There are two additional items required to completely specify the grammar of the language. First, the if/else

WhileStatement

Claite Grammar

IfStatement

Conjunction

Assignment

Expression

Statements

Program

Addition

Boolean

Integer

AddOp

RelOp

Letter

Digit

Char

Expression

Block

IfStatement

Statement

WhileStatement

EquOp

Relation

Addition

AddOp

Term

MulOp

Factor

UnaryOp

Primary

Identifier

Letter

Digit

Literal

Integer

Float

Char

Boolean

Clite Grammar (from Tucker and Noonan, 2007)
Discussion

- Are there any items required to completely specify the grammar?
Stages of Compilation

Source program

Lexical analyzer

Tokens

Syntactic analyzer

Abstract syntax

Semantic analyzer

Intermediate code (IC)

Code optimizer

Intermediate code (IC)

Code generator

Machine code

**Generate a sequence of tokens:** catch illegal characters

**Generate a parse tree:** catch syntax errors and ill-formed expressions

**Generate a detailed, explicit parse tree:** catch semantic errors

**Generate an improved IC:** speed up code execution

**Generate the target machine code**
Concrete Syntax and Concrete Syntax Tree

Concrete grammar for Assignment and Expression (Tucker and Noonan, 2007).

Assignment → Identifier [ [ Expression ] ] = Expression;

Expression → Conjunction { || Conjunction }
Conjunction → Equality { && Equality }
   Equality → Relation [ EquOp Relation ]
   EquOp → == | !=
Relation → Addition [ RelOp Addition ]
   RelOp →< | <= | > | >=
Addition → Term { AddOp Term }
   AddOp → + | -
Term → Factor { MulOp Factor }
   MulOp → * | / | %
Factor → [ UnaryOp ] Primary
UnaryOp → - | !
Primary → Identifier [ [ Expression ] ] | Literal |
   ( Expression ) | Type ( Expression )

\[ a = b + 3 \cdot c; \]
Simplify Convert Concrete Syntax Tree

- Discard all separator or terminating symbols
- Discard all nonterminals that are trivial roots. A trivial root is a symbol with only one subtree
- Replace all remaining nonterminals with the operators which are a leaf or one of their immediate subtrees
Abstract Syntax

Abstract syntax is a notation that allows parser to remove nonessential symbols and generate a tree that contains only the essential elements of the computation.

- Abstract syntax can be defined in the form: \( Lhs = Rhs \)

- \( Lhs \): the name of an abstract syntactic class

- \( Rhs \): defines the class as:
  - A list of one or more alternatives
  - A list of essential components that define a member of that class, separated by semicolons (;)
Concrete VS. Abstract

Abstract grammar for Conditional, Assignment and Expression

Concrete grammar for IfStatement, Assignment and Expression
Abstract Syntax Tree

- Class with a list of essential components has an associated node with as many fields as there are distinct elements

<table>
<thead>
<tr>
<th>Conditional</th>
<th>test</th>
<th>thenbranch</th>
<th>elsebranch</th>
</tr>
</thead>
</table>

- Class with a list of one or more alternatives has no associated node

\[ a = b + 3 \times c; \]

Conditional = Expression test; Statement thenbranch, elsebranch
Assignment = Variable target; Expression source
Expression = Variable | Value | Binary | Unary
  Binary = Operator op; Expression term1, term1
  Unary = Operator op; Expression term
Variable = String id
  Value = IntValue | BoolValue | FloatValue | CharValue
    IntValue = Integer value
    BoolValue = Boolean value
    FloatValue = Float value
    CharValue = Char value
  Operator = + | - | * | / | ! | == | != | < | > | <= | >= | && | ||