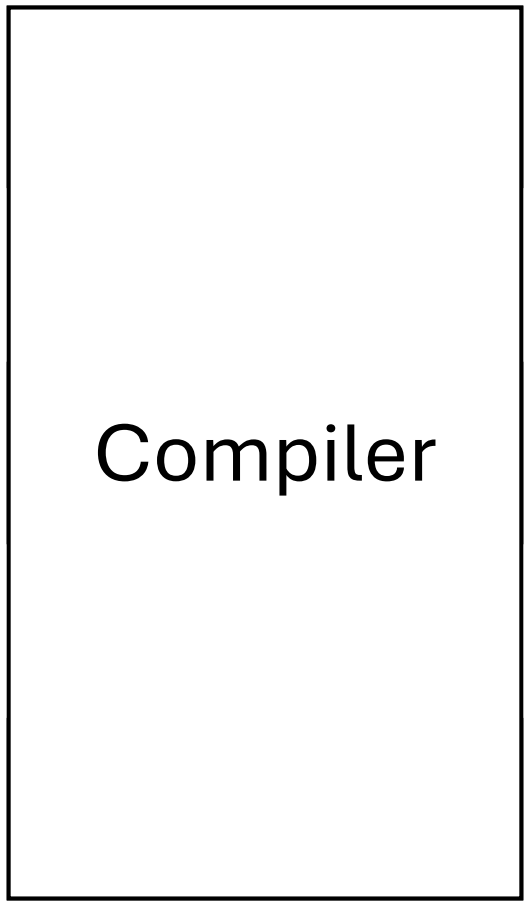
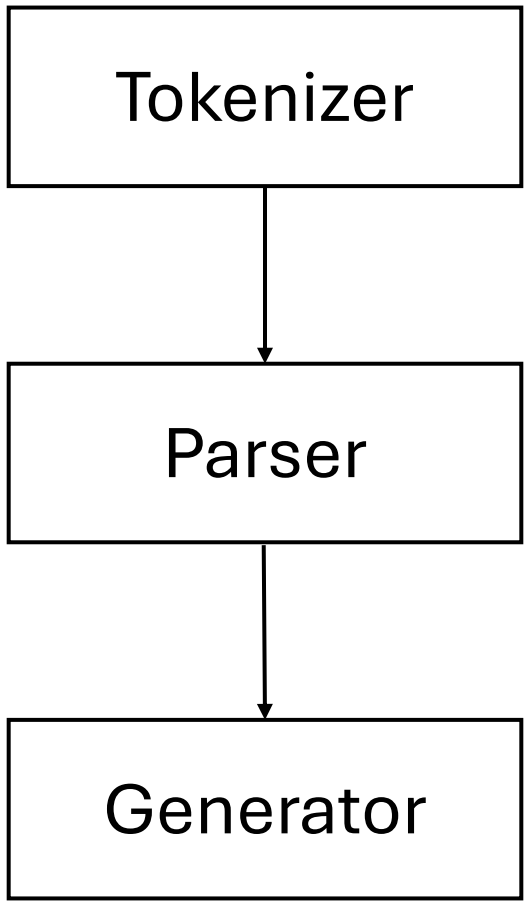
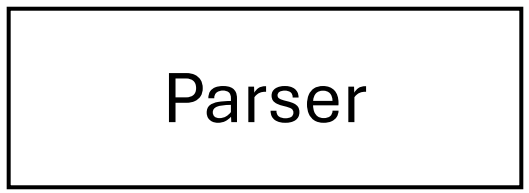
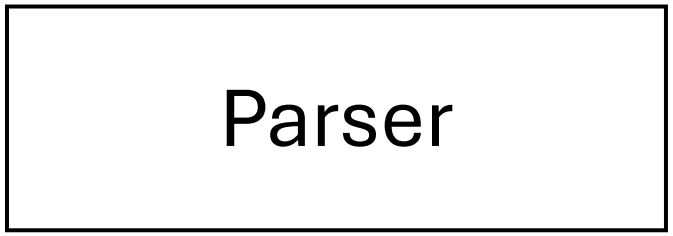
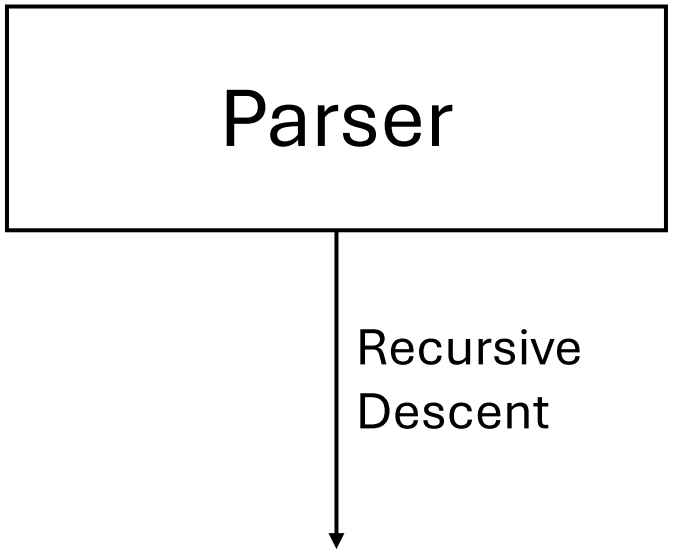
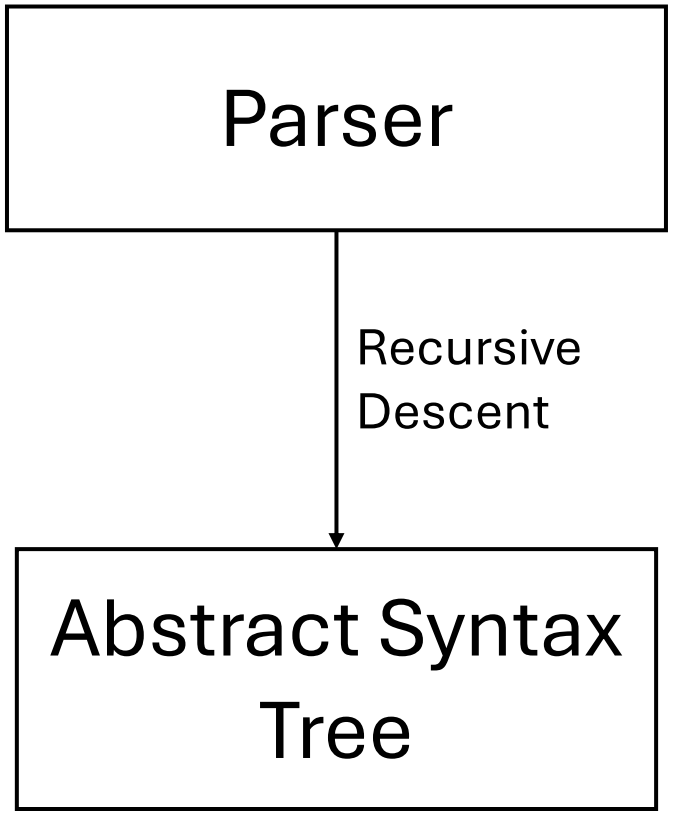

Compiler

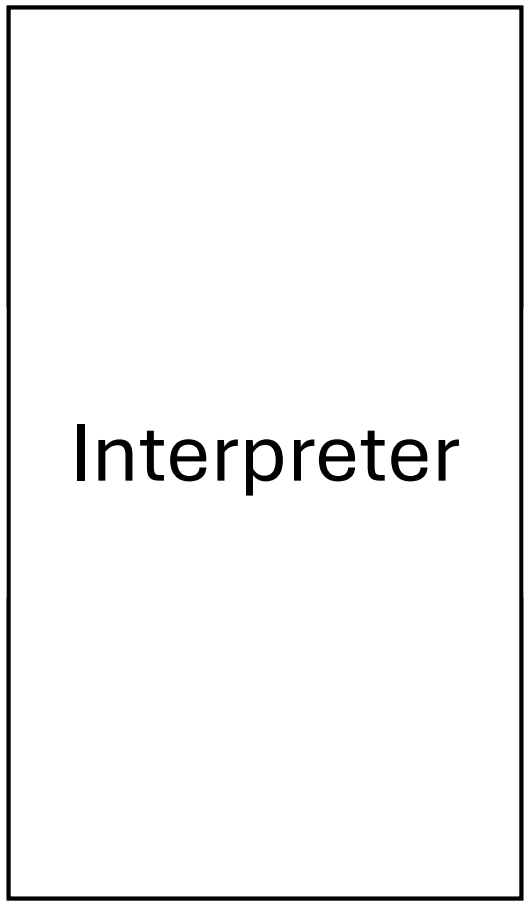

Tokenizer
Parser
Generator

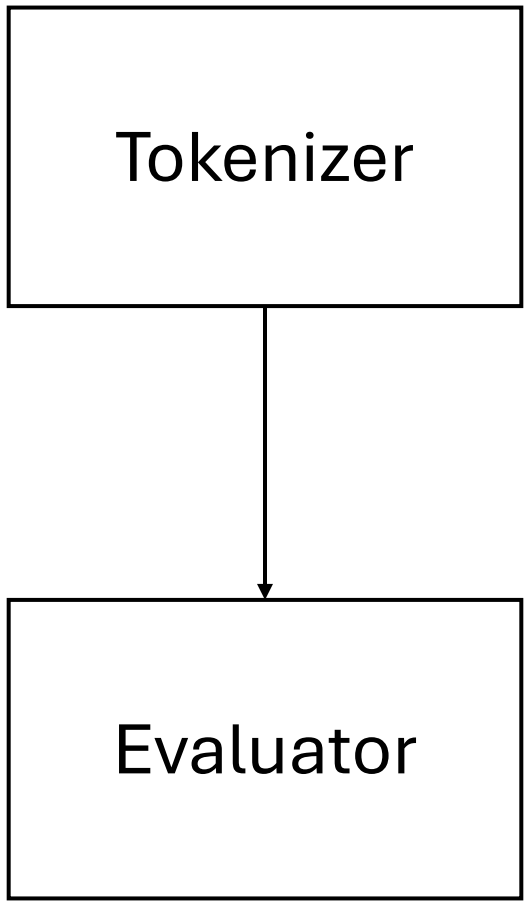

Parser

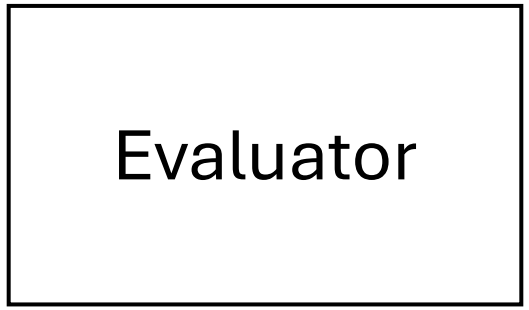

Parser


Parser
Recursive
Descent

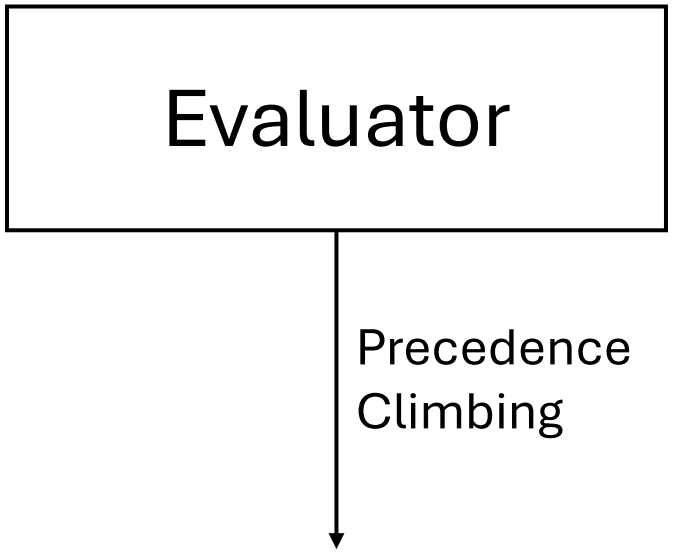

Parser
Recursive
Descent
Abstract Syntax
Tree

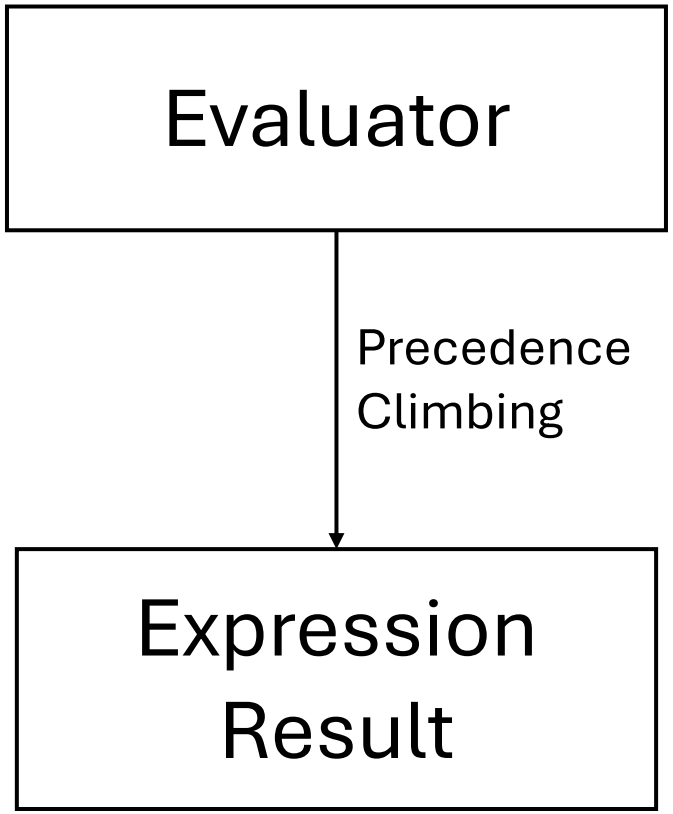

Interpreter


Tokenizer
Evaluator


Evaluator


Evaluator


Evaluator
Precedence
Climbing


Evaluator
Precedence
Climbing
Expression
Result

Task:

Take a file containing an expression,
return the value of the expression

Tokenization:

Convert string of characters to “tokens”, containing meaning in them

```
struct Token {
    int token_type;
    int value;
};
```

Expression: 3 + 5 * 2

- Corresponding tokensArray:

Values:

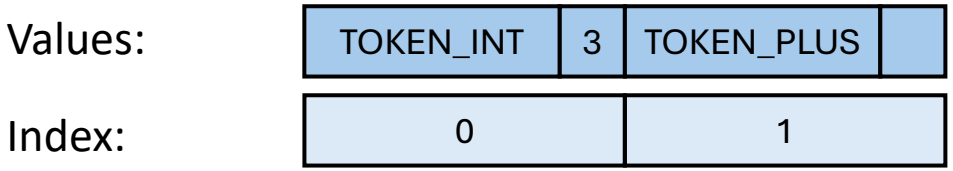
Index:

Expression: 3 + 5 * 2

- Corresponding tokensArray:

Expression: 3 + 5 * 2

- Corresponding tokensArray:

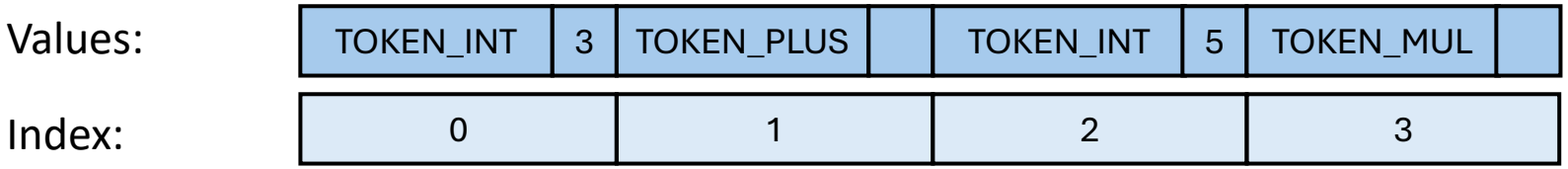


Expression: 3 + 5 * 2

- Corresponding tokensArray:

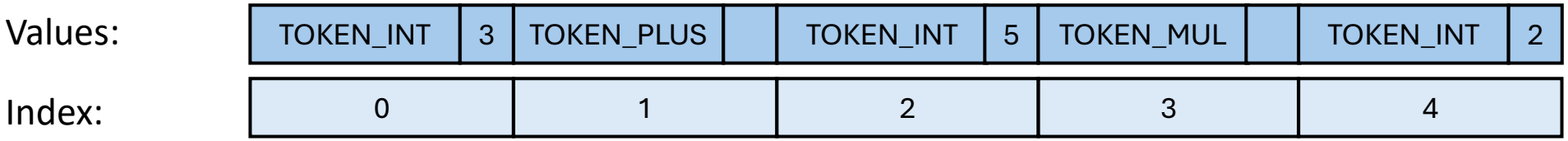
Expression: 3 + 5 * 2

- Corresponding tokensArray:



Expression: 3 + 5 * 2

- Corresponding tokensArray:



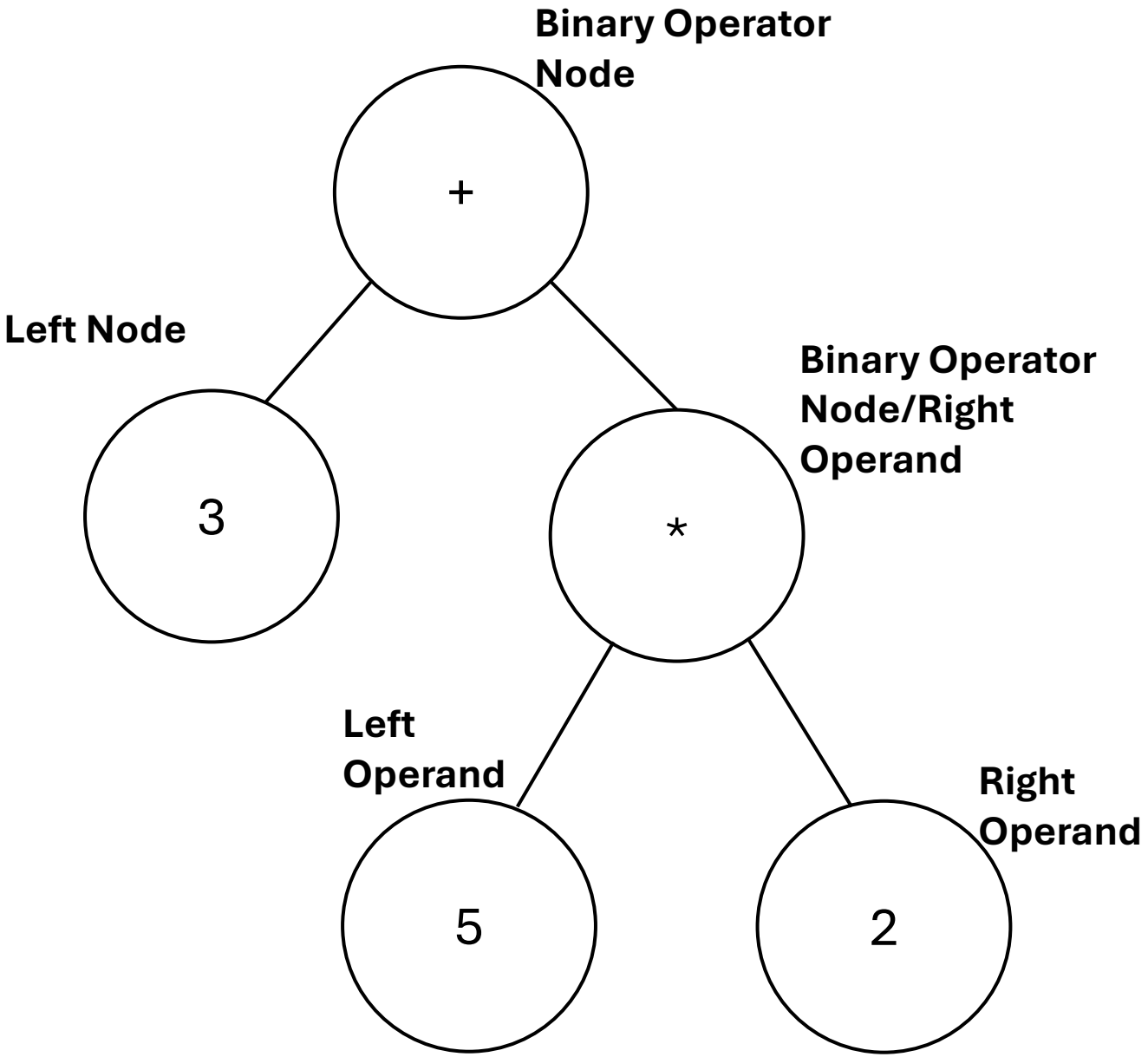
Expression: 3 + 5 * 2

- Corresponding tokensArray:

Values:	TOKEN_INT	3	TOKEN_PLUS		TOKEN_INT	5	TOKEN_MUL		TOKEN_INT	2	TOKEN_EOE	
Index:	0		1		2		3		4		5	

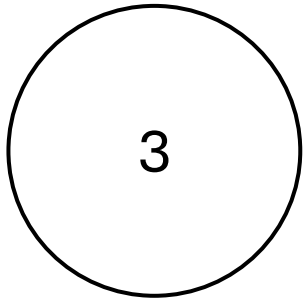
Parsing:

- Convert tokens to something else
- In compilers, this is an AST
  - Useful for representing source code structure
- In interpreters, this is evaluation

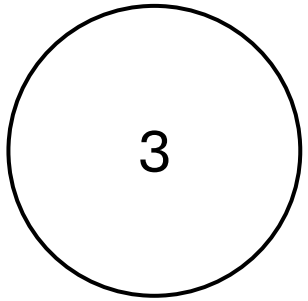

Binary Operator Node
+
Left Node
3
Binary Operator Node/Right Operand
*
Left Operand
5
Right Operand
2

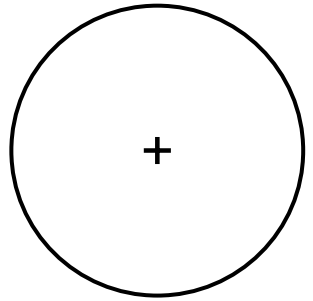
3 + 5 * 2

3 + 5 * 2

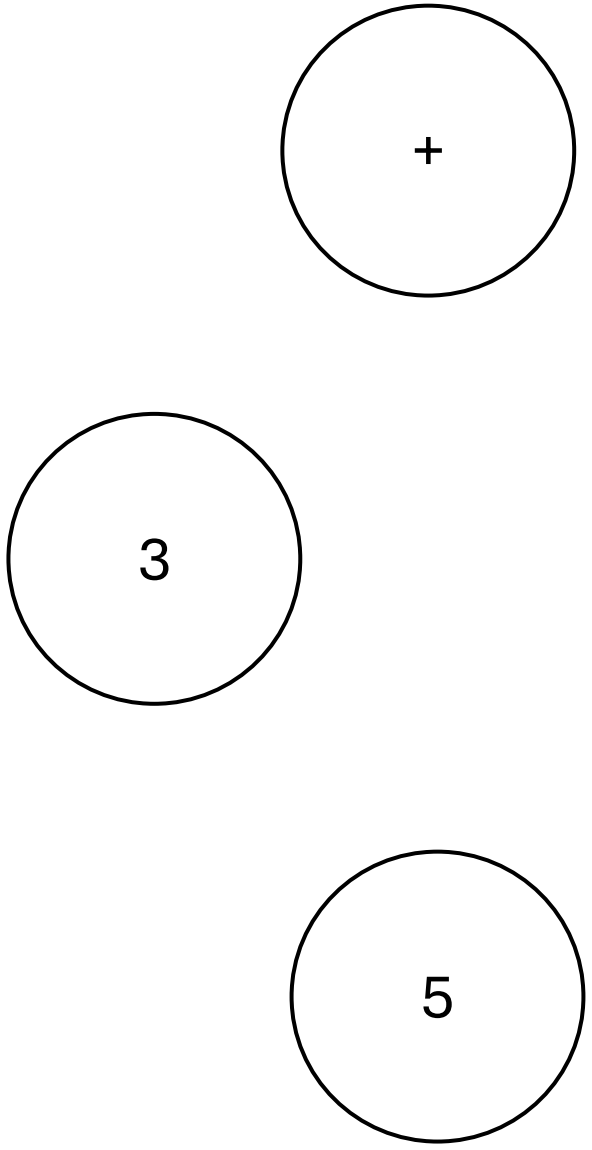


3 + 5 * 2

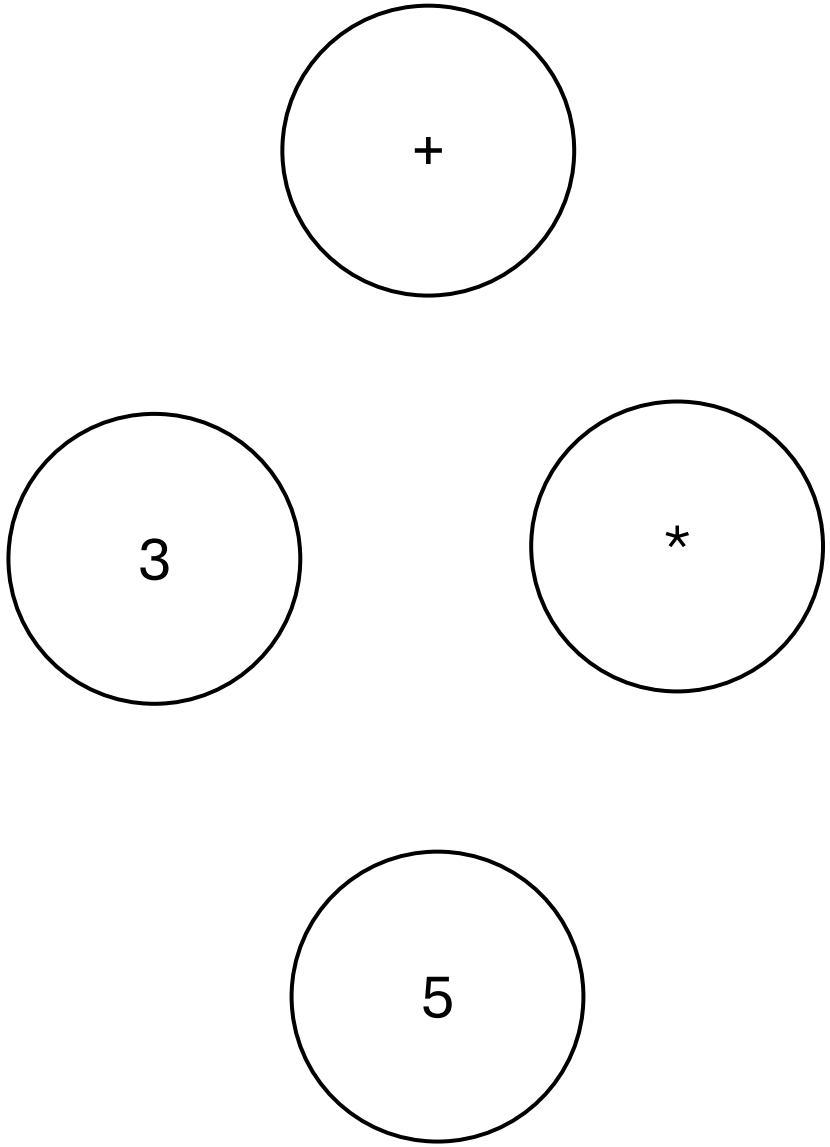




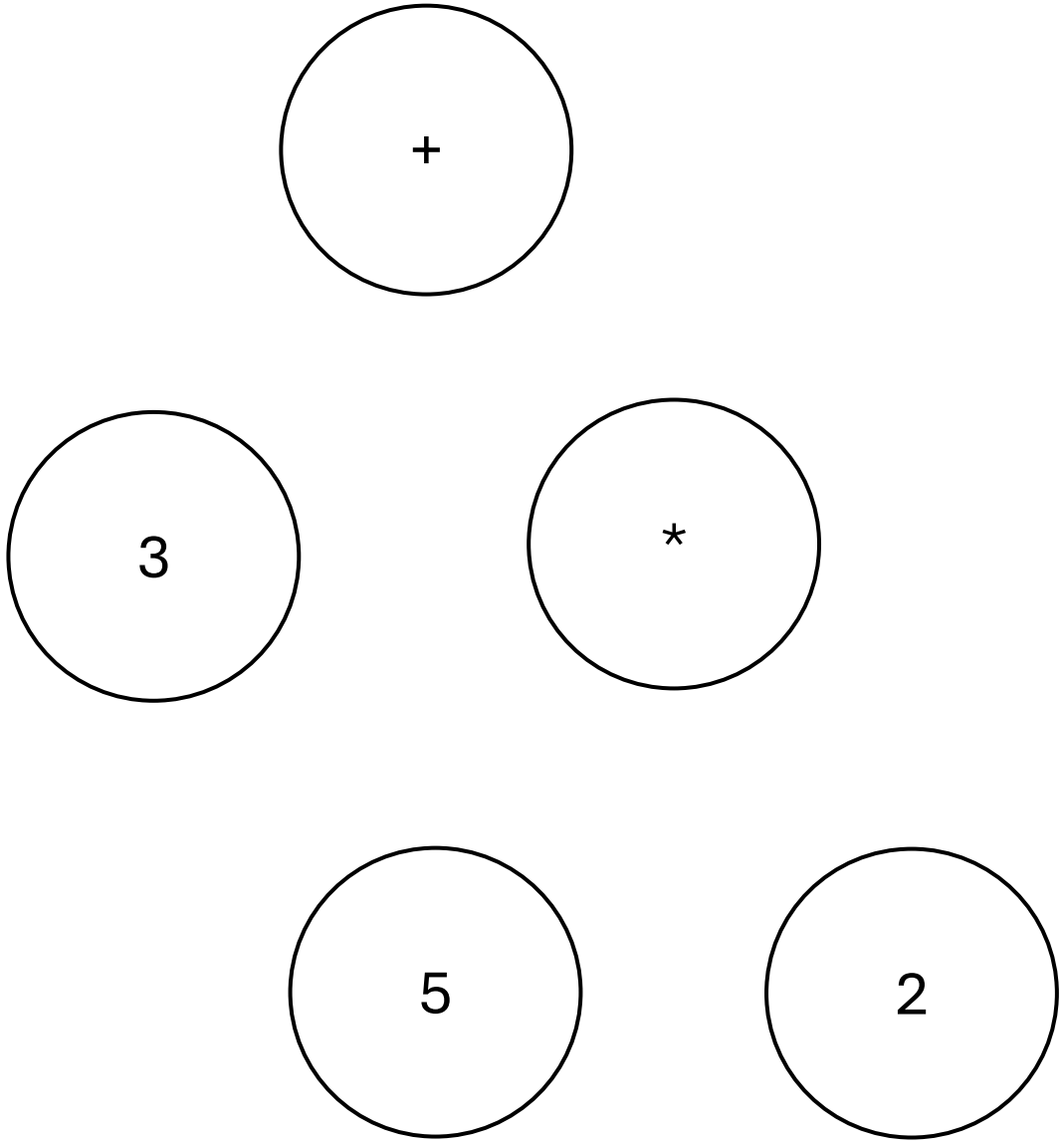
3 + 5 * 2



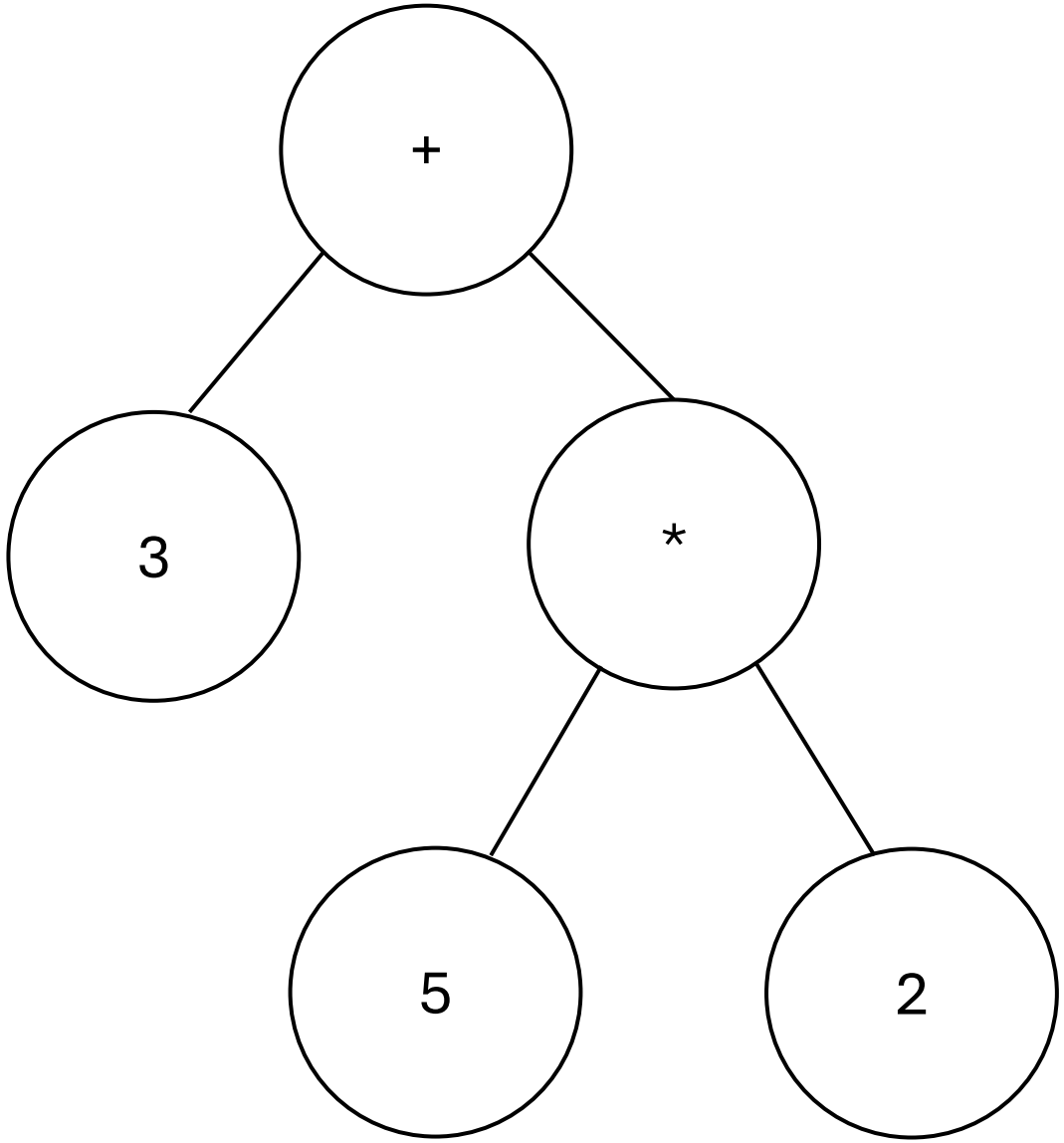
3 + 5 * 2



3 + 5 * 2



3 + 5 * 2



Precedence:

- Operators have precedence
  - Says when operator must be evaluated
- Unary operators evaluated innermost to outermost
  - Example on webpage

Tokens	Precedence
TOKEN_PLUS (+)	1
TOKEN_MINUS (-)	1
TOKEN_MUL (*)	2
TOKEN_DIV (/)	2
TOKEN_BITWISE_AND (&)	3
TOKEN_BITWISE_OR (|)	4
TOKEN_SHIFT_LEFT (<<)	5
TOKEN_SHIFT_RIGHT (>>)	5
TOKEN_LOGICAL_AND (&&)	6
TOKEN_LOGICAL_OR (||)	7

Associativity:

- Some operators have same precedence
- Example: 1 - 2 + 3
- Left-associative: parsed as (1 - 2) + 3
- Right-associative: parsed as 1 - (2 + 3)
- All operators are left-associative in this lab

Algorithm:

- Pseudocode on webpage

Evaluation

```
// Current precedence will be set to 1 higher than operator precedence
// This forces left-associativity
right_operand, position = parse_expression(tokens, position, op_prec + 1)

// In this line, token is an operator
left_operand = evaluate(left_operand, token, right_operand)
```

evaluate() section replaced by operations, in webpage

Functions:

- parse
- precedence
- parse_expression
- parse_factor

parse:

- Main entry point of program
- Arguments:
  - a0: Pointer to expression string
  - a1: Length of string
- Returns:
  - a0: Result of expression

precedence:

- Calculates precedence of an operator
- Arguments:
  - a0: Token type
- Returns:
  - a0: Precedence if valid operator, -1 if not

parse_expression:

- Evaluates an expression
- Arguments:
  - a0: Pointer to array of tokens
  - a1: Position in tokens array
  - a2: Current precedence
- Returns:
  - a0: Result of expression
  - a1: Position in tokens array

parse_factor:

- Evaluates an integer/expression with unary operators
- Arguments:
  - a0: Pointer to array of tokens
  - a1: Position in tokens array
- Returns:
  - a0: Result of integer/expression
  - a1: Position in tokens array