L = node.left.accept(this);
19                      const R = node.right.accept(this);
20                      return `(${op} ${L} ${R})`;
21                  }
```

```
22
23                  if (node instanceof Operation) {
24                      const op = node.oper.accept(this);
25                      const items = node.args.map((a) => a.accept(this
                            ));
26                      return `(${op} ${items.join(", ")})`;
27                  }
28
29                  // Generic node
30                  const items = [node.head, ...node.children].map((x)
                        =>
31                  x instanceof Node ? x.accept(this) : String(x)
32                  );
33                  return `(${parts.join(", ")})`;
34              }
35          }
```

Listing 2: PrefixNotation

```
1           export class PostfixNotation extends Visitor {
2               visit(node) {
3                   if (node == null) return "";
4
5                   if (node instanceof Num) return String(node.value);
6                   if (node instanceof Oper) return String(node.name);
7                   if (node instanceof Id) return String(node.name);
8
9                   if (node instanceof UnaryOp) {
10                      const op = node.oper.accept(this);
11                      const e = node.expr.accept(this);
12                      const opname = op === "-" ? "neg" : op === "+" ?
                            "pos" : op;
13                      return `(${e} ${opname})`;
14                  }
15
16                  if (node instanceof BinaryOp) {
17                      const op = node.oper.accept(this);
18                      const L = node.left.accept(this);
19                      const R = node.right.accept(this);
20                      return `(${L} ${R} ${op})`;
21                  }
22
23                  if (node instanceof Operation) {
24                      const op = node.oper.accept(this);
25                      const items = node.args.map((a) => a.accept(this
                            ));
26                      return `(${items.join(", ")} ${op})`;
27                  }
```

```
28
29                  // Generic node
30                  const items = [node.head, ...node.children].map((x)
                        =>
31                  x instanceof Node ? x.accept(this) : String(x)
32                  );
33                  if (items.length === 0) return "()";
34                  const [head, ...rest] = items;
35                  return `(${rest.join(" ")} ${head})`;
36              }
37          }
```

Listing 3: PostfixNotation

```
1           function test_notations() {
2               const x = new Id("x");
3               const n3 = new Num(3);
4               const n5 = new Num(5);
5               const plus = new Oper("+");
6               const minus = new Oper("-");
7               const mul = new Oper("*");
8
9               const add = new BinaryOp(plus, x, n3); // x + 3
10              const prod = new BinaryOp(mul, add, n5); // (x + 3) * 5
11              const negx = new UnaryOp(minus, x); // -x
12
13              const Pre = new PrefixNotation();
14              const Post = new PostfixNotation();
15
16              console.log("PREFIX  add  =", add.accept(Pre)); // (+ x
                    3)
17              console.log("POSTFIX add  =", add.accept(Post)); // (x 3
                     +)
18              console.log("PREFIX  prod =", prod.accept(Pre)); // (*
                    (+ x 3) 5)
19              console.log("POSTFIX prod =", prod.accept(Post)); // (x
                    3 + 5 *)
20              console.log("PREFIX  -x   =", negx.accept(Pre)); // (neg
                     x)
21              console.log("POSTFIX -x   =", negx.accept(Post)); // (x
                    neg)
22          }
```

Listing 4: TestAnotations