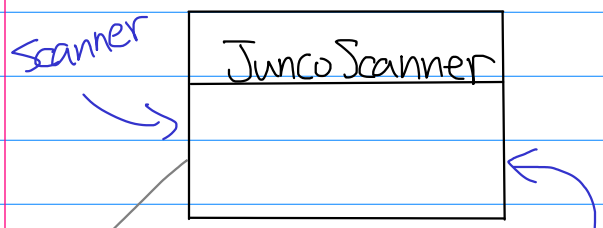
JuncoCompiler.java

main:

compile

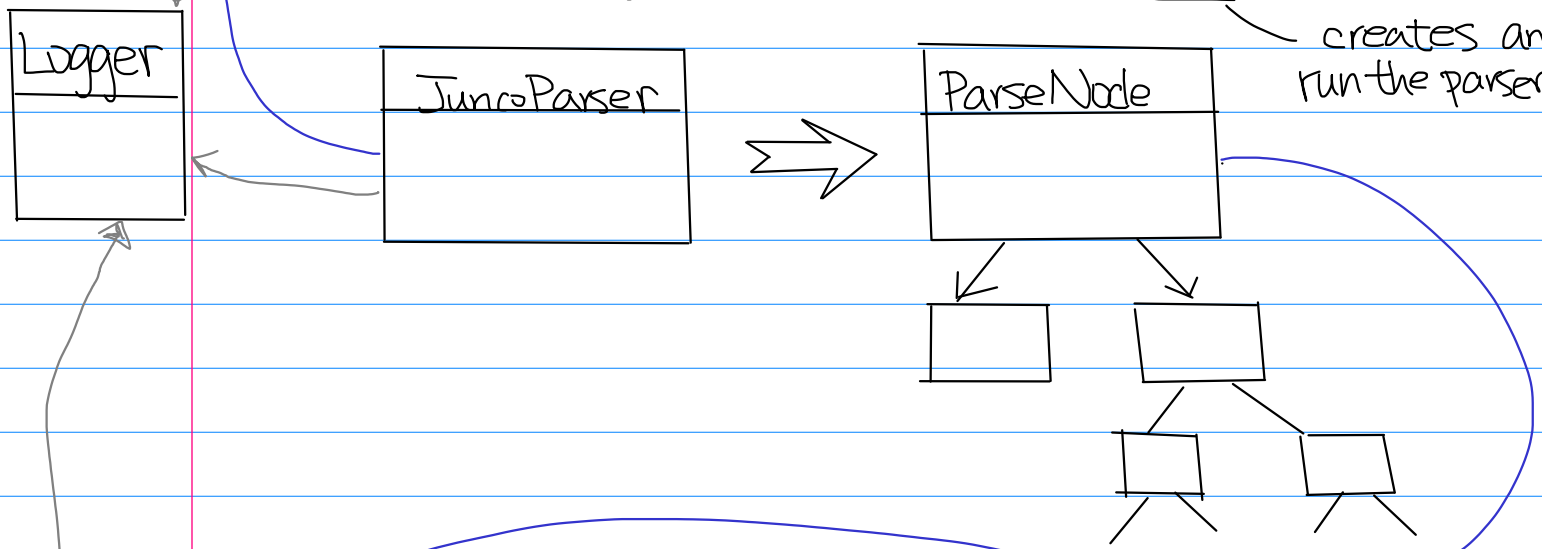
① Scanner scanner = JuncoScanner.make(filename)



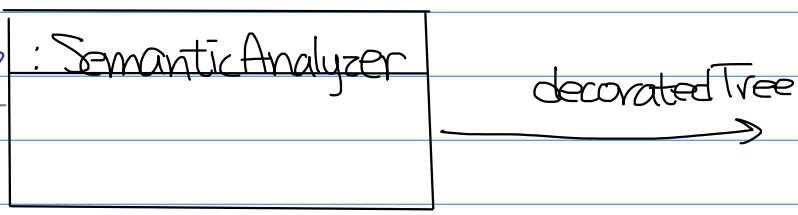
creates an instance of a Scanner

② ParseNode syntaxTree = JuncoParser.Parse(scanner);

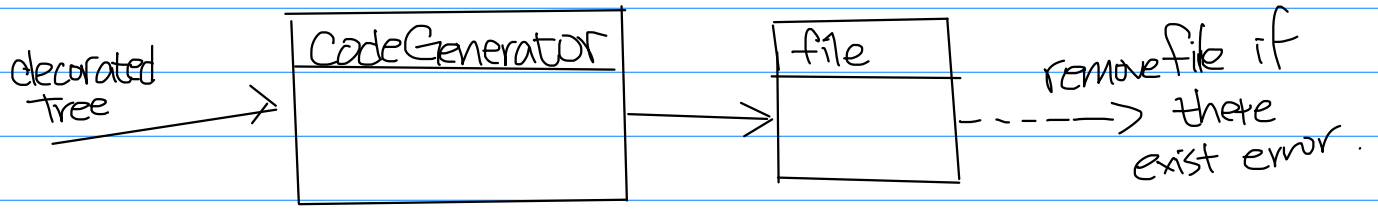
creates and run the parser!



③ ParseNode decoratedTree =
JuncoSemanticAnalyzer.analyze(syntaxTree);



④ generateCodeIfNoError(decoratedTree);

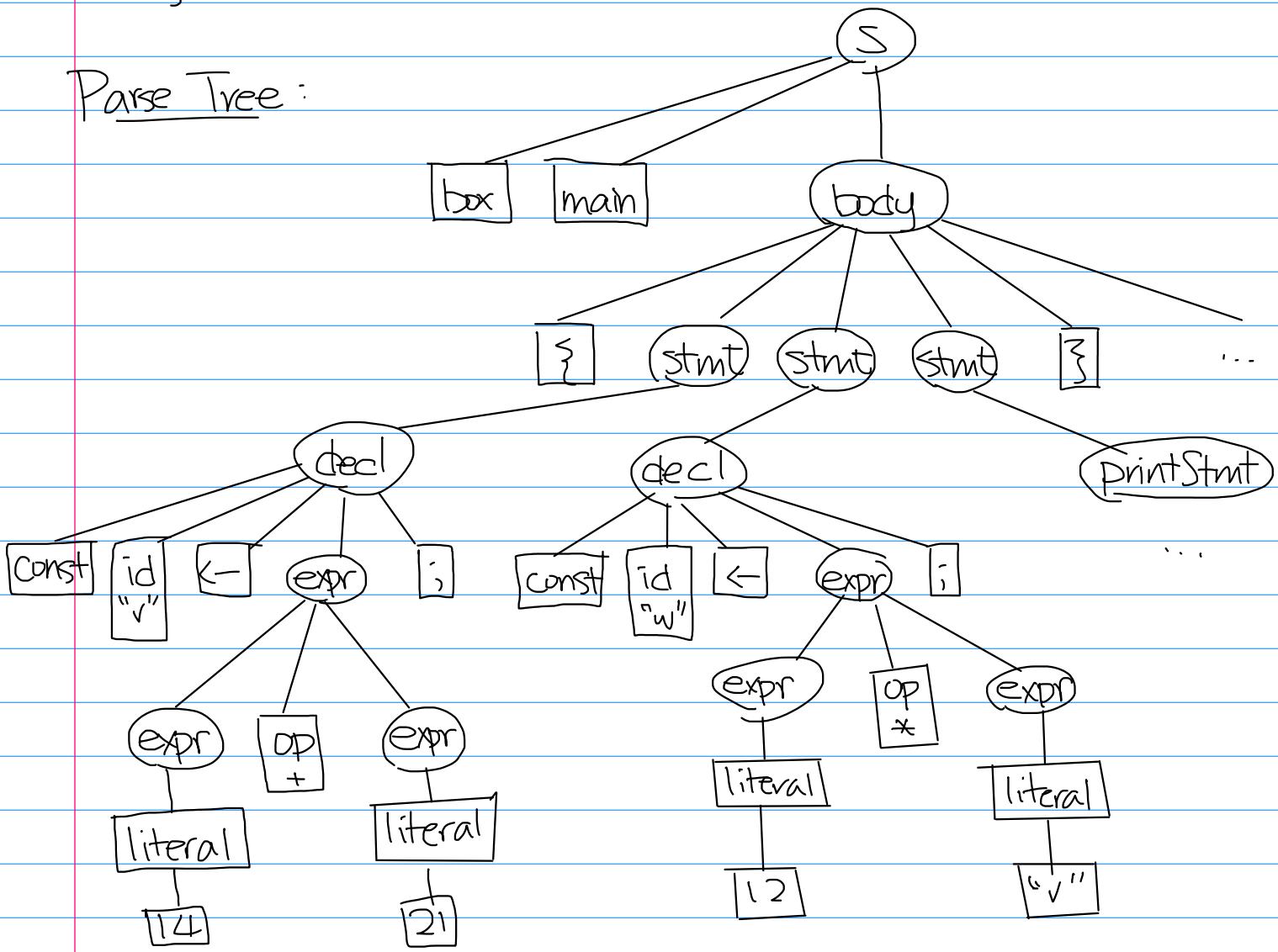


Sample Junco Program:

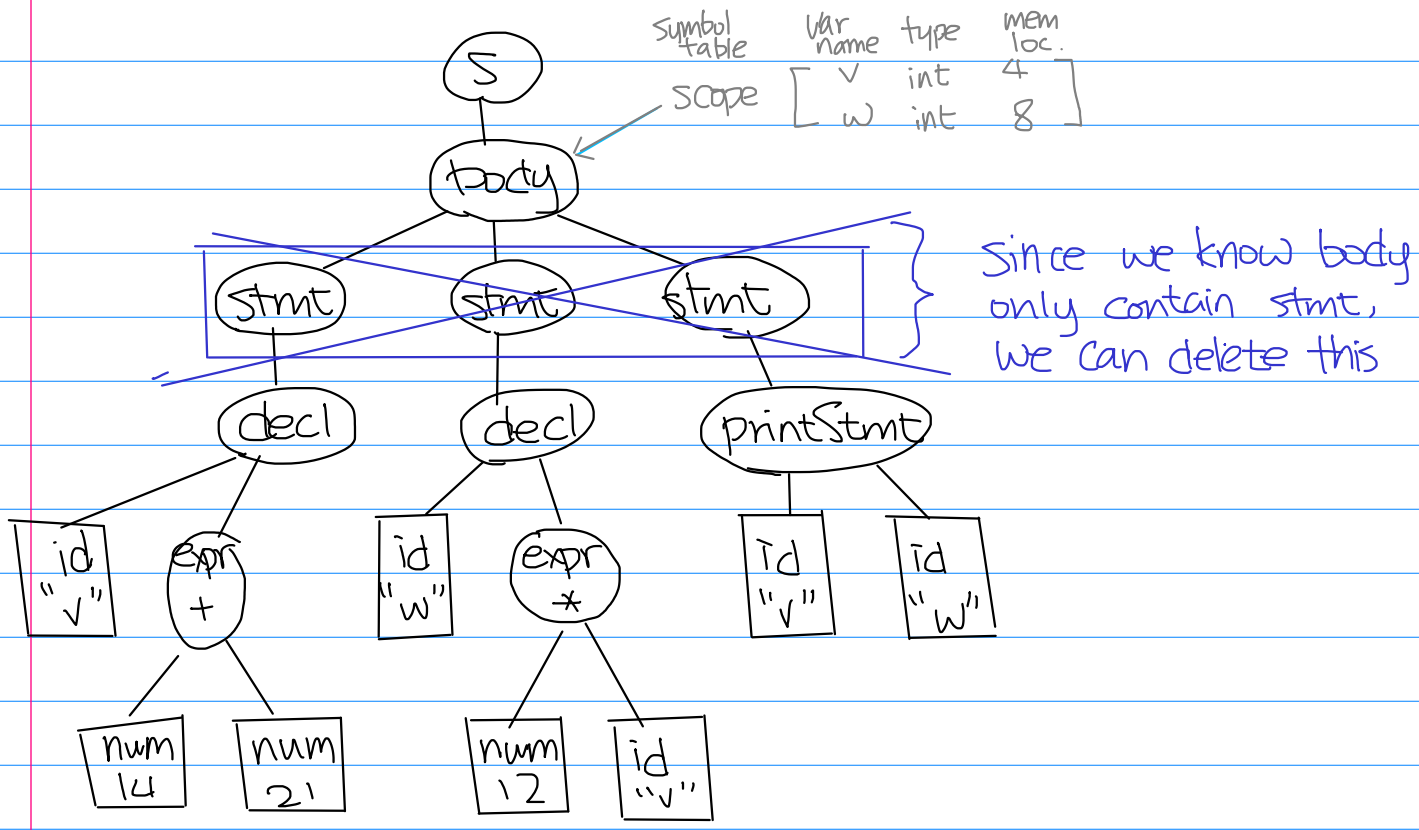
```
box main {
    const v ← 14 + 21;
    const w ← 12 * v;
    print v, w $;
}
```

S → box main body
body → { stmt* }
stmt → decl | printStmt
decl → const id ← expr ;

Parse Tree:



Abstract Syntax Tree (AST): keep only the circles from parse tree



Main Idea on Code Generation

* convert expr to postfix

$(a + bc) * (d + ef) \Rightarrow \alpha \ \beta \ *$

α β

γ

1. Compute the value of α
2. Compute the value of β
3. Multiply $\alpha * \beta$ (Compute value of γ)

Types of Code: ① address ② value ③ void (put on stack)