Expression Trees

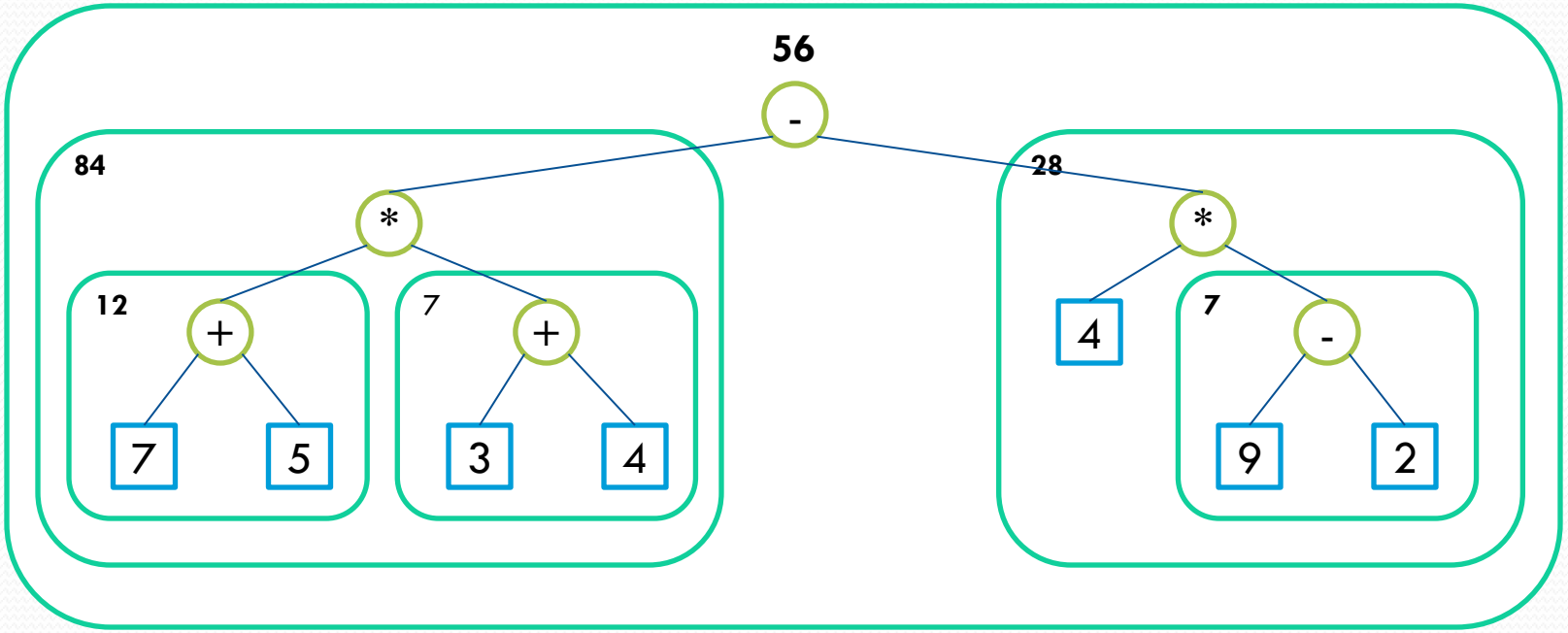
Expression Trees

- Expression trees are like binary trees, but they are not "sorted" in the usual way.
- Expression trees are a way to
  - Solve arithmetic expressions.
  - Compile a language

Expression Trees: Example

- Evaluation the tree in *post-order traversal* yields the same result

$$(7 + 5) * (3 + 4) - (4 * (9 - 2)) = 56$$



Grammar

- Grammar is a programming language-independent way of describing a a *syntax*.
- This is an example of the grammar for an expression.

```
<expression> ::= <term> { <addop> <term> }
<term>       ::= <factor> { <mulop> <factor> }
<factor>     ::= ( <expression> ) | <identifier> | <literal>
<addop>      ::= + | -
<mulop>      ::= * | /
<identifier> ::= a | b | c | ... | z | A | B | C | ... | Z
<literal>    ::= 0 | 1 | 2 | ... | 9
```

- Note that the grammar is (indirectly) recursive.
  - Vertical bars read as “OR”
  - Curly braces means that the item inside can be repeated 0 or more times

Grammar continued

- Our "*language*" consists of the following tokens

```
()+-*/abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

- Valid Expressions:
  - `A, B, 1, A + 2, A + B, A * (B), A * (B-2), (1)`
- Invalid Constructs:
  - `AB, 3A, 123, A(3), A + (), A * -3`
    - Remember that this is our language, in C++, most of these constructs will be valid.

Parsing

```
MakeExpression(Tree)
    Make a term, setting Tree to point to it
    while the next token is '+' or '-'
    Make an operator node, setting left child to Tree and right to NULL. (Tree
    points to new node)
    Get the next token.
    Make a term, setting the right child of Tree to point to it.
    end while
End MakeExpression
```

```
MakeTerm(Tree)
    Make a factor, setting Tree to point to it
    while the next token is '*' or '/'
    Make an operator node, setting left child to Tree and right to NULL. (Tree
    points to new node)
    Get the next token.
    Make a factor, setting the right child of Tree to point to it.
    end while
End MakeTerm
```

Parsing

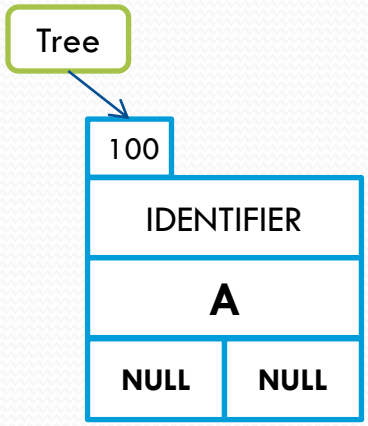
```
MakeFactor(Tree)
        if current token is '(', then
        Get the next token
        Make an expression, setting Tree to point to it
        else if current token is an IDENTIFIER
        Make an identifier node, set Tree to point to it, set left/right
children to NULL.
        else if current token is a LITERAL
        Make a literal node, set Tree to point to it, set left/right children
to NULL.
        end if
        Get the next token
End MakeFactor
```

```
GetNextToken
    while whitespace
        Increment CurrentPosition
    end while
    CurrentToken = Expression[CurrentPosition]
    Increment CurrentPosition
End GetNextToken
```

Example: “A + B”

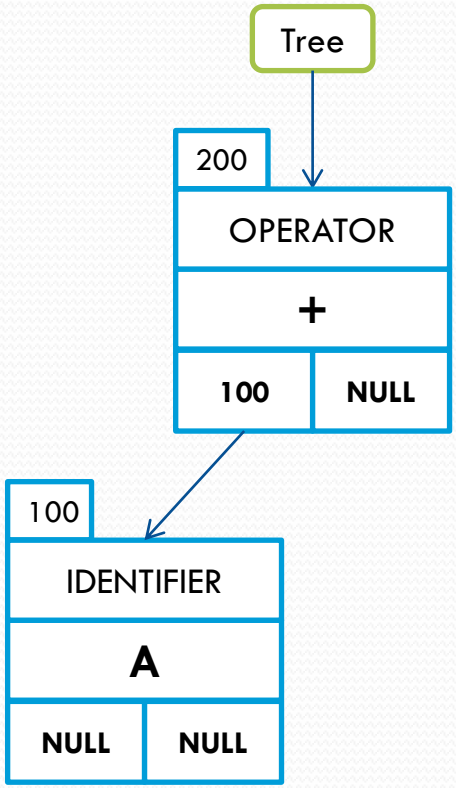
1. Make an IDENTIFIER node and set `Tree` to point to this term.
    1. EXPRESSION
    2. TERM
    3. FACTOR
    4. IDENTIFIER

“A”



Example: “A + B”

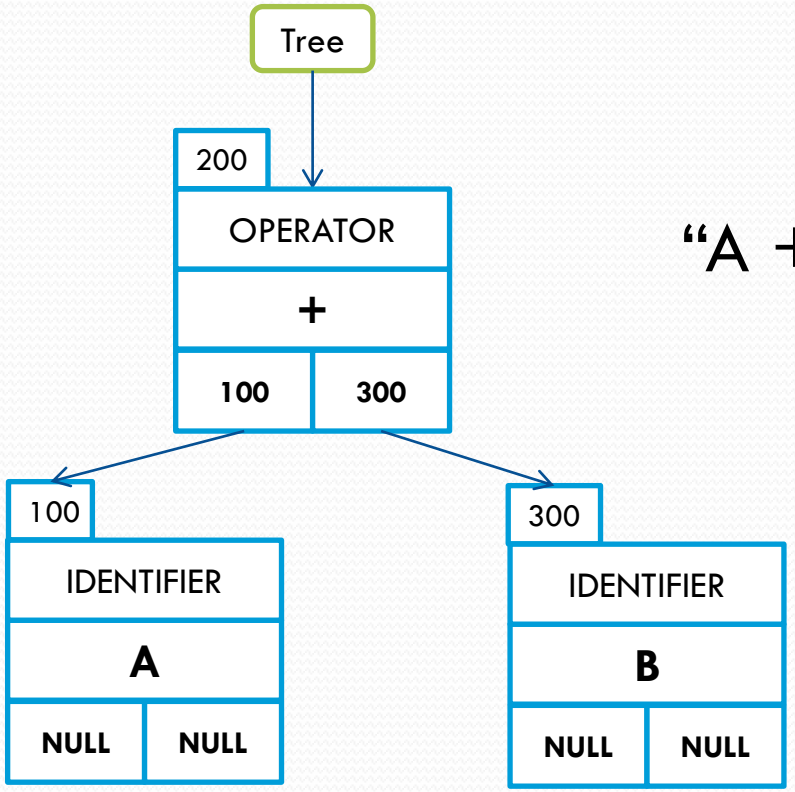
1. Make an OPERATOR node set left child to `Tree` and right child to `NULL` (Tree now points to this new operator node).



“A + ”

Example: “A + B”

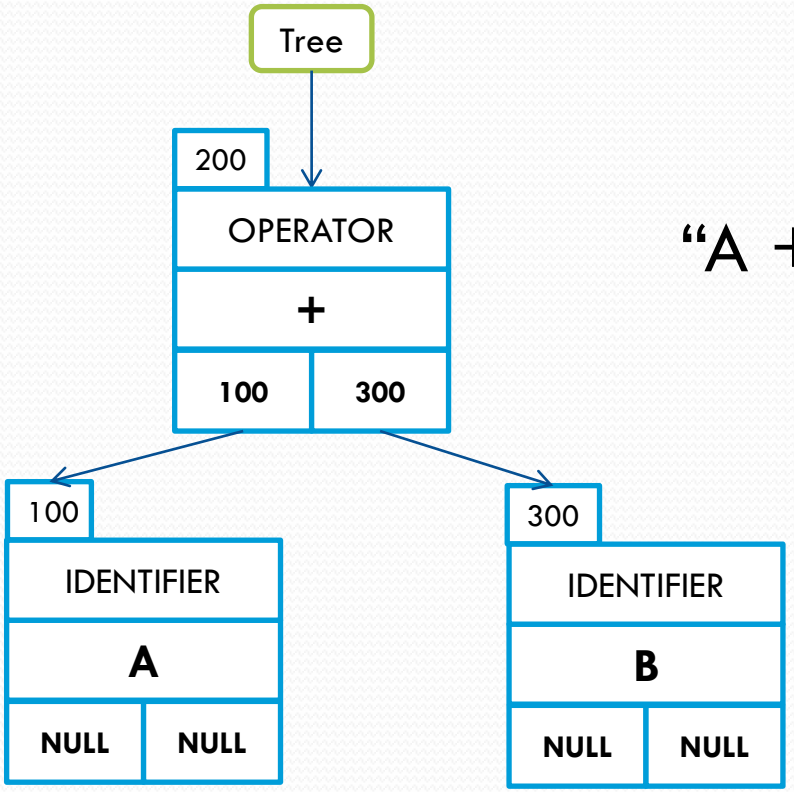
1. Make an IDENTIFIER node and set right child of `Tree` to point to this term.



“A + B ”

Example: “A + B * 5”

1. Starting with previous tree… Adding “* 5”

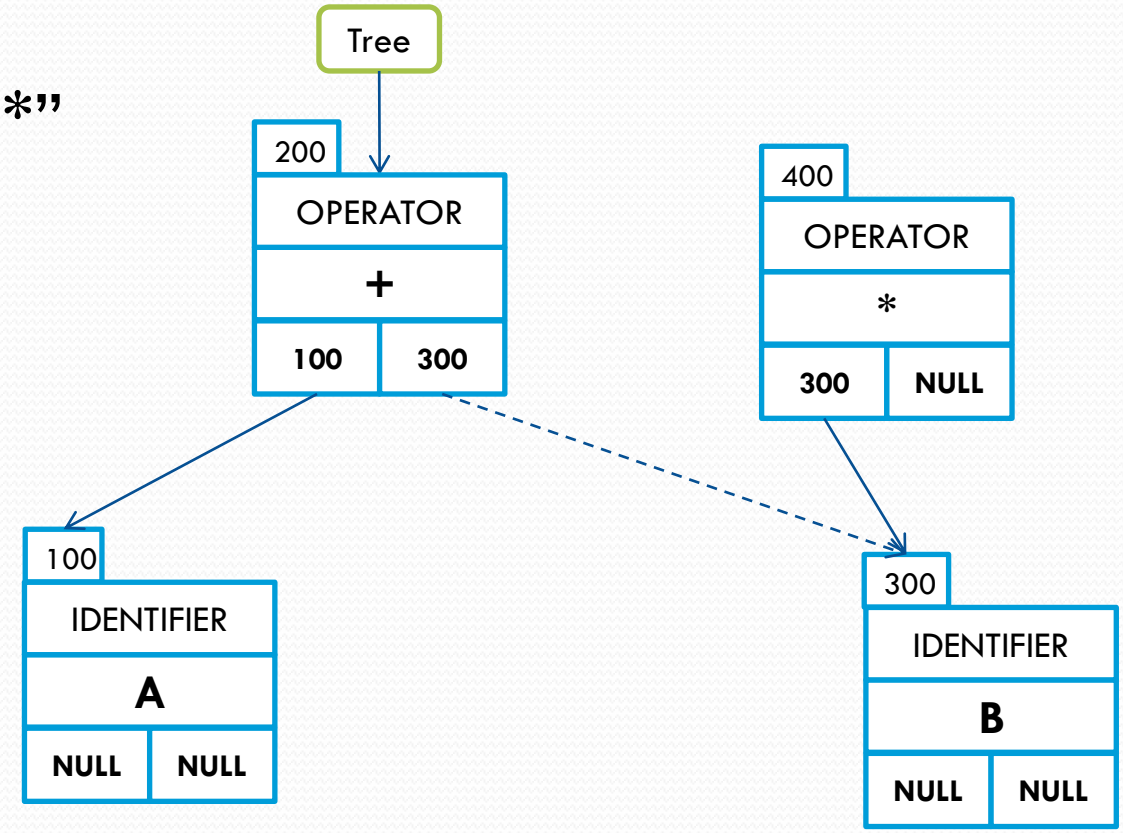


“A + B ”

Example: “A + B * 5”

1. Make an OPERATOR node, set left child to `Tree` **(from above, tree is right pointer of ‘+’)** and right child to `NULL`. Tree now points to this new operator node.

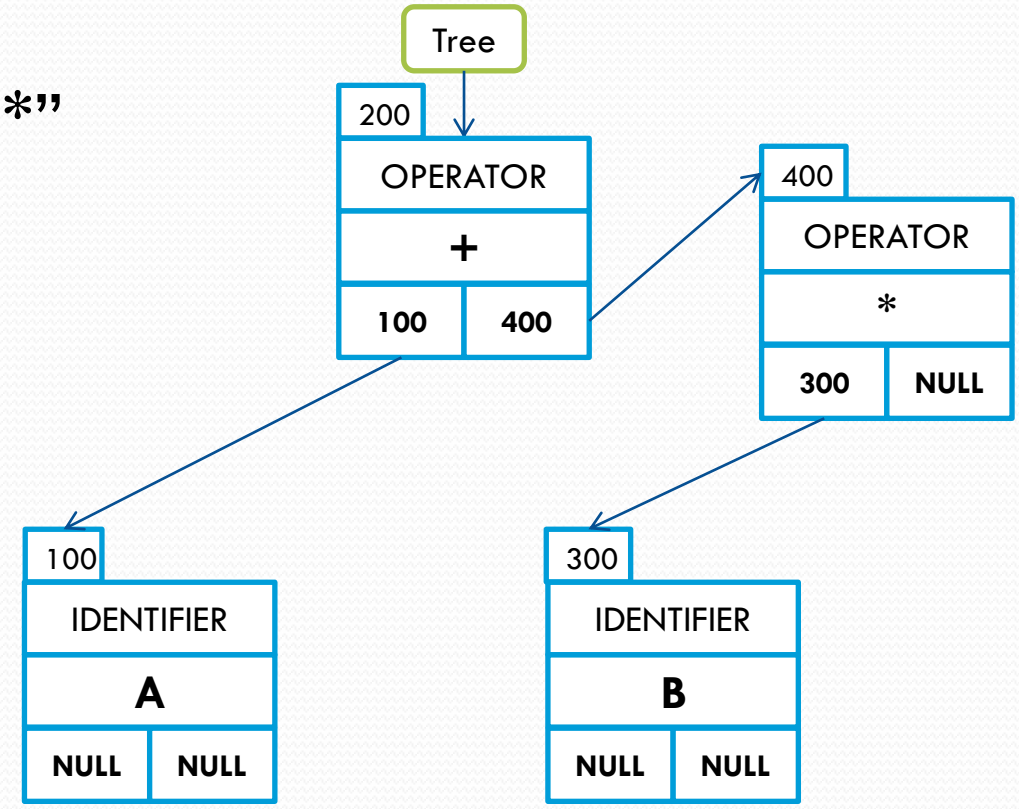
“A + B *”



Example: “A + B * 5”

1. Make an OPERATOR node, set left child to `Tree` (from above, tree is right pointer of ‘+’) and right child to `NULL`. Tree now points to this new operator node.

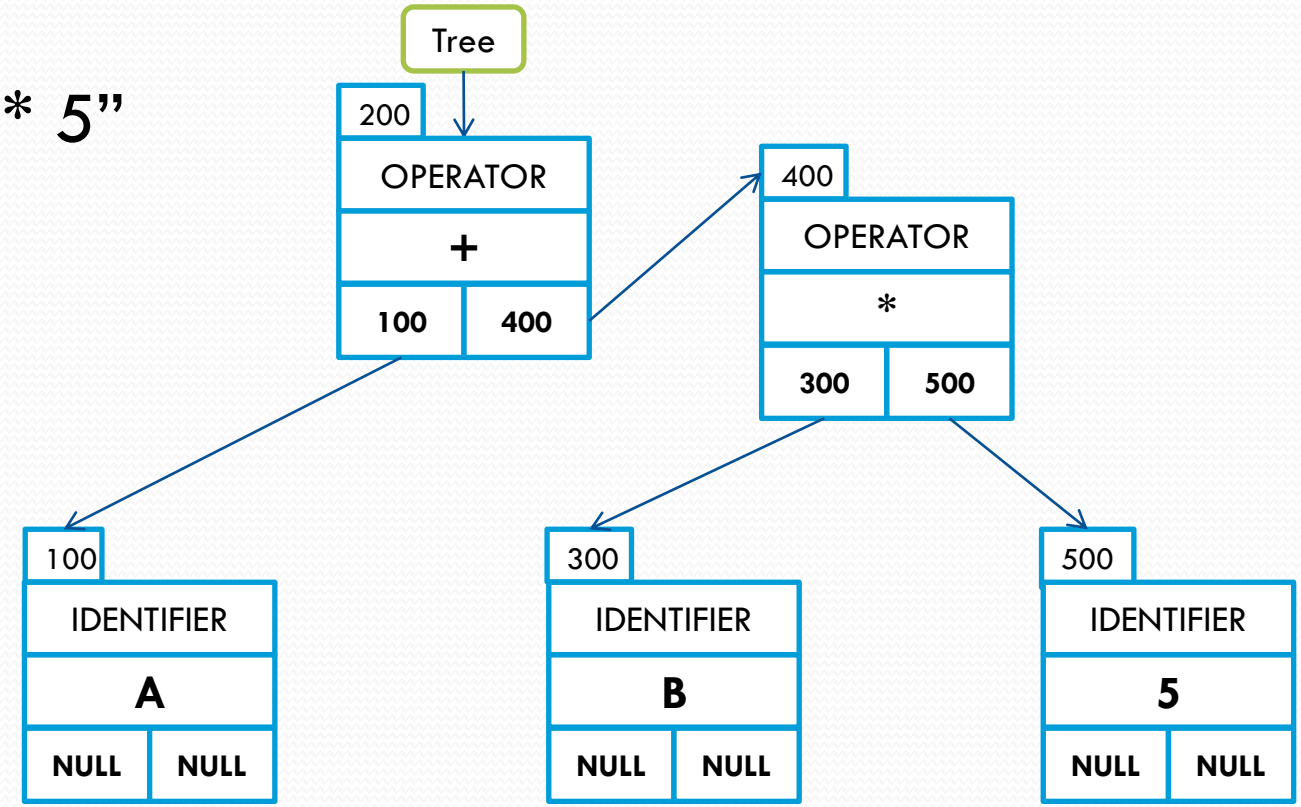
“A + B *”



Example: “A + B * 5”

1. Make an OPERATOR node, set left child to `Tree` (from above, tree is right pointer of ‘+’) and right child to `NULL`. Tree now points to this new operator node.

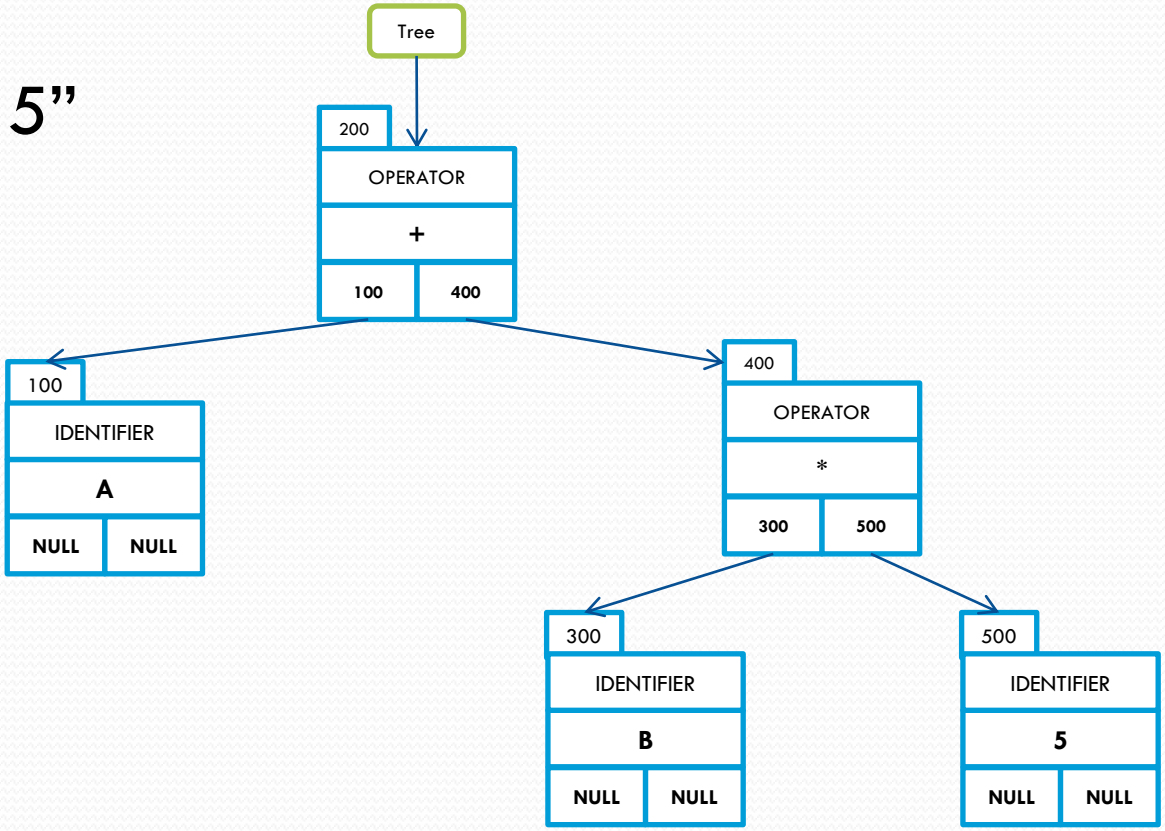
“A + B * 5”



Example: “A + B * 5”

1. Make an OPERATOR node, set left child to `Tree` (from above, tree is right pointer of ‘+’) and right child to `NULL`. Tree now points to this new operator node.

“A + B * 5”



Exercise

- Build the parse tree for these expressions:
  - $A + B + C$
  - $A * B + C$
  - $(A + B) * C$
  - $A + B * C$
  - $A * (B + C)$