PLO Compiler Example

February 22, 2021

Example: Lexical Analysis

Input (sequence of characters):

$$\textbf{if } x < 0 \textbf{ then } z := -x \textbf{ else } z := x$$

Output (sequence of **lexical tokens**):

KW_IF, *IDENTIFIER*(“x”), *LESS*, *NUMBER*(0), *KW_THEN*,
IDENTIFIER(“z”), *ASSIGN*, *MINUS*, *IDENTIFIER*(“x”), *KW_ELSE*,
IDENTIFIER(“z”), *ASSIGN*, *IDENTIFIER*(“x”)

Example: Concrete syntax tree

$$\textbf{if } x < 0 \textbf{ then } z := -x \textbf{ else } z := x$$

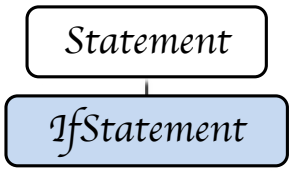
The **grammar** of the language can be used to extract the **structure** of the program statement (from its sequence of lexical tokens).

The structure can be described using a **concrete syntax tree**.

if $x < 0$ **then** $z := -x$ **else** $z := x$

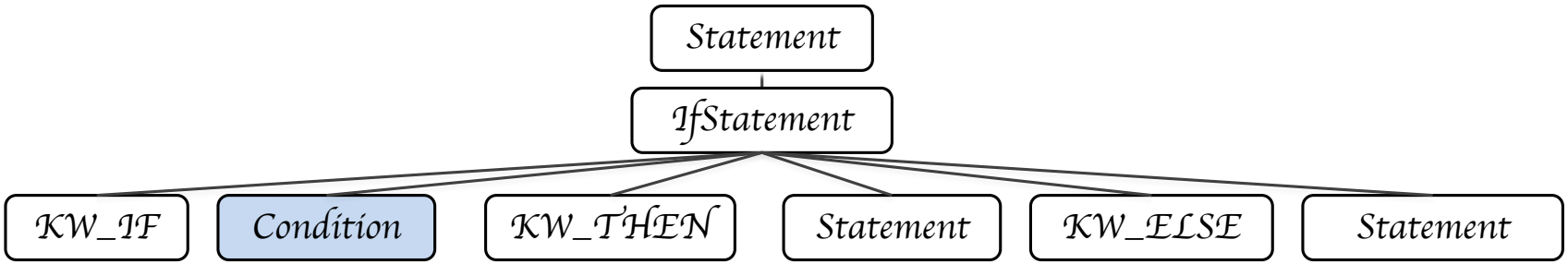
$$
\begin{aligned}
\textit{Statement} &\rightarrow \textit{Assignment} \mid \textit{CallStatement} \mid \textit{ReadStatement} \mid \textit{WriteStatement} \mid \\
&\quad \textit{WhileStatement} \mid \textit{IfStatement} \mid \textit{CompoundStatement} \\
&\ldots \\
\textit{IfStatement} &\rightarrow \textit{KW\_IF Condition KW\_THEN Statement KW\_ELSE Statement} \\
\textit{Condition} &\rightarrow \textit{RelCondition} \\
\textit{RelCondition} &\rightarrow \textit{Exp} \; [\; \textit{RelOp Exp} \;] \\
\textit{Exp} &\rightarrow \ldots
\end{aligned}
$$

if $x < 0$ **then** $z := -x$ **else** $z := x$



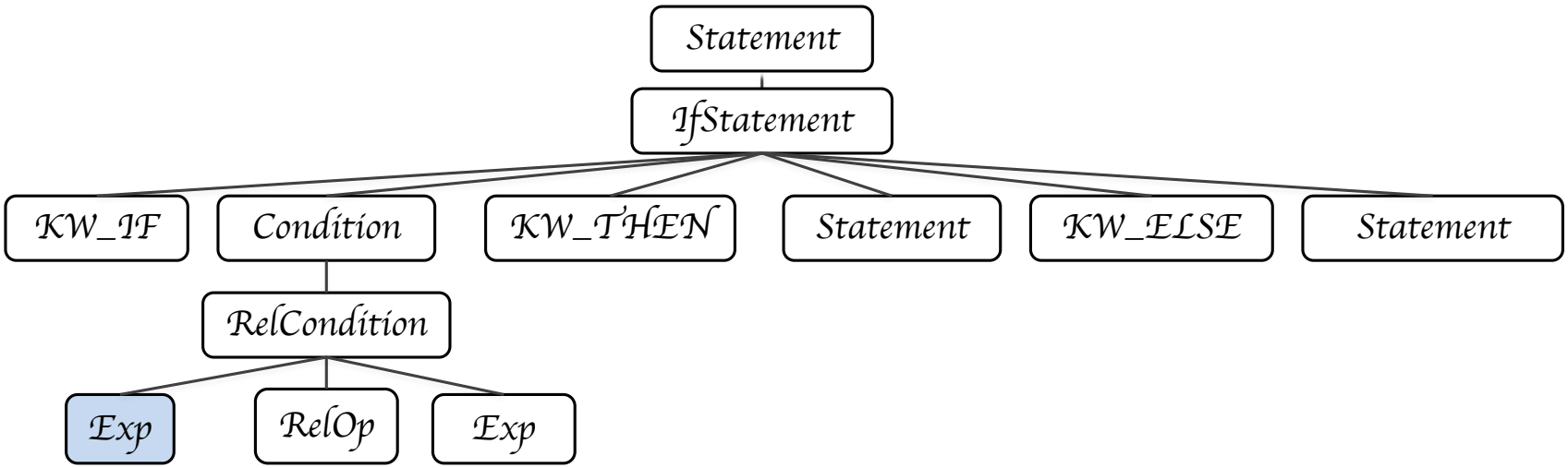
$$
\begin{aligned}
\textit{Statement} \rightarrow\ & \textit{Assignment} \mid \textit{CallStatement} \mid \textit{ReadStatement} \mid \textit{WriteStatement} \mid \\
& \textit{WhileStatement} \mid \textit{IfStatement} \mid \textit{CompoundStatement} \\
\ldots & \\
\textit{IfStatement} \rightarrow\ & \textit{KW\_IF Condition KW\_THEN Statement KW\_ELSE Statement} \\
\textit{Condition} \rightarrow\ & \textit{RelCondition} \\
\textit{RelCondition} \rightarrow\ & \textit{Exp} \ [\ \textit{RelOp Exp}\ ] \\
\textit{Exp} \rightarrow\ & \ldots
\end{aligned}
$$

if $x < 0$ **then** $z := -x$ **else** $z := x$



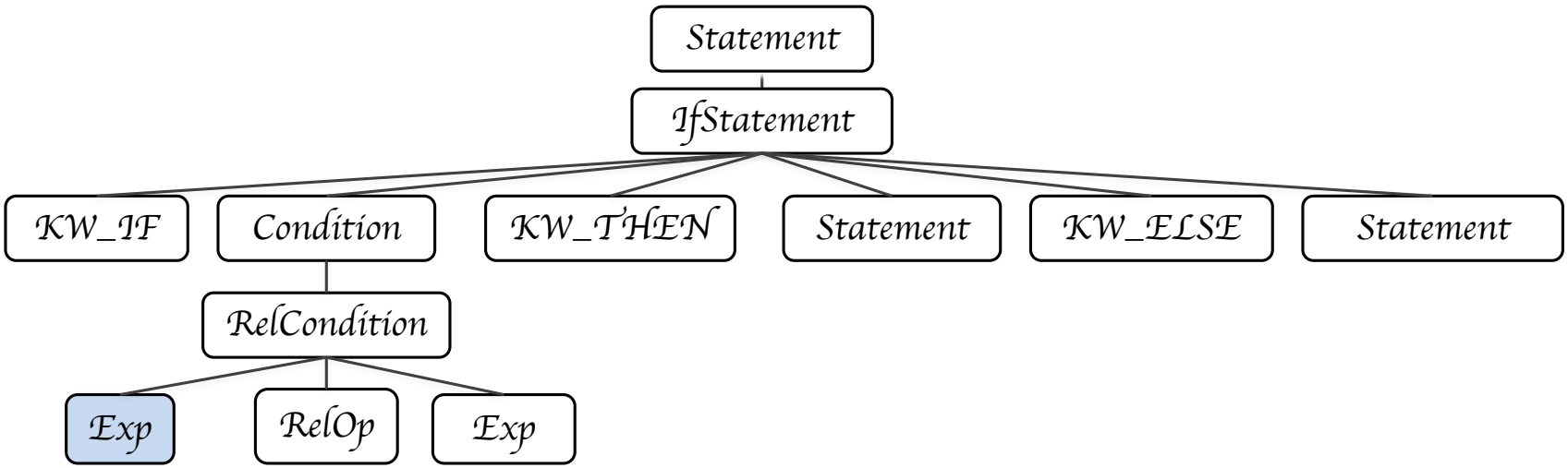
$$
\begin{aligned}
\textit{Statement} &\rightarrow \textit{Assignment} \mid \textit{CallStatement} \mid \textit{ReadStatement} \mid \textit{WriteStatement} \mid \\
&\quad \textit{WhileStatement} \mid \textit{IfStatement} \mid \textit{CompoundStatement} \\
&\ldots \\
\textit{IfStatement} &\rightarrow \textit{KW\_IF Condition KW\_THEN Statement KW\_ELSE Statement} \\
\textit{Condition} &\rightarrow \textit{RelCondition} \\
\textit{RelCondition} &\rightarrow \textit{Exp} \; [\; \textit{RelOp Exp} \;] \\
\textit{Exp} &\rightarrow \ldots
\end{aligned}
$$

if $x < 0$ **then** $z := -x$ **else** $z := x$



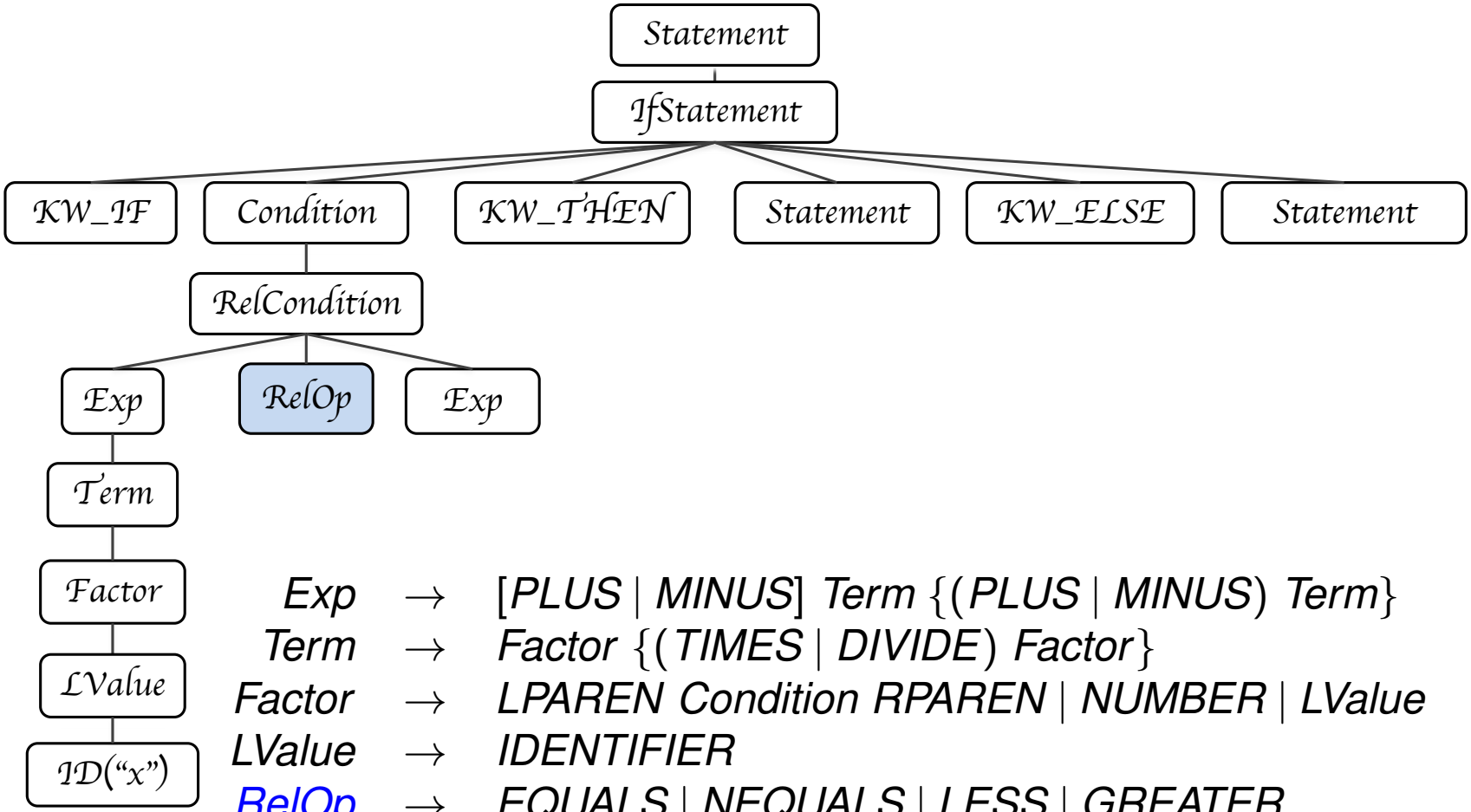
$$
\begin{aligned}
\mathit{Statement} \rightarrow\ & \mathit{Assignment} \mid \mathit{CallStatement} \mid \mathit{ReadStatement} \mid \mathit{WriteStatement} \mid \\
& \mathit{WhileStatement} \mid \mathit{IfStatement} \mid \mathit{CompoundStatement} \\
& \ldots \\
\mathit{IfStatement} \rightarrow\ & \mathit{KW\_IF}\ \mathit{Condition}\ \mathit{KW\_THEN}\ \mathit{Statement}\ \mathit{KW\_ELSE}\ \mathit{Statement} \\
\mathit{Condition} \rightarrow\ & \mathit{RelCondition} \\
\mathit{RelCondition} \rightarrow\ & \mathit{Exp}\ [\ \mathit{RelOp}\ \mathit{Exp}\ ] \\
\mathit{Exp} \rightarrow\ & \ldots
\end{aligned}
$$

if $x < 0$ **then** $z := -x$ **else** $z := x$



$$
\begin{aligned}
\underline{Exp} &\rightarrow [PLUS \mid MINUS]\ Term\ \{(PLUS \mid MINUS)\ Term\} \\
Term &\rightarrow Factor\ \{(TIMES \mid DIVIDE)\ Factor\} \\
Factor &\rightarrow LPAREN\ Condition\ RPAREN \mid NUMBER \mid LValue \\
LValue &\rightarrow IDENTIFIER \\
RelOp &\rightarrow EQUALS \mid NEQUALS \mid LESS \mid GREATER \\
&\quad \mid LEQUALS \mid GEQUALS
\end{aligned}
$$

if $x < 0$ **then** $z := -x$ **else** $z := x$



Exp → [*PLUS* | *MINUS*] *Term* {(*PLUS* | *MINUS*) *Term*}

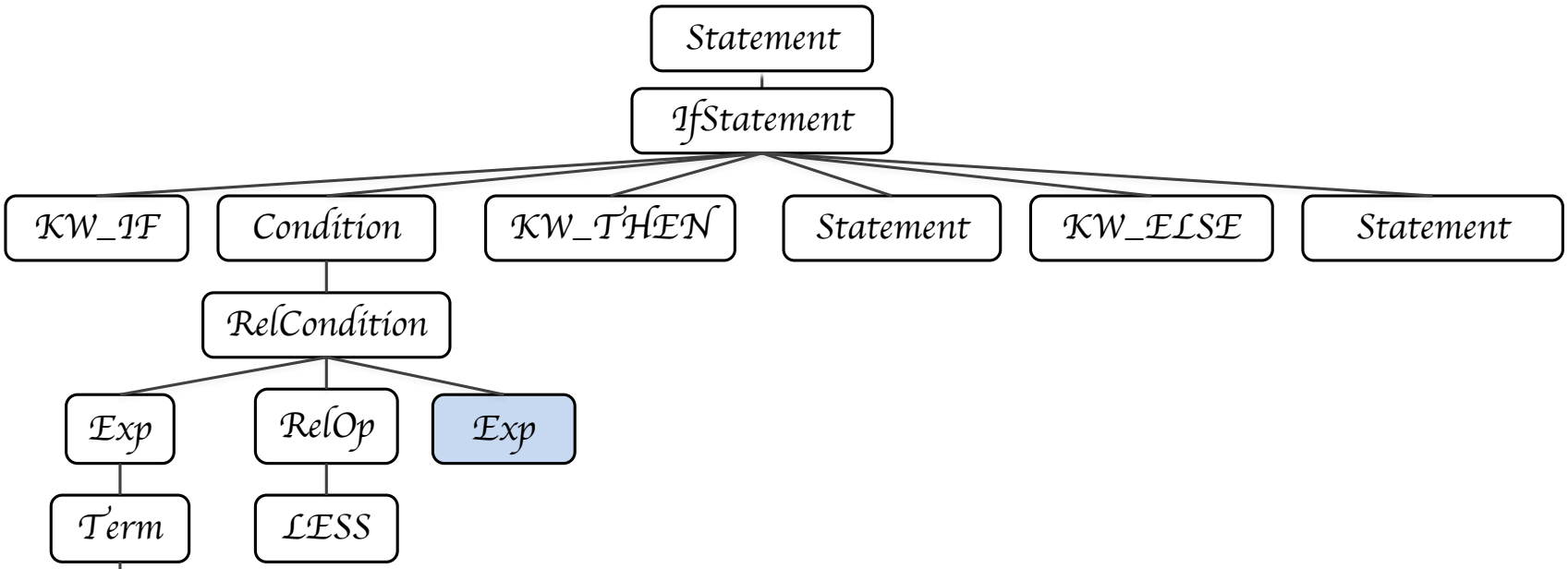
Term → *Factor* {(*TIMES* | *DIVIDE*) *Factor*}

Factor → *LPAREN Condition RPAREN* | *NUMBER* | *LValue*

LValue → *IDENTIFIER*

RelOp → *EQUALS* | *NEQUALS* | *LESS* | *GREATER*
| *LEQUALS* | *GEQUALS*

if $x < 0$ **then** $z := -x$ **else** $z := x$



Exp → [*PLUS* | *MINUS*] *Term* {(*PLUS* | *MINUS*) *Term*}

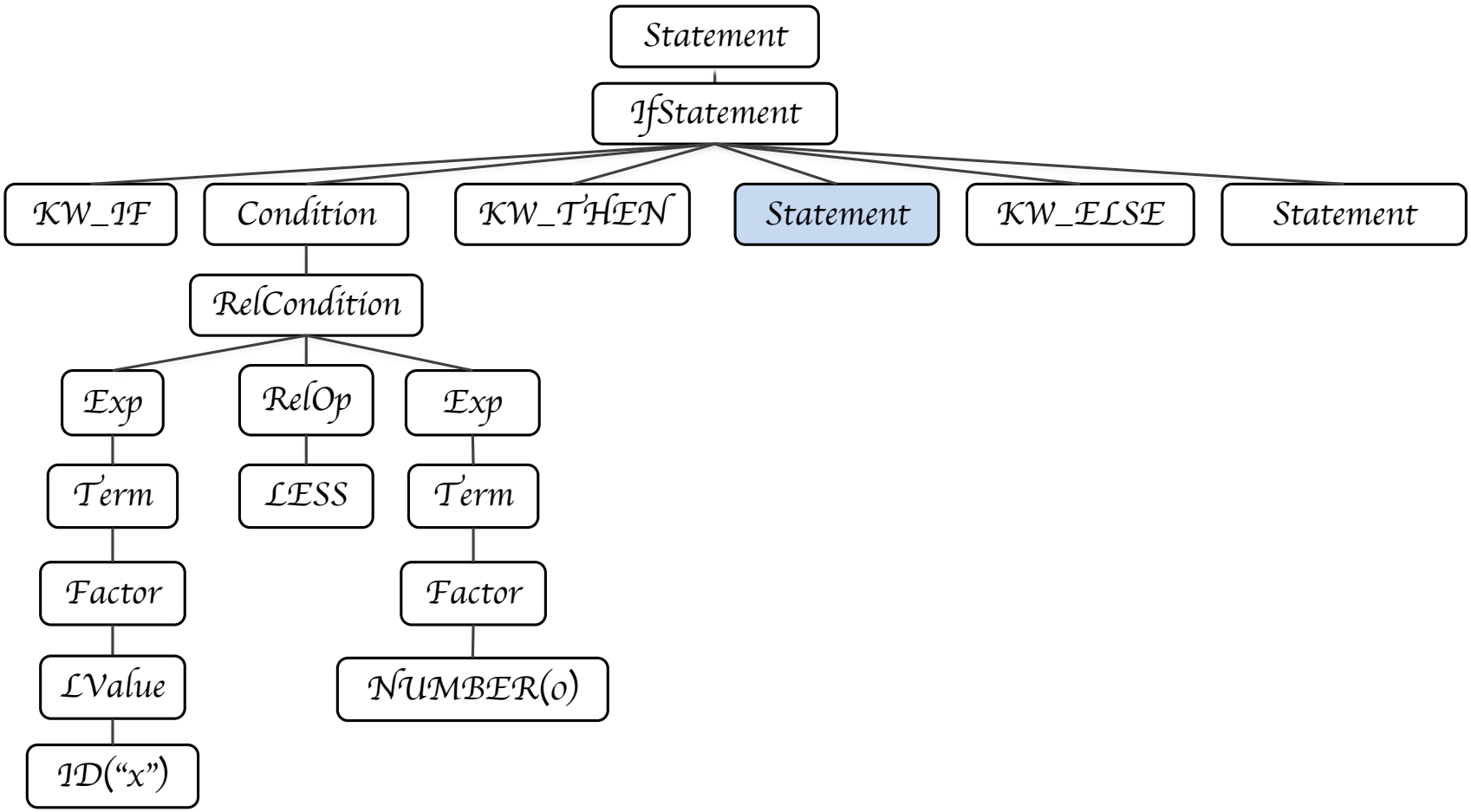
Term → *Factor* {(*TIMES* | *DIVIDE*) *Factor*}

Factor → *LPAREN Condition RPAREN* | *NUMBER* | *LValue*

LValue → *IDENTIFIER*

RelOp → *EQUALS* | *NEQUALS* | *LESS* | *GREATER* | *LEQUALS* | *GEQUALS*

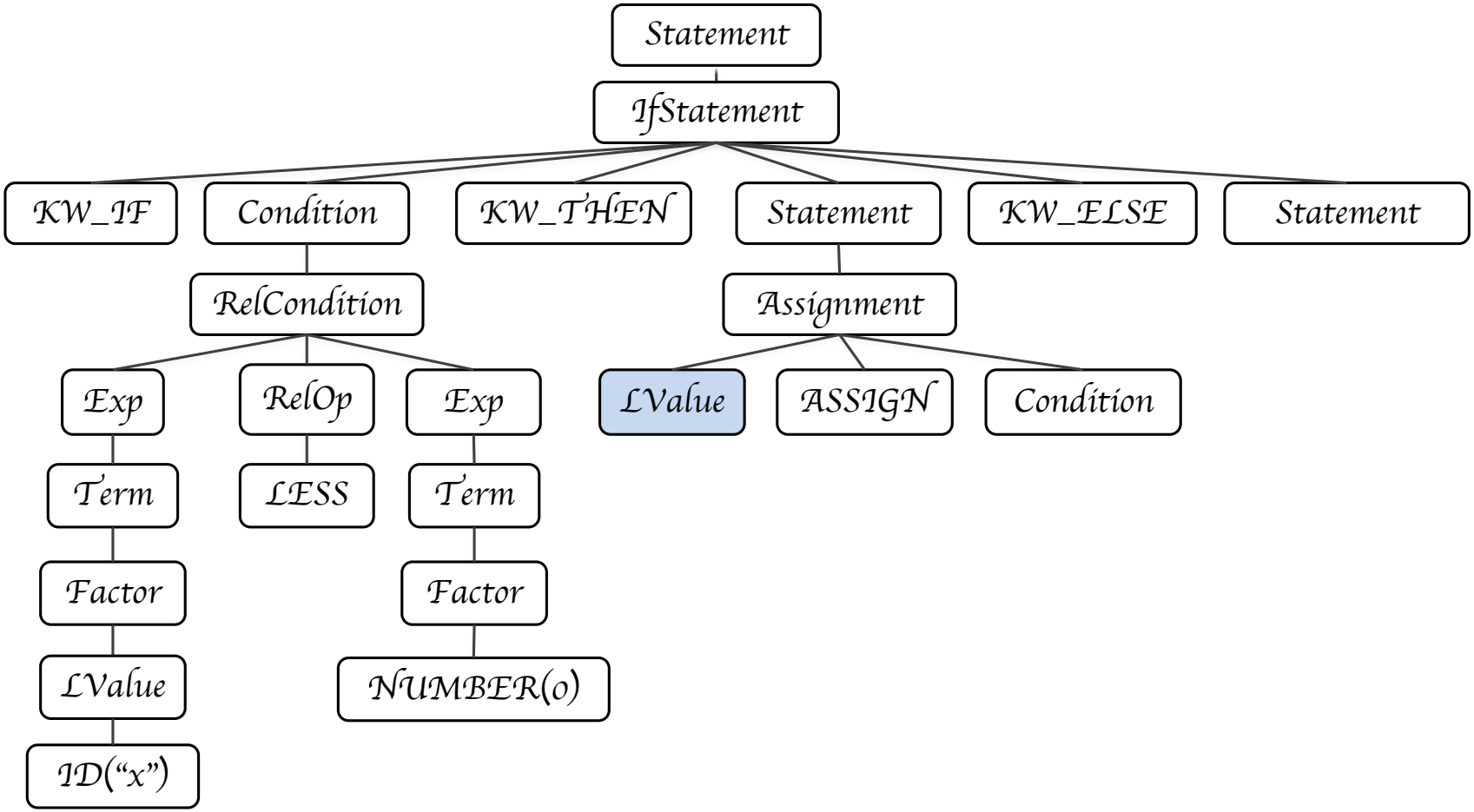
if $x < 0$ **then** $z := -x$ **else** $z := x$



Statement → *Assignment* | *CallStatement* | *ReadStatement* | *WriteStatement*| *WhileStatement* | *IfStatement* | *CompoundStatement*

Assignment → *LValue ASSIGN Condition*

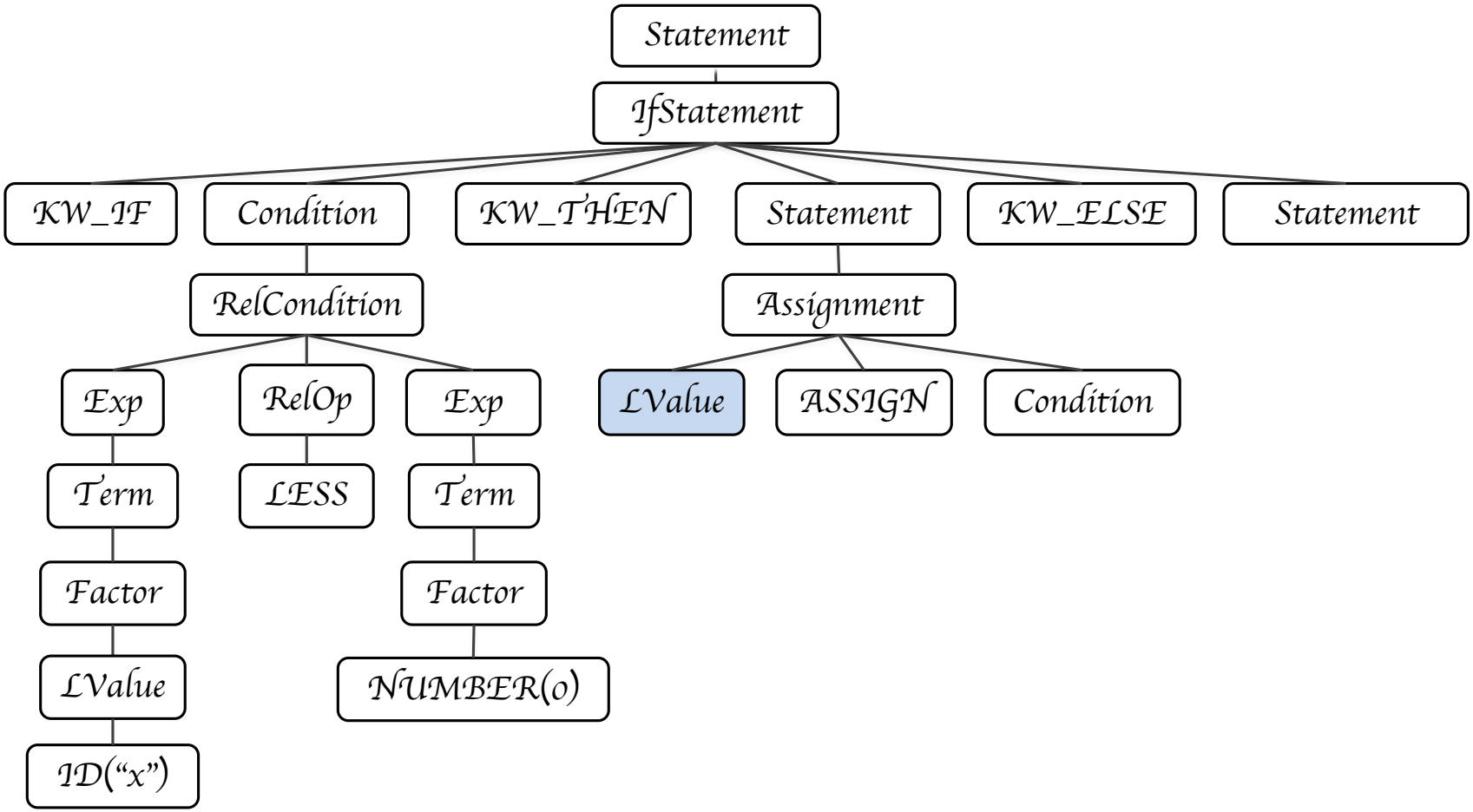
if $x < 0$ **then** $z := -x$ **else** $z := x$



Statement → *Assignment* | *CallStatement* | *ReadStatement* | *WriteStatement*|
WhileStatement | *IfStatement* | *CompoundStatement*

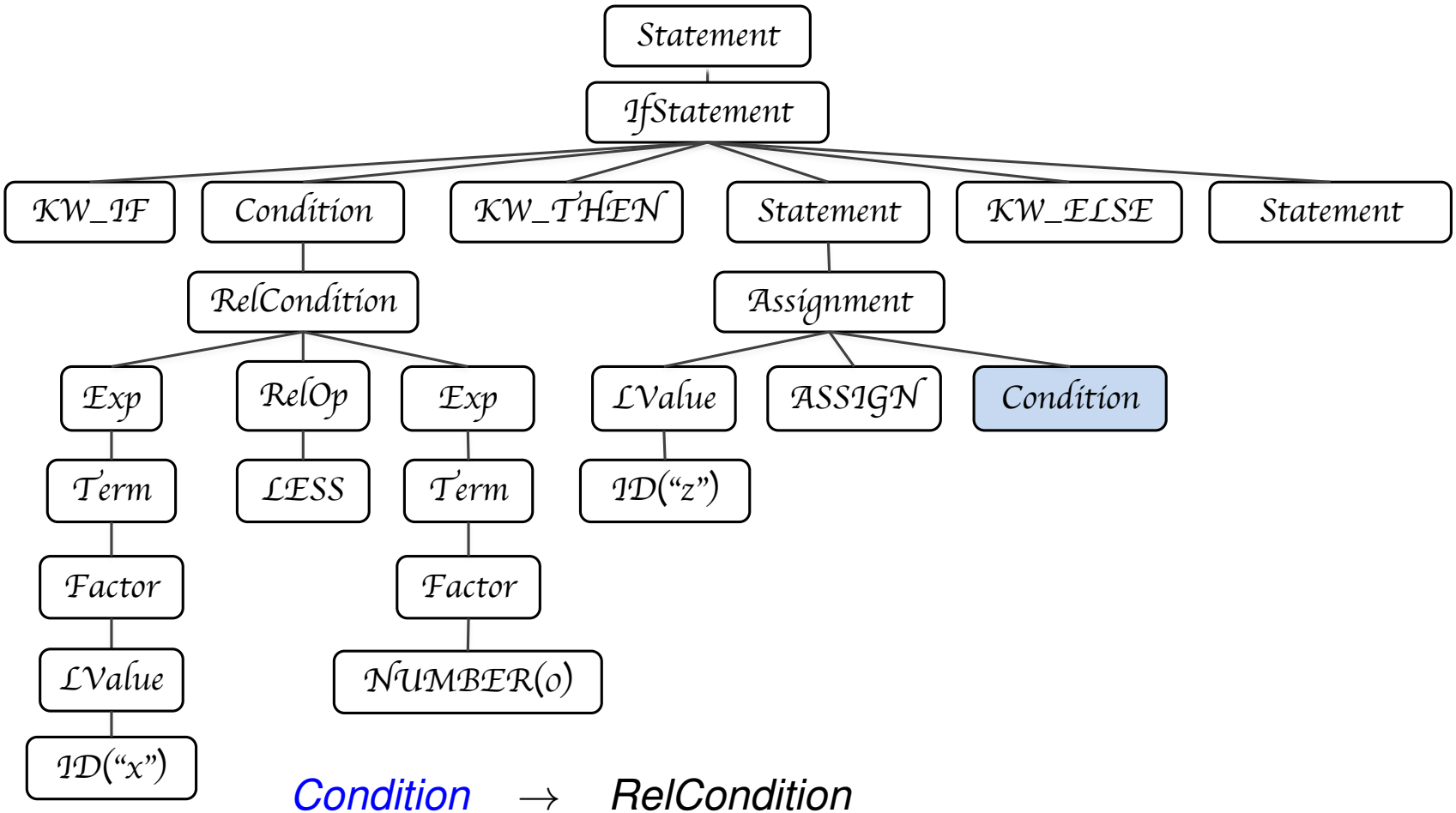
Assignment → *LValue ASSIGN Condition*

if $x < 0$ **then** $z := -x$ **else** $z := x$



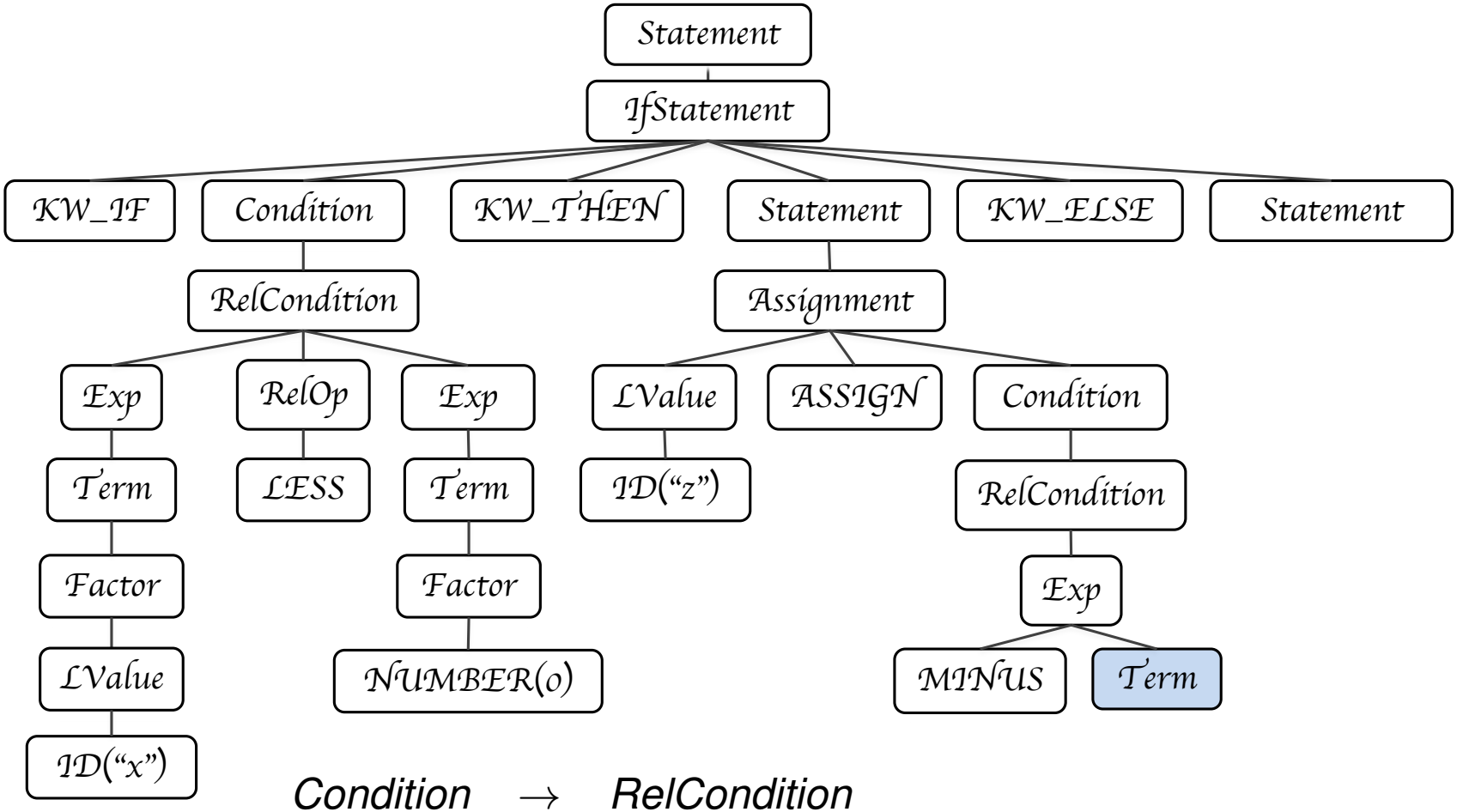
LValue $\rightarrow$ *IDENTIFIER*

if $x < 0$ **then** $z := \underline{-x}$ **else** $z := x$



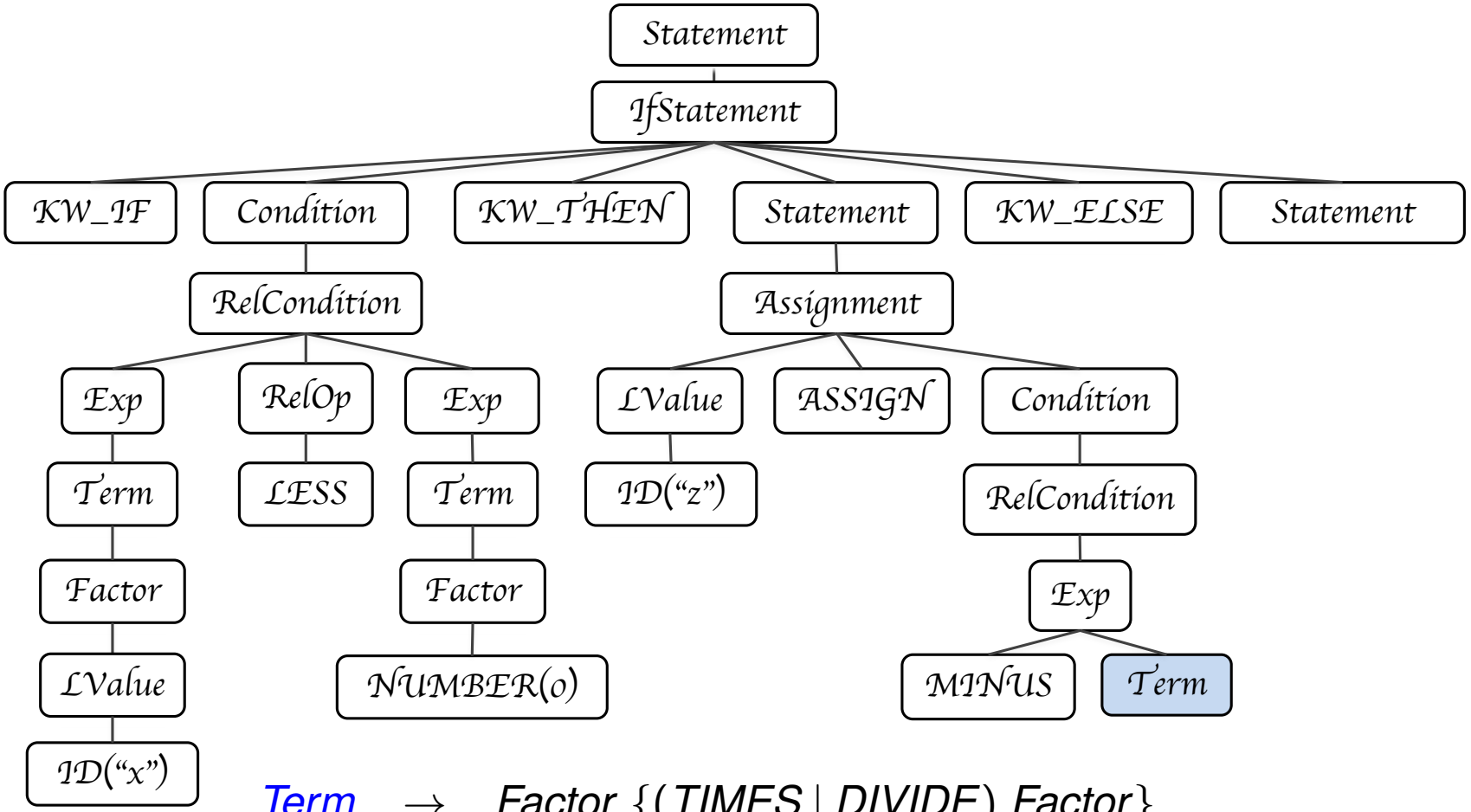
$$\begin{aligned}
\underline{\textit{Condition}} &\rightarrow \textit{RelCondition} \\
\textit{RelCondition} &\rightarrow \textit{Exp}\ [\ \textit{RelOp}\ \textit{Exp}\ ] \\
\textit{Exp} &\rightarrow [\textit{PLUS} \mid \textit{MINUS}]\ \textit{Term}\ \{(\textit{PLUS} \mid \textit{MINUS})\ \textit{Term}\} \\
\textit{Term} &\rightarrow \ldots
\end{aligned}$$

if $x < 0$ **then** $z := -x$ **else** $z := x$



$$
\begin{aligned}
\textit{Condition} &\rightarrow \textit{RelCondition} \\
\textit{RelCondition} &\rightarrow \textit{Exp}\ [\ \textit{RelOp}\ \textit{Exp}\ ] \\
\textit{Exp} &\rightarrow [\textit{PLUS} \mid \textit{MINUS}]\ \textit{Term}\ \{(\textit{PLUS} \mid \textit{MINUS})\ \textit{Term}\} \\
\textit{Term} &\rightarrow \ldots
\end{aligned}
$$

if $x < 0$ **then** $z := -x$ **else** $z := x$

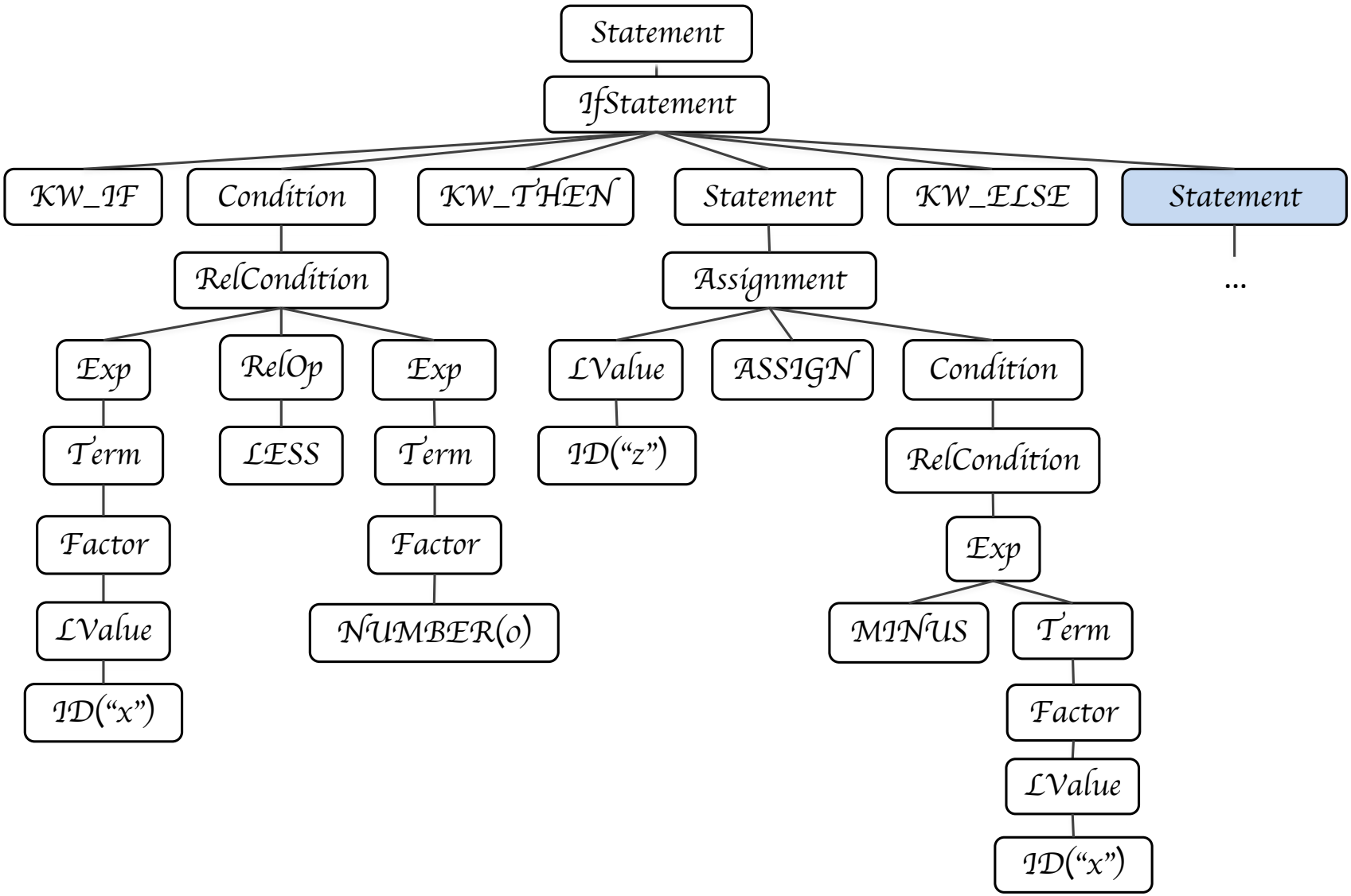


Term → *Factor* {(*TIMES* | *DIVIDE*) *Factor*}

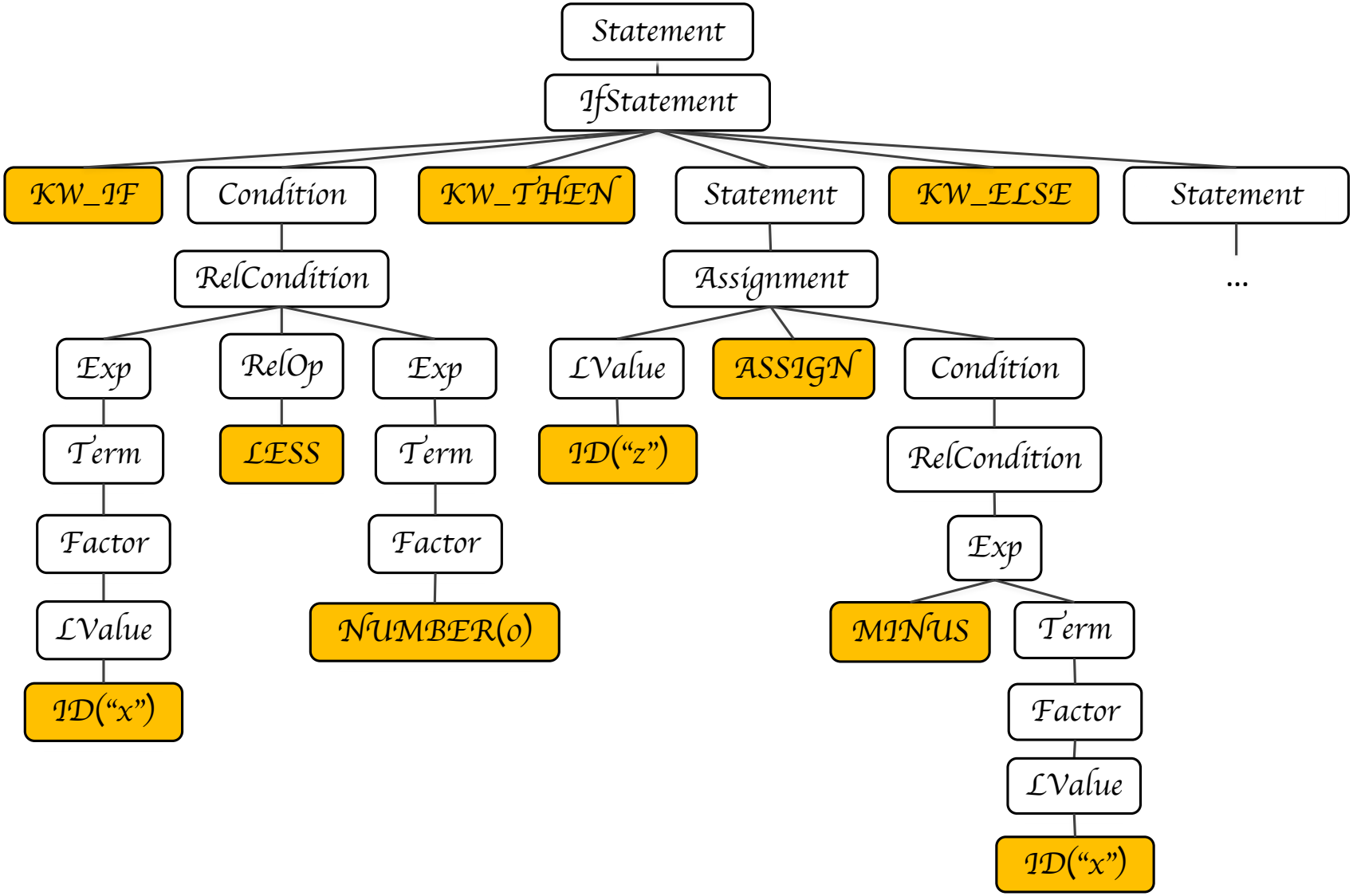
Factor → *LPAREN Condition RPAREN* | *NUMBER* | *LValue*

LValue → *IDENTIFIER*

if $x < 0$ **then** $z := -x$ **else** $z := x$

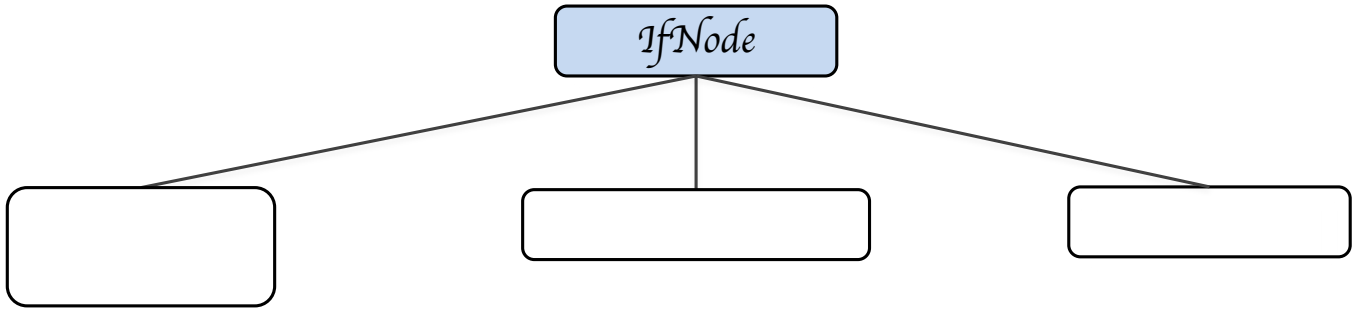


if $x < 0$ **then** $z := -x$ **else** $z := x$



Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$

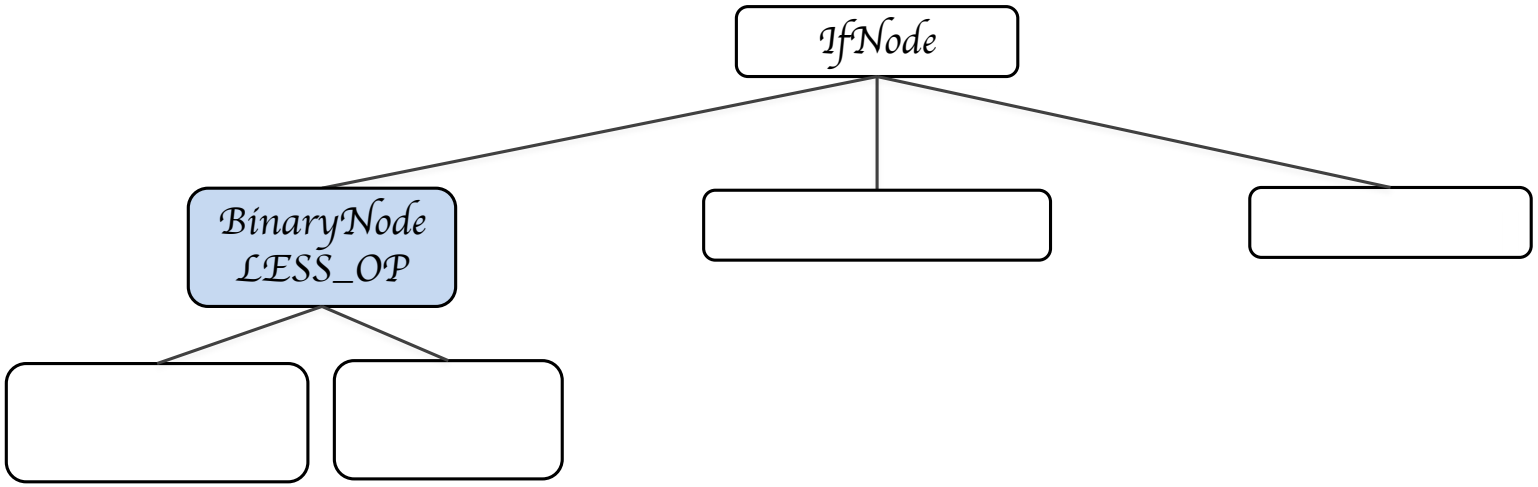


StatementNode(Location loc) with subclasses
- AssignmentNode(ExpNode lvalue, ExpNode e)
- IfNode(ExpNode cond, StatementNode s1, StatementNode s2)
- ...

ExpNode(Location loc, Type t) with subclasses
- ConstNode(int value)
- IdentifierNode(String id)
- BinaryNode(Operator op, ExpNode left, ExpNode right)

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$

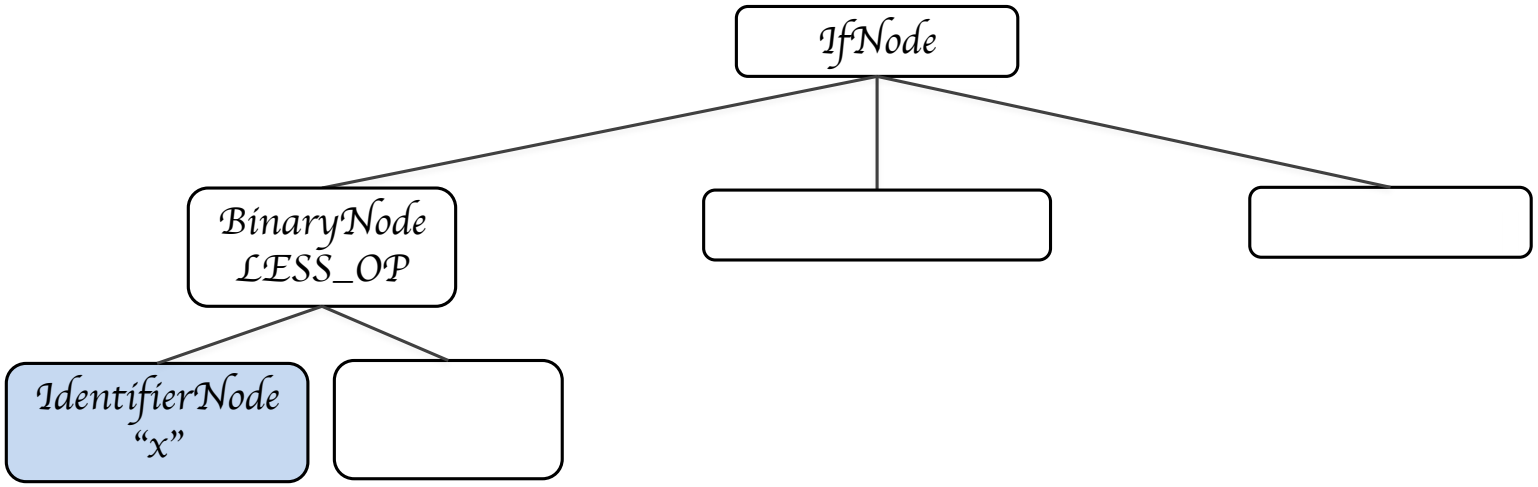


StatementNode(Location loc) with subclasses
- AssignmentNode(ExpNode lvalue, ExpNode e)
- IfNode(ExpNode cond, StatementNode s1, StatementNode s2)
- ...

ExpNode(Location loc, Type t) with subclasses
- ConstNode(int value)
- IdentifierNode(String id)
- BinaryNode(Operator op, ExpNode left, ExpNode right)

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$

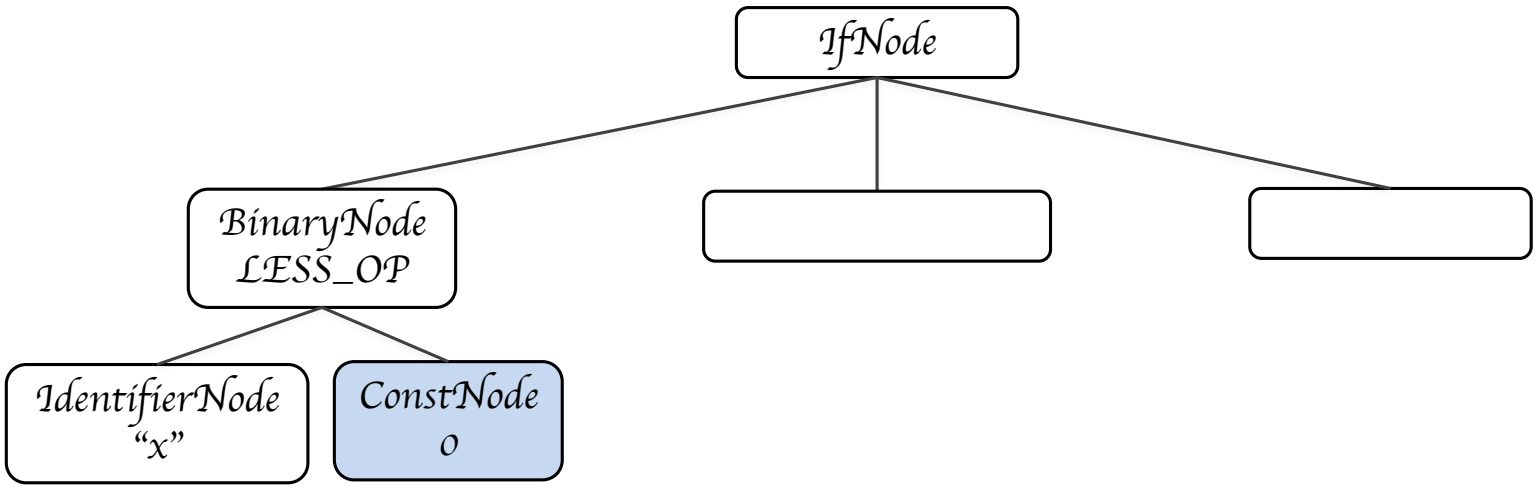


StatementNode(Location loc) with subclasses
- AssignmentNode(ExpNode lvalue, ExpNode e)
- IfNode(ExpNode cond, StatementNode s1, StatementNode s2)
- ...

ExpNode(Location loc, Type t) with subclasses
- ConstNode(int value)
- IdentifierNode(String id)
- BinaryNode(Operator op, ExpNode left, ExpNode right)

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$

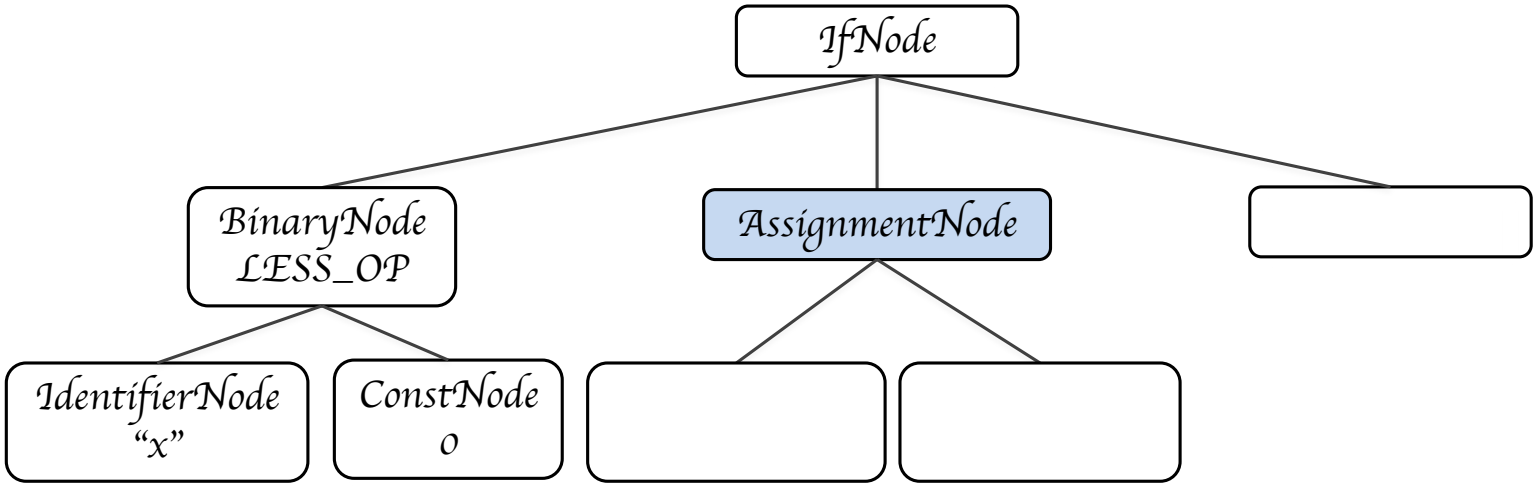


StatementNode(Location loc) with subclasses
- AssignmentNode(ExpNode lvalue, ExpNode e)
- IfNode(ExpNode cond, StatementNode s1, StatementNode s2)
- ...

ExpNode(Location loc, Type t) with subclasses
- ConstNode(int value)
- IdentifierNode(String id)
- BinaryNode(Operator op, ExpNode left, ExpNode right)

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$



StatementNode(Location loc) with subclasses

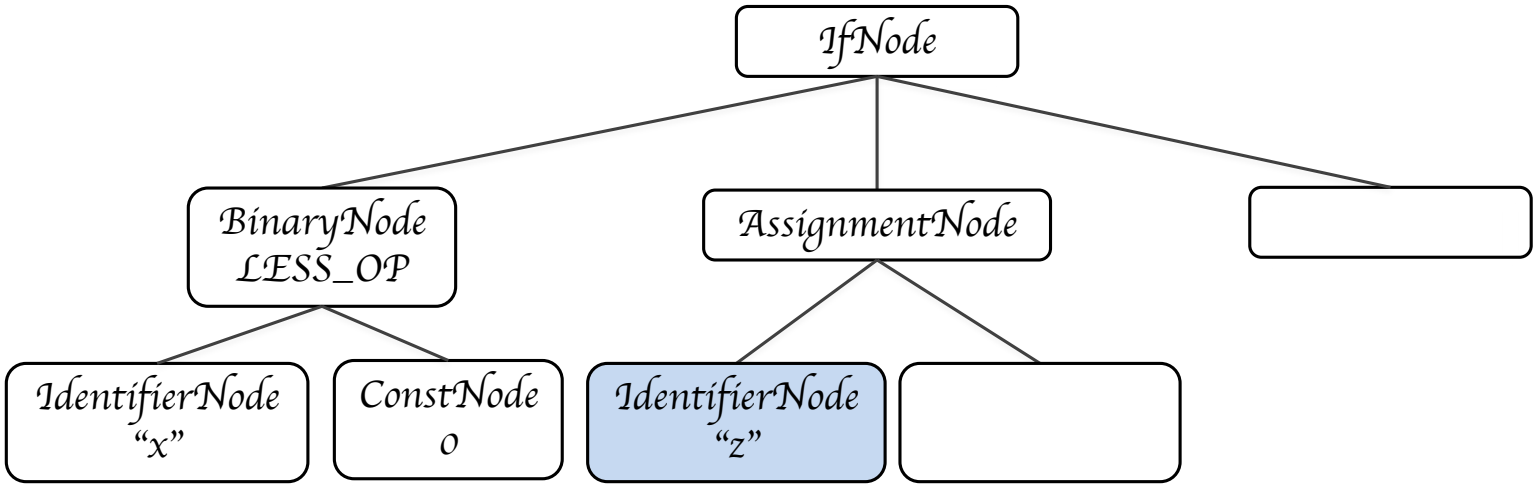
- AssignmentNode(ExpNode lvalue, ExpNode e)
- IfNode(ExpNode cond, StatementNode s1, StatementNode s2)
- ...

ExpNode(Location loc, Type t) with subclasses

- ConstNode(int value)
- IdentifierNode(String id)
- UnaryNode(Operator op, ExpNode arg)

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$



StatementNode(Location loc) with subclasses

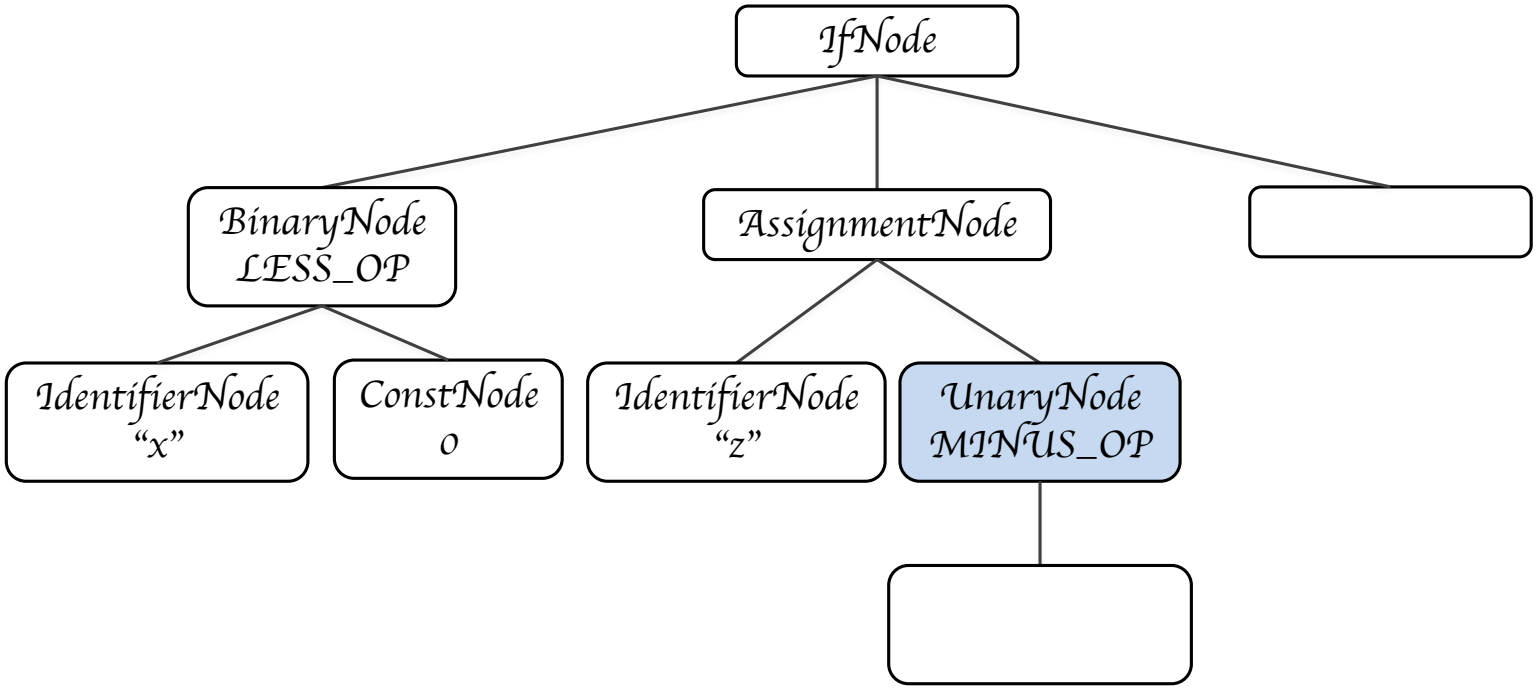
- AssignmentNode(ExpNode lvalue, ExpNode e)
- IfNode(ExpNode cond, StatementNode s1, StatementNode s2)
- ...

ExpNode(Location loc, Type t) with subclasses

- ConstNode(int value)
- IdentifierNode(String id)
- UnaryNode(Operator op, ExpNode arg)

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$



StatementNode(Location loc) with subclasses

AssignmentNode(ExpNode lvalue, ExpNode e)

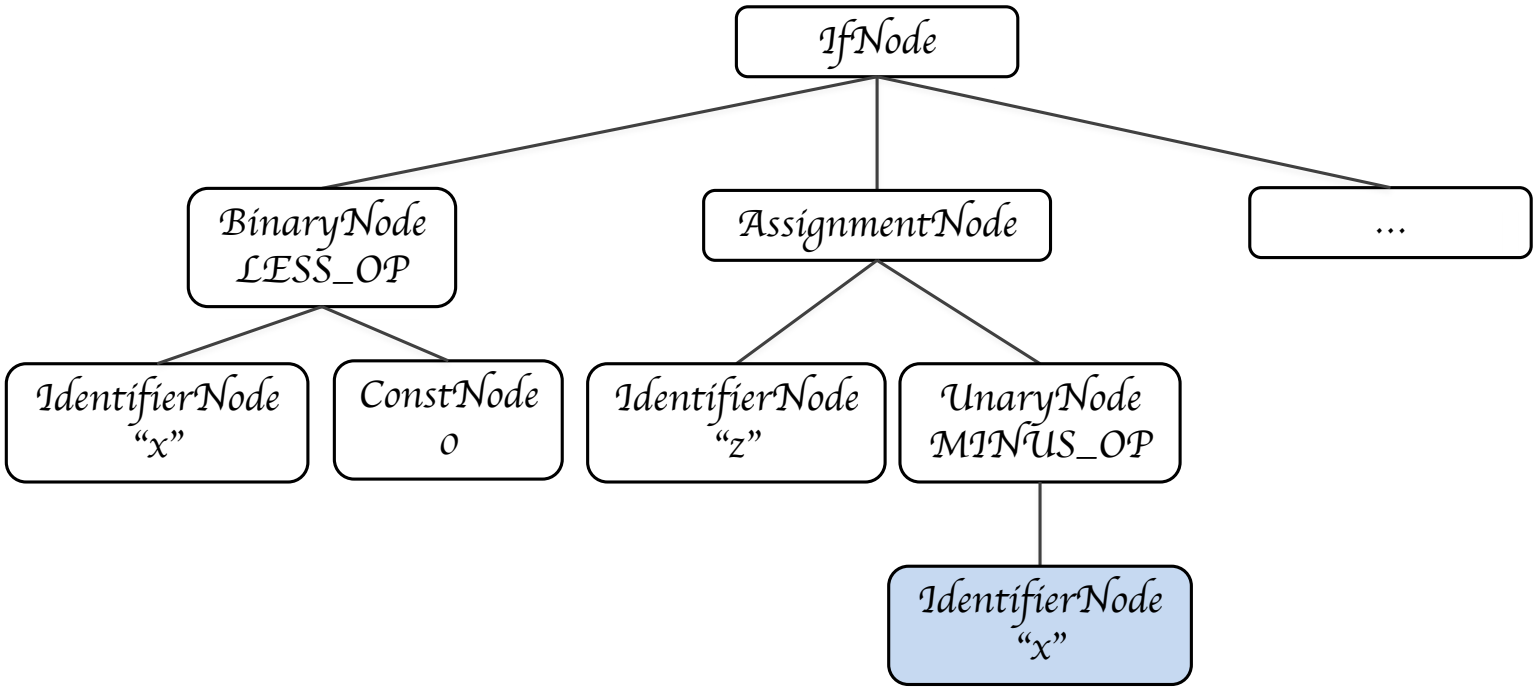
...

ExpNode(Location loc, Type t) with subclasses

UnaryNode(Operator op, ExpNode arg)

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses

ConstNode(int value)

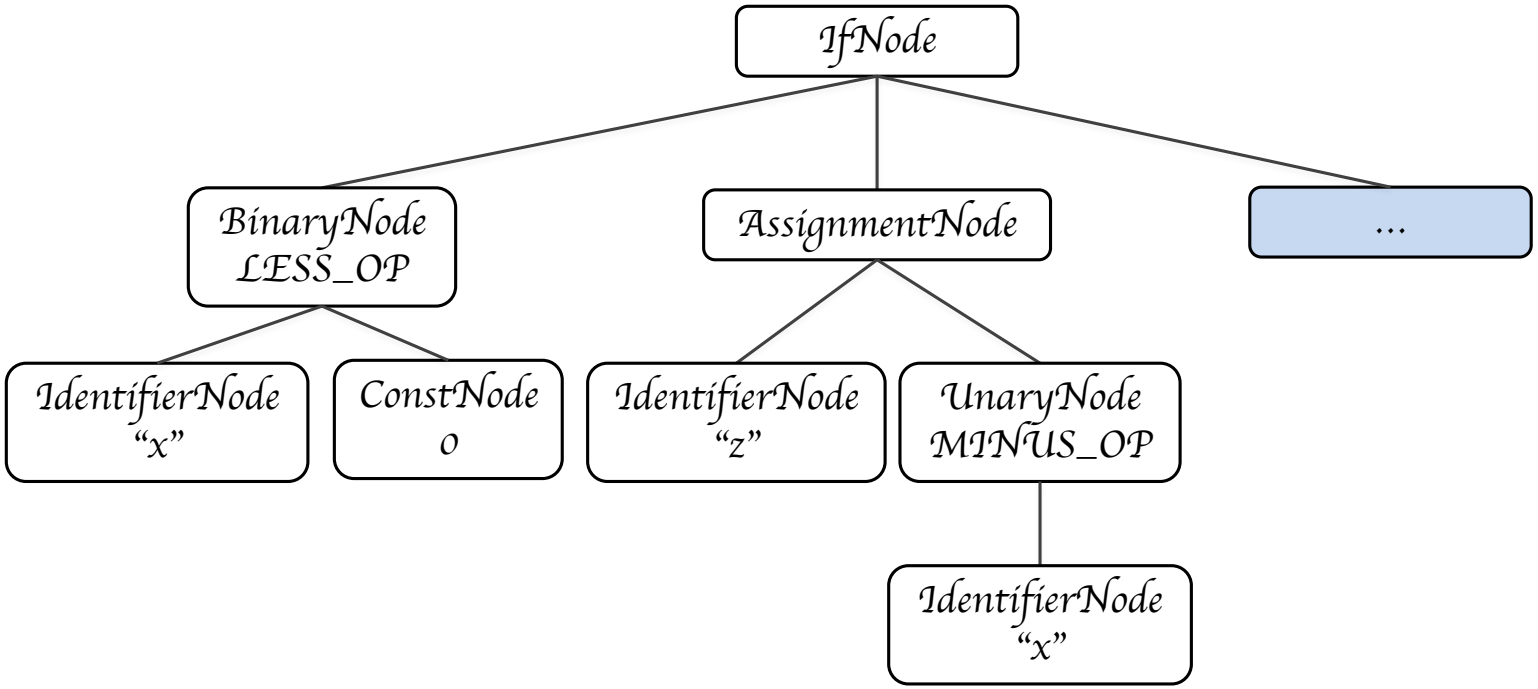
IdentifierNode(String id)

UnaryNode(Operator op, ExpNode arg)

...

Example: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$



StatementNode(Location loc) with subclasses

AssignmentNode(ExpNode lvalue, ExpNode e)

IfNode(ExpNode cond, StatementNode s1, StatementNode s2)

...

Type Checking the AST

IdentifierNode is used during parsing to represent a reference to either a symbolic constant or a variable.

As part of the static semantics (type checking) phase it will be transformed to either a ConstNode or a VariableNode.

ExpNode(Location loc, Type t) with subclasses
- ConstNode(int value)
- IdentifierNode(String id)
- VariableNode(SymEntry.VarEntry entry)
- ...

Type Checking the AST

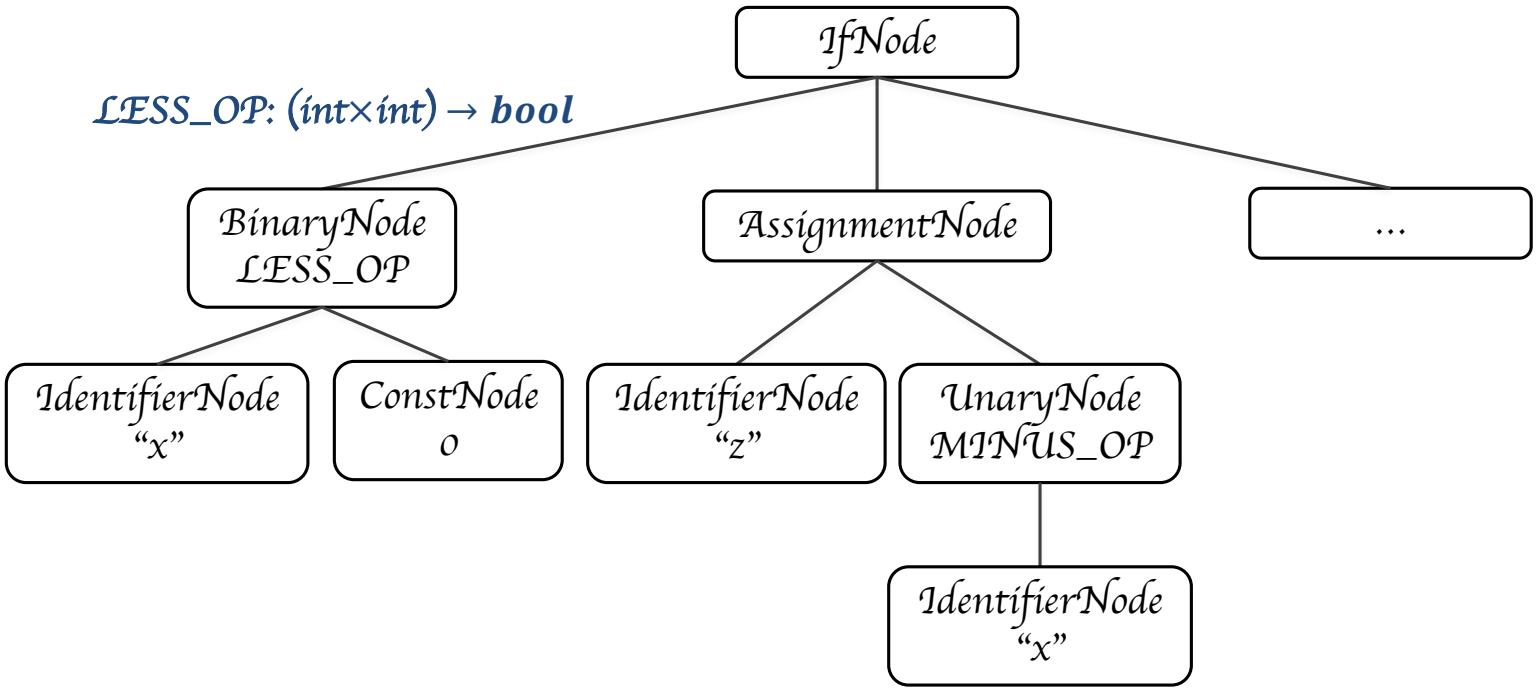
A number of other node types are only introduced in the static semantics phase:

- a DereferenceNode represents a dereference of a variable address (left value) to access its (right) value;
- an expression of a type, T, can be narrowed to a subrange of T (NarrowSubrangeNode).
- an expression of a subrange type can be widened to the base type of the subrange (WidenSubrangeNode);

```
ExpNode(Location loc, Type t) with subclasses
        DereferenceNode(ExpNode leftValue)
        NarrowSubrangeNode(ExpNode e)
        WidenSubrangeNode(ExpNode e)
        ...
```

Example:Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses

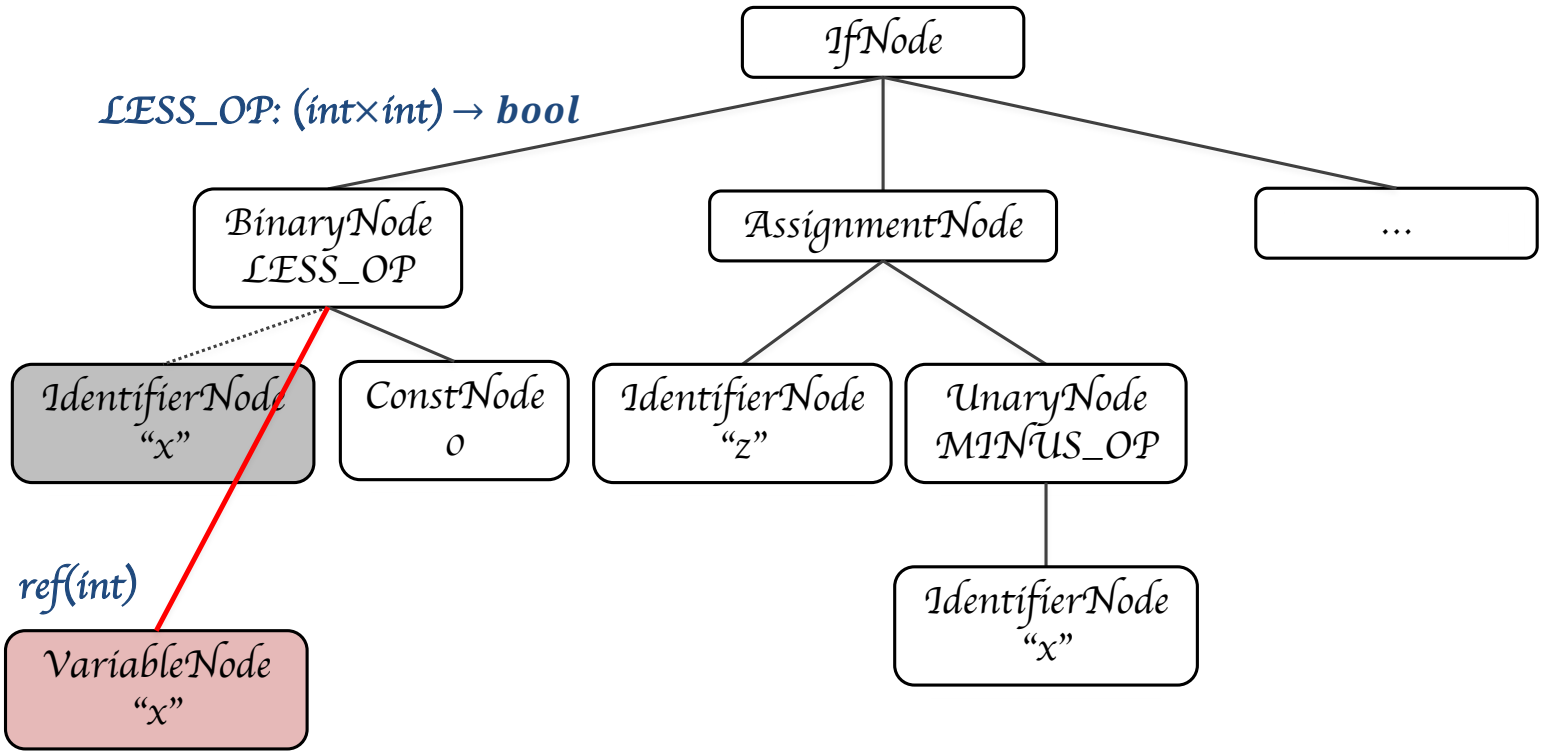
VariableNode(SymEntry.VarEntry entry)

DereferenceNode(ExpNode leftValue)

...

Example:Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses

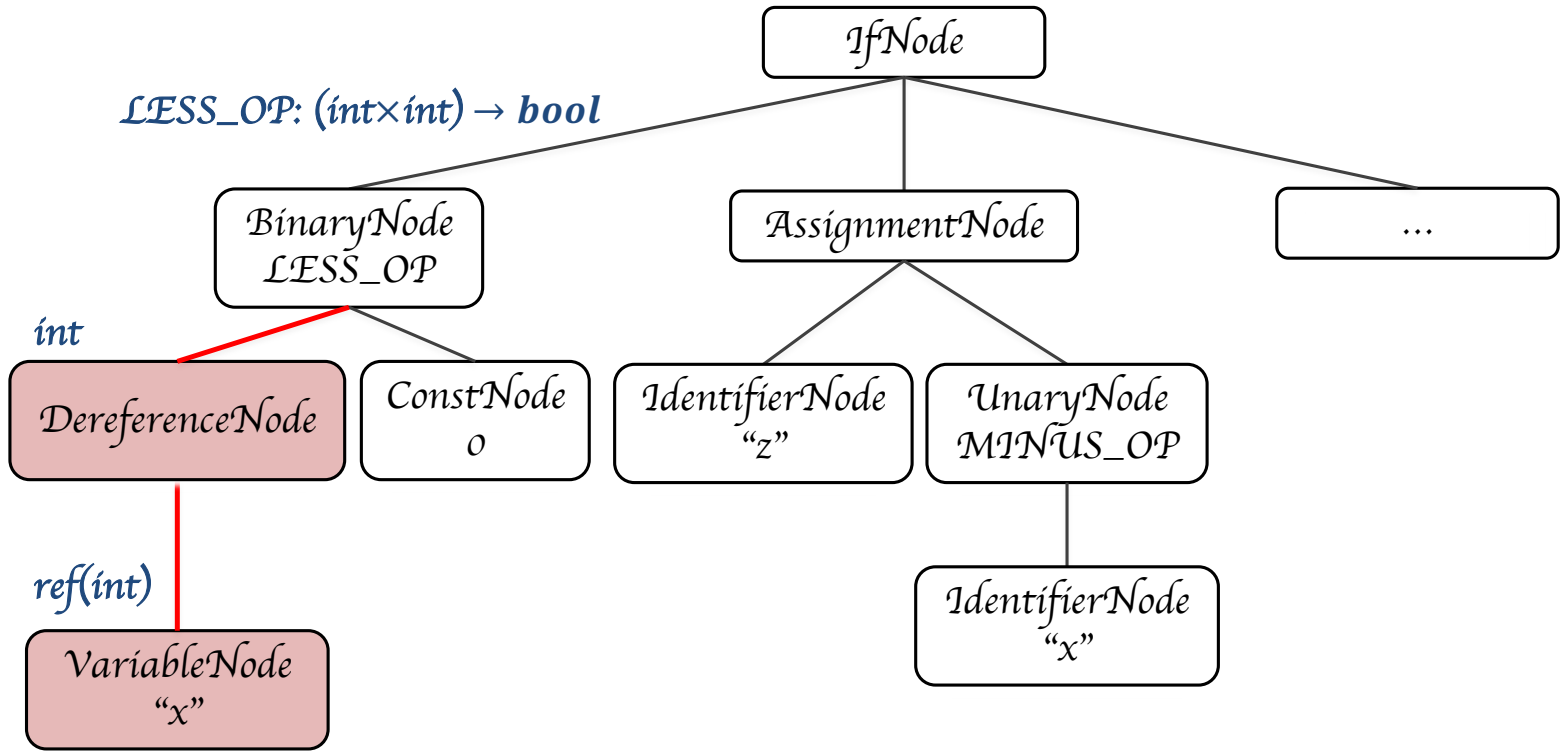
VariableNode(SymEntry.VarEntry entry)

DereferenceNode(ExpNode leftValue)

...

Example:Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses

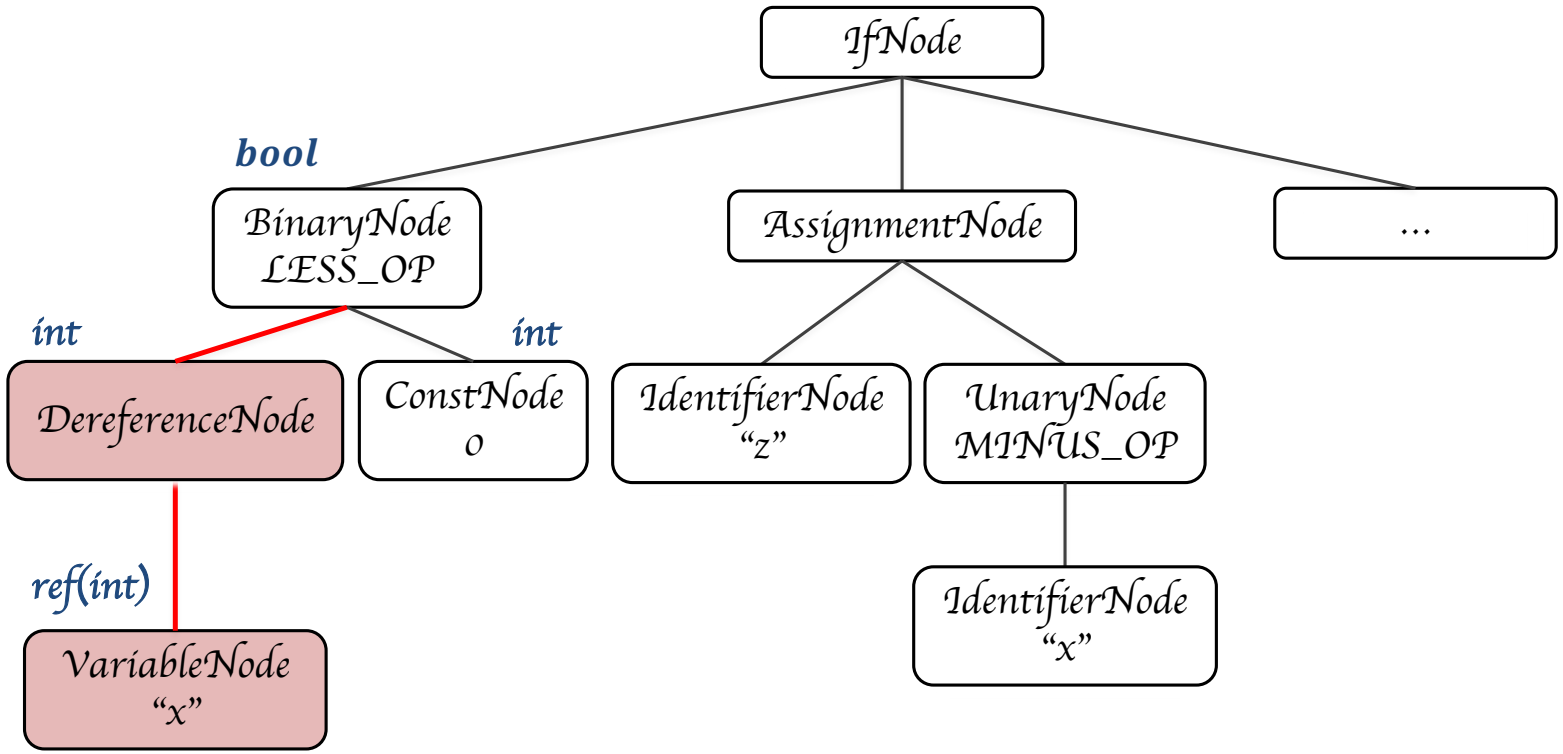
VariableNode(SymEntry.VarEntry entry)

DereferenceNode(ExpNode leftValue)

...

Example:Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses

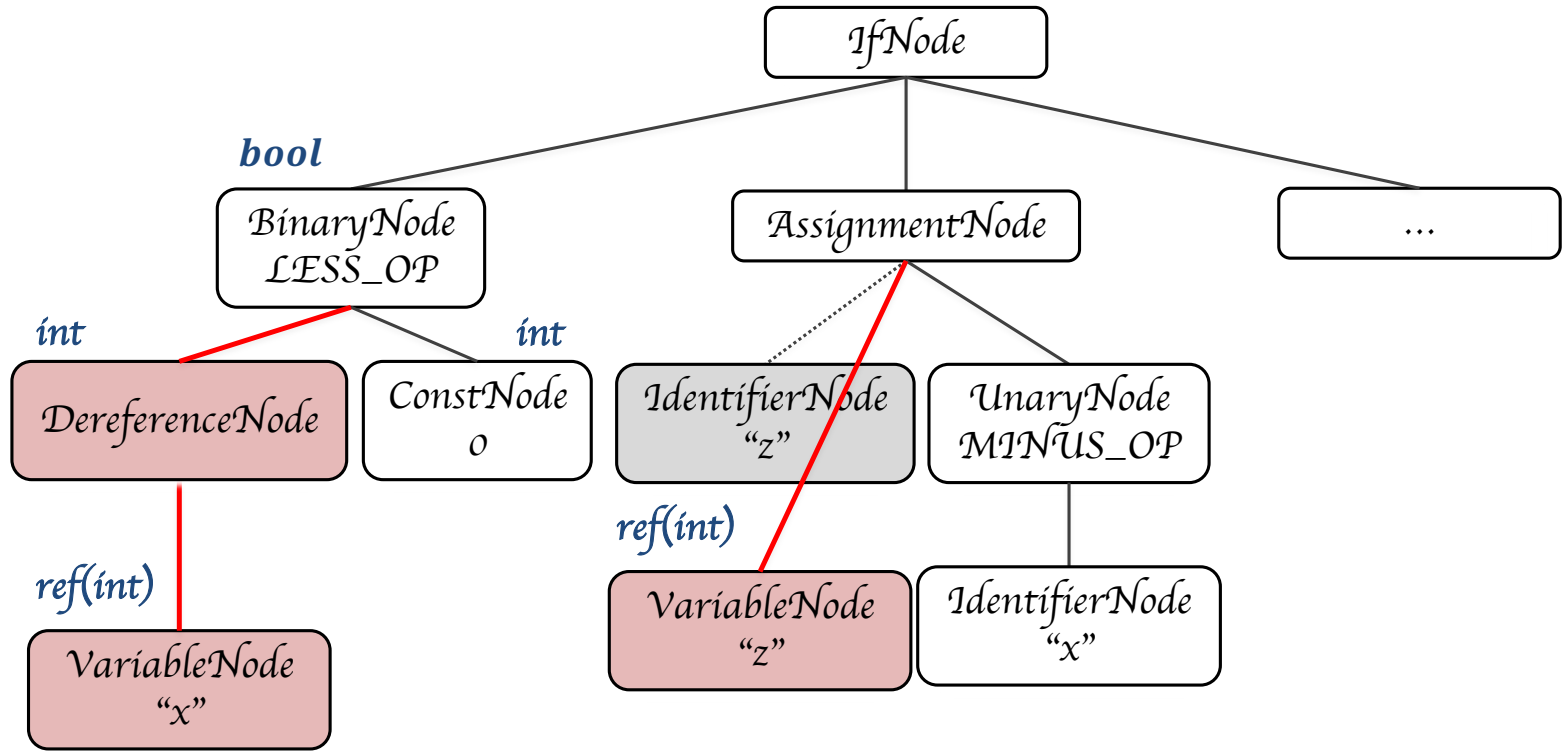
VariableNode(SymEntry.VarEntry entry)

DereferenceNode(ExpNode leftValue)

...

Example:Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses

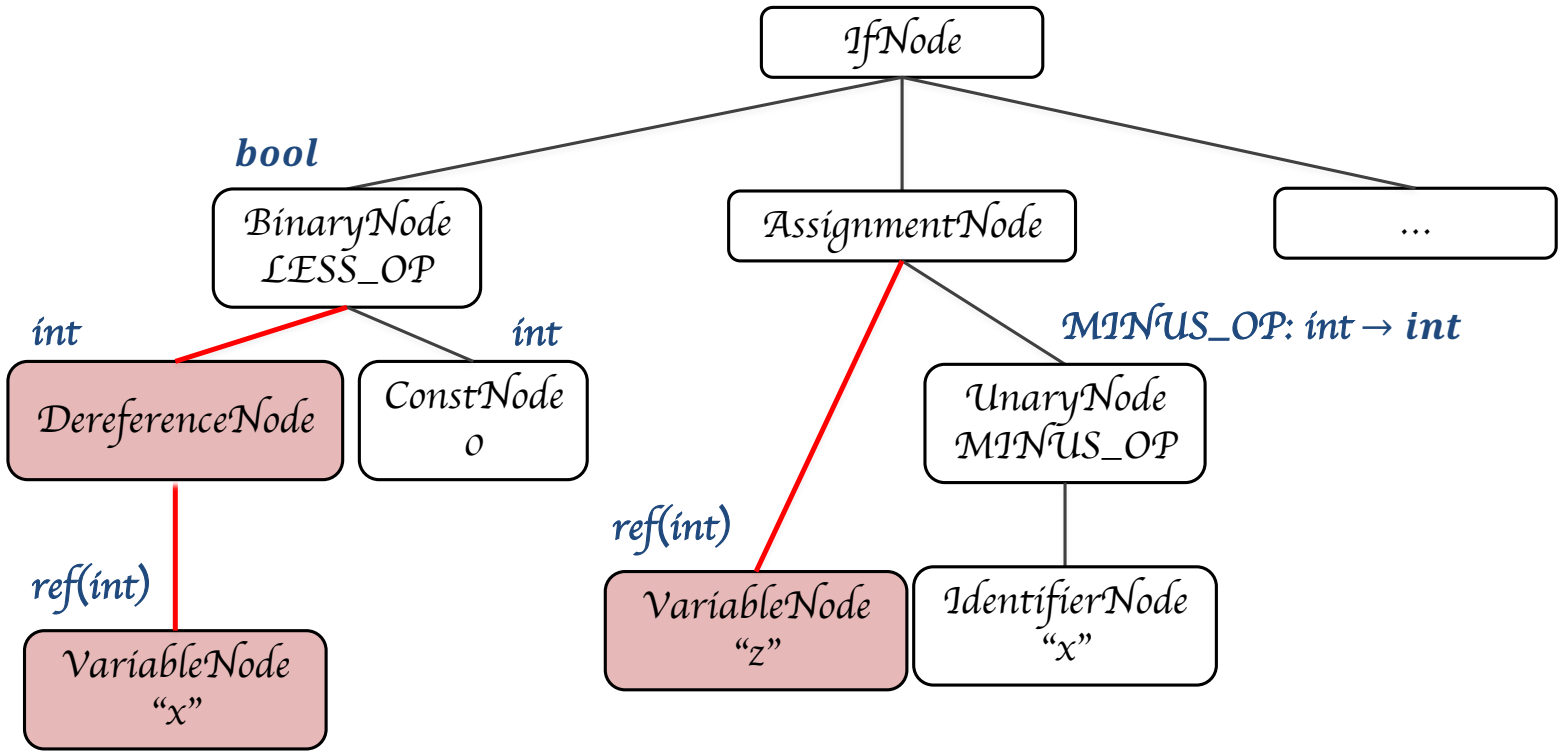
VariableNode(SymEntry.VarEntry entry)

DereferenceNode(ExpNode leftValue)

...

Example:Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses

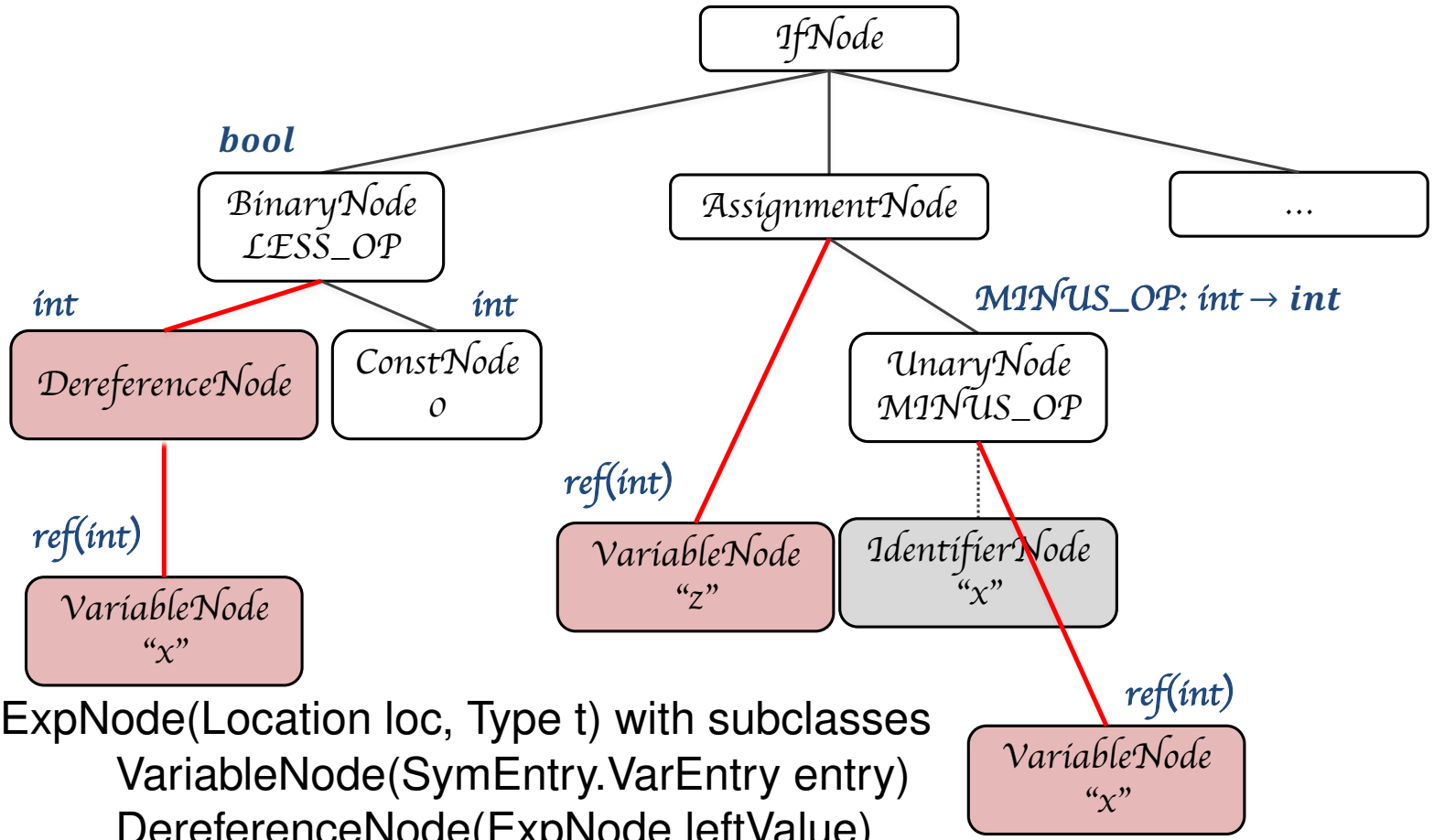
VariableNode(SymEntry.VarEntry entry)

DereferenceNode(ExpNode leftValue)

...

Example:Type Checking the AST

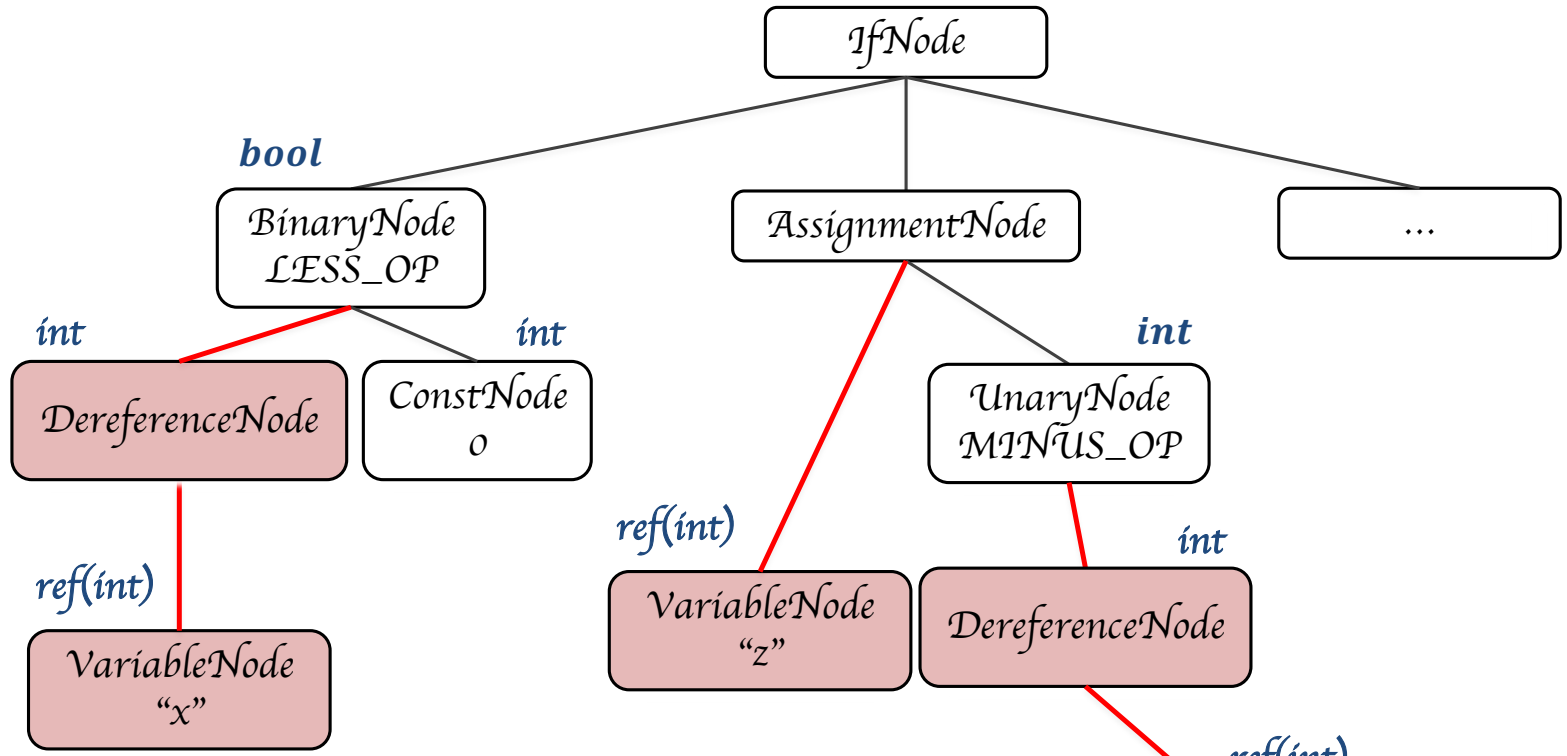
if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses
VariableNode(SymEntry.VarEntry entry)
DereferenceNode(ExpNode leftValue)
...

Example:Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$



ExpNode(Location loc, Type t) with subclasses
VariableNode(SymEntry.VarEntry entry)
DereferenceNode(ExpNode leftValue)
...

Example summary

$$\textbf{if } x < 0 \textbf{ then } z := -x \textbf{ else } z := x$$

- Lexical Analysis of the sequence of program characters produced a sequence of lexical tokens.
- Syntax Analysis of the sequence of lexical tokens was used to produce:
  - A concrete syntax tree – not produced by the compiler!
  - An abstract syntax tree (AST)
- Static Analysis was used to:
  - resolve references to identifiers;
  - type check the abstract syntax tree; and
  - update the abstract syntax tree with type coercions.

Lexical Analysis

Input (sequence of characters):

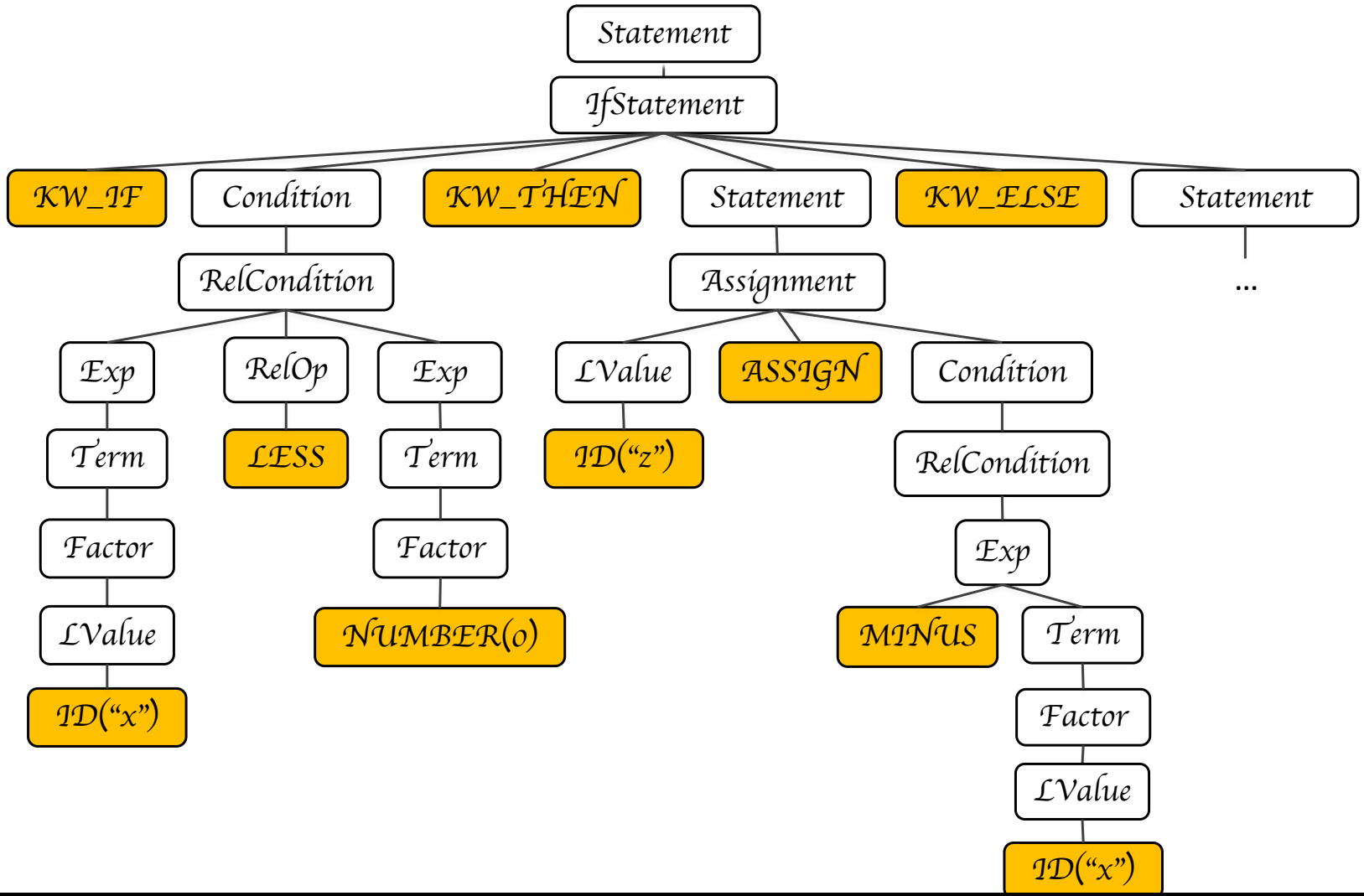
$$\textbf{if } x < 0 \textbf{ then } z := -x \textbf{ else } z := x$$

Output (sequence of **lexical tokens**):

KW_IF, *IDENTIFIER*(“x”), *LESS*, *NUMBER*(0), *KW_THEN*, *IDENTIFIER*(“z”), *ASSIGN*, *MINUS*, *IDENTIFIER*(“x”), *KW_ELSE*, *IDENTIFIER*(“z”), *ASSIGN*, *IDENTIFIER*(“x”)

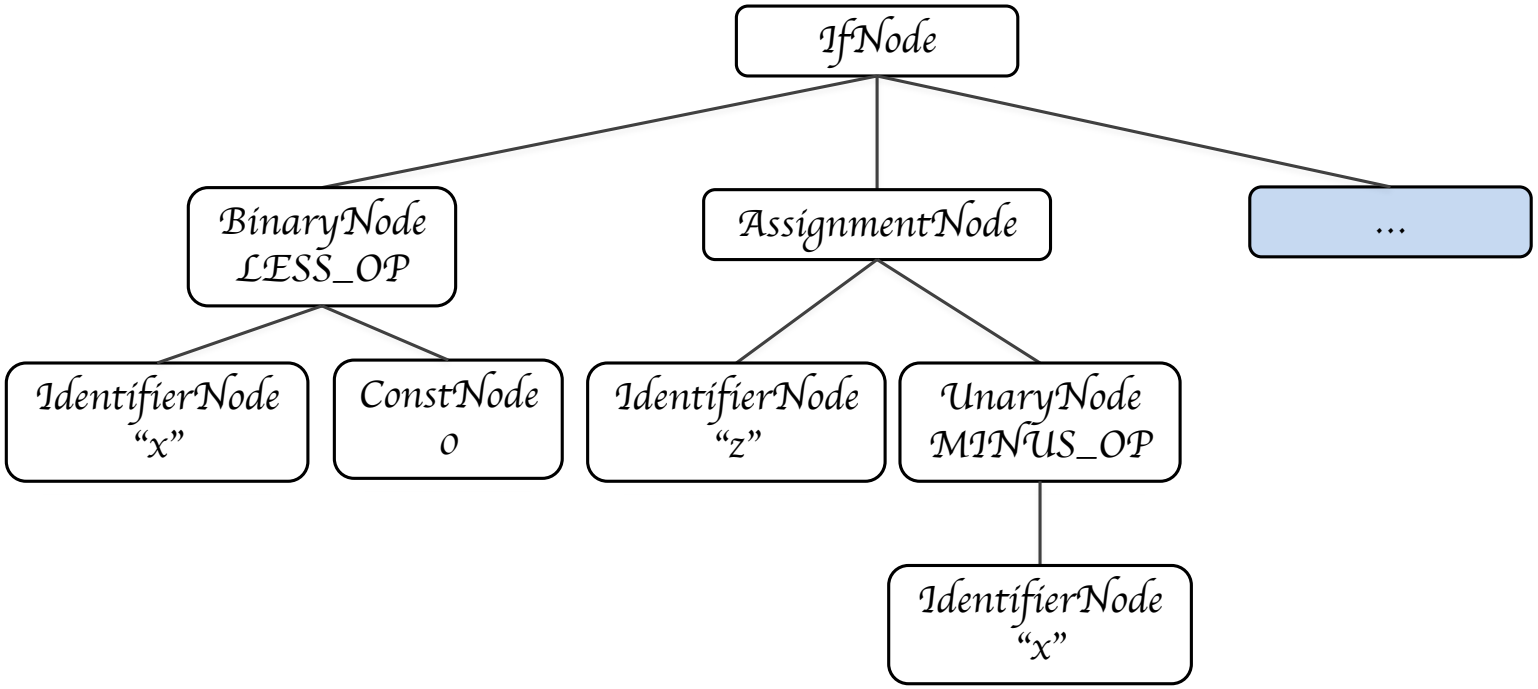
Syntax Analysis: Concrete Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$



Syntax Analysis: Abstract Syntax Tree

if $x < 0$ **then** $z := -x$ **else** $z := x$



Static Analysis: Type Checking the AST

if $x < 0$ **then** $z := -x$ **else** $z := x$