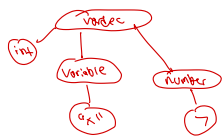
Parsing

vardec int x 7

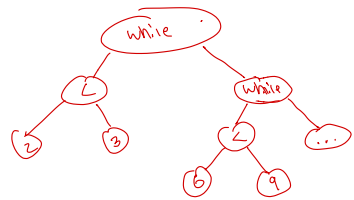
Tokens
LeftParenToken
VardecToken
IntTOken
IdentifierToken("x")
NumberToken(7)
RightParenToken

Parser →

Abstract Syntax Tree



(while (< 2 3)
(while (< 6 9)
...)



Stmt
- statement

interface Type
class IntType
class BoolType
interface Stmt
class VardecStmt
class LoopStmt
class AssignStmt
interface Exp
class NumberLiteralExp
class BooleanLiteralExp
class BinaryOperatorExp
interface Op
class PlusOp
class MinusOp
class LogicalAndOp
class LogicalOrOp
class LessThanOp
- class Program

```
public class VarDecStmt implements {
    Public final Type type;
    Public final Variable variable;
    Public final Exp exp;
```

```
interface AST
- Class IntType
- Class BoolType
  -Class VarDecStmt
  - Class LoopStmt
```

```
Public Class VarDec
  implements AST {
  Public final AST type
  Public final AST variable;
  Public final AST exp;
```

Abstract Grammar

Recursive descent parsing
Parser generators (ANTLR)
Parser combinators

1.) Left recursion
exp ::= num | exp + exp
concrete grammar
2) precedence
1 + 2 * 3

(+ 1 (* 3 2))

// S-expressions: LISP, Scheme, Racket, Clojure
(while (< 7 4)
  (while (< 3 2) .))

Parsing: The solved problem that isn't