# Grammar

```
Addition returns Expression:
  Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST

# Stack

1. Addition

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST

# Stack

1. Addition
2. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST

# Stack

1. Addition
2. Multiplication
3. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST

## Stack

1. Addition
2. Multiplication
3. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST

## Stack

1. Addition
2. Multiplication
3. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
    PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST

# Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST

# Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST

## Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication
6. PrimaryExpression

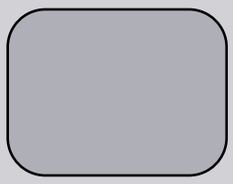
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST



# Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication
6. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** ( 1 + 20 ) * 2

## AST

1

## Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication
6. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST

1

# Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication
6. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** ( 1 + 20 ) * 2

**Reduce!**

## AST

1

## Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** ( 1 + 20 ) * 2

**Reduce!**

## AST


## Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST


# Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

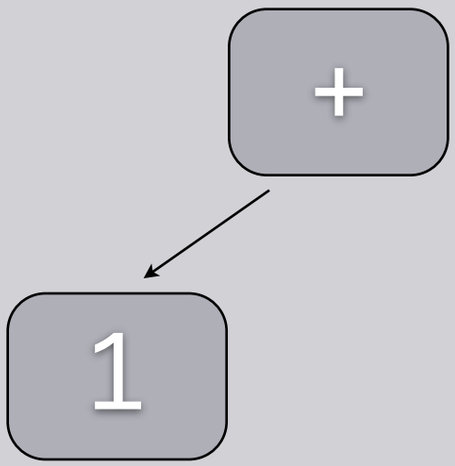
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST




# Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

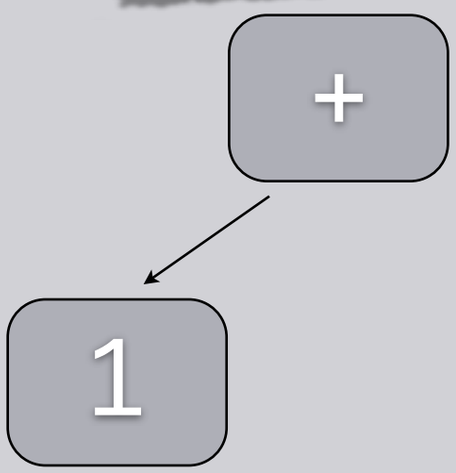
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
   {
   '
```

```
current = ruleMultiplication();
// {AdditionalExpression.left=current}
AdditionalExpression temp = new AdditionalExpression();
temp.setLeft(current);
current = temp:
```

Te

AS




2.Multiplication
3.PrimaryExpression
4.Addition

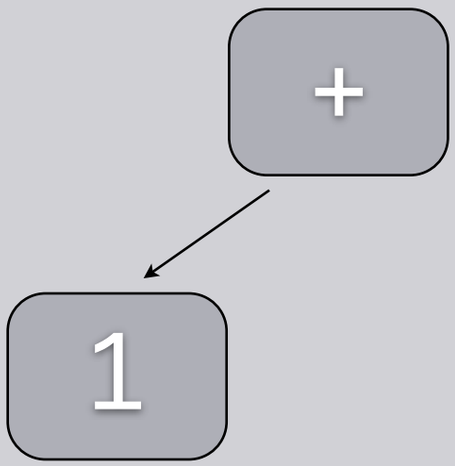
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST




# Stack

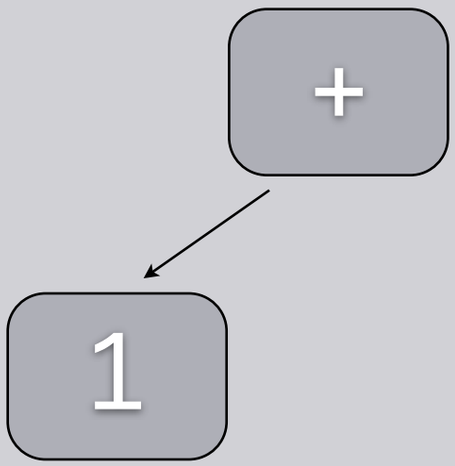
1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST




## Stack

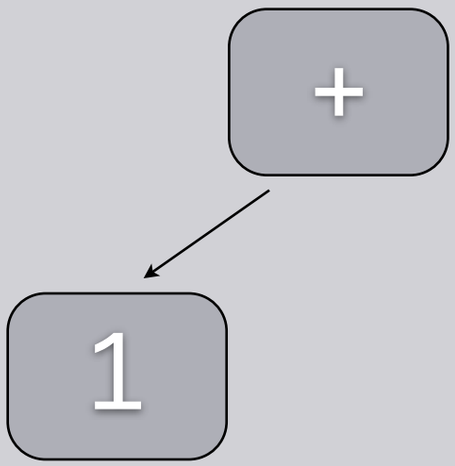
1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST




## Stack

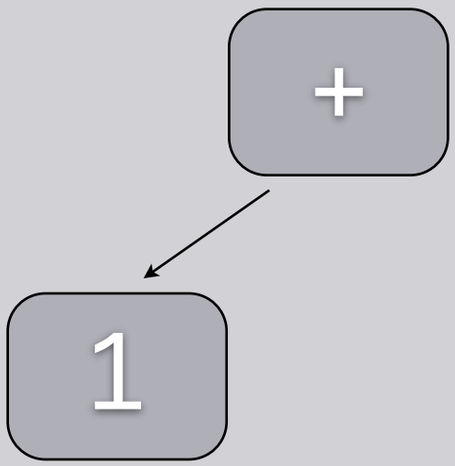
1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

# Grammar

```
Addition returns Expression:
    Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
  PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST




## Stack

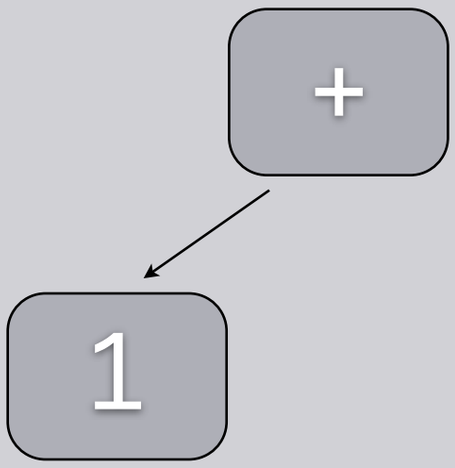
1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication

# Grammar

```
Addition returns Expression:
    Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
   {NumberLiteral} value=INT |
   '(' Addition ')';
```

**Text** ( 1 + 20 ) * 2

## AST




## Stack

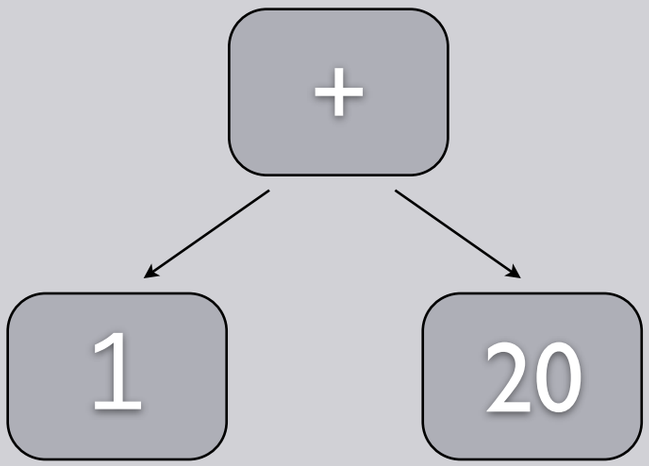
1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition
5. Multiplication
6. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST




**Reduce!**

## Stack

1. Addition
2. Multiplication
3. PrimaryExpression
4. Addition

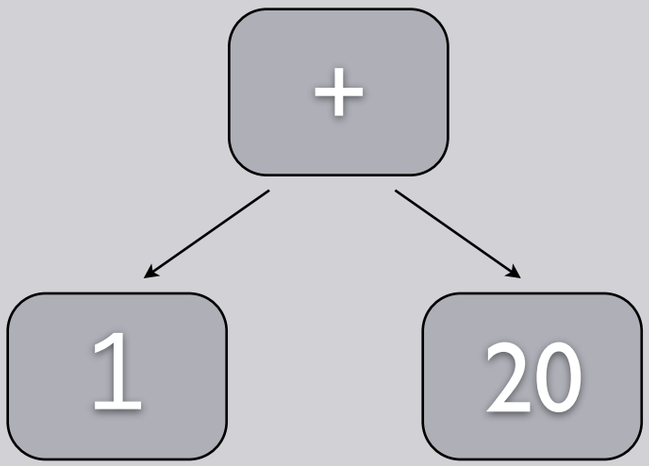
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** ( 1 + 20 ) * 2

**Reduce!**

## AST




## Stack

1. Addition
2. Multiplication
3. PrimaryExpression

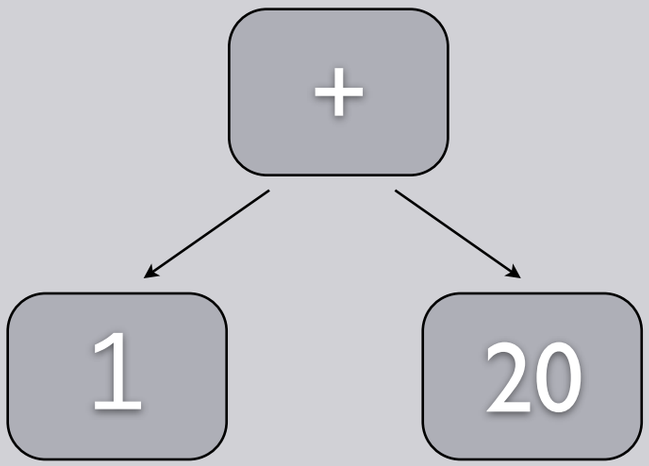
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST




# Stack

1. Addition
2. Multiplication
3. PrimaryExpression

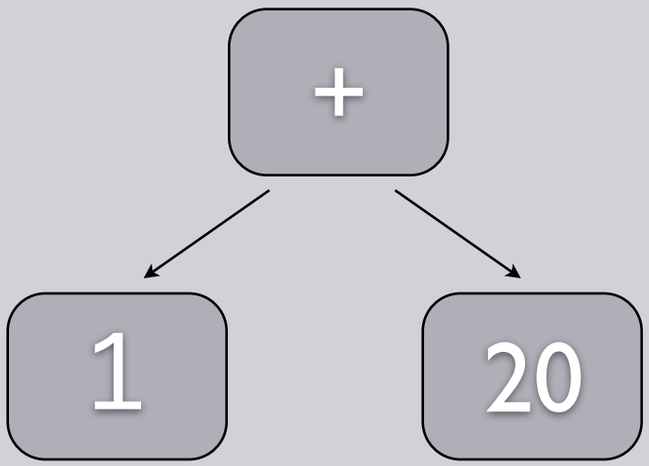
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

# Text

( 1 + 20 ) * 2

# AST




# Stack

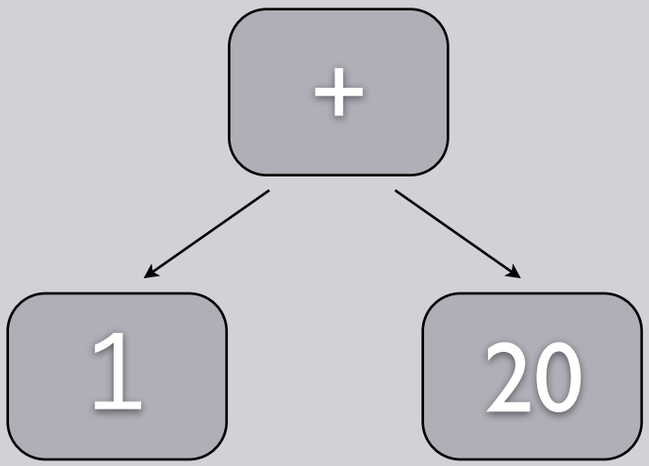
1. Addition
2. Multiplication
3. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** ( 1 + 20 ) * 2

## AST




## Reduce!

## Stack

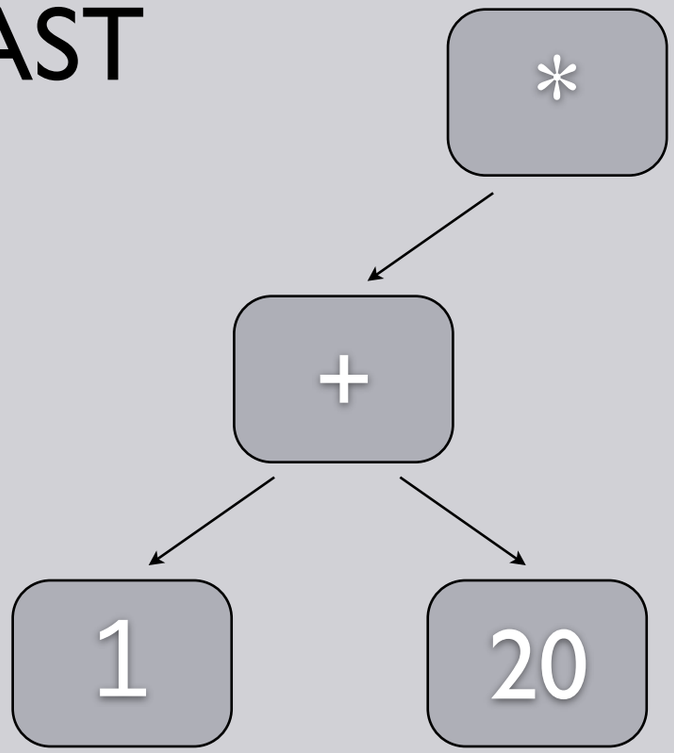
1. Addition
2. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST




## Stack

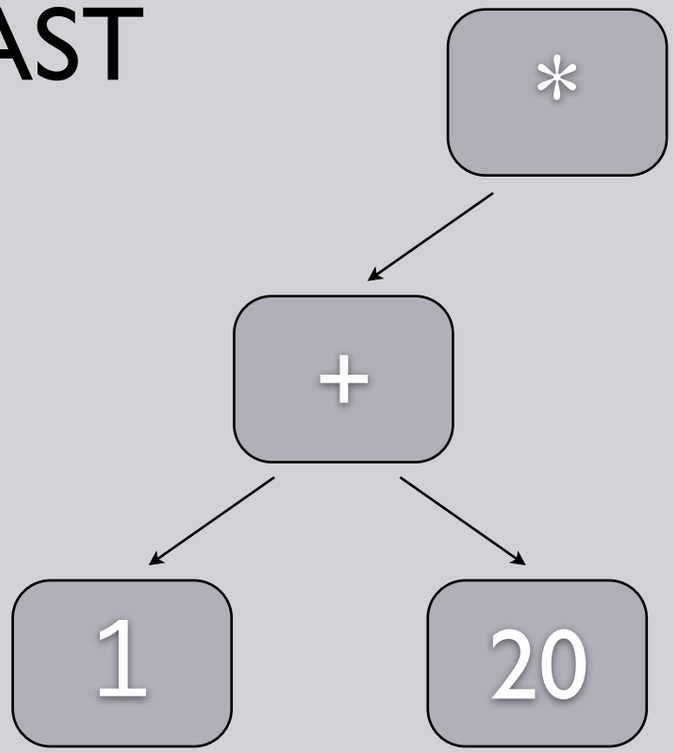
1. Addition
2. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST




## Stack

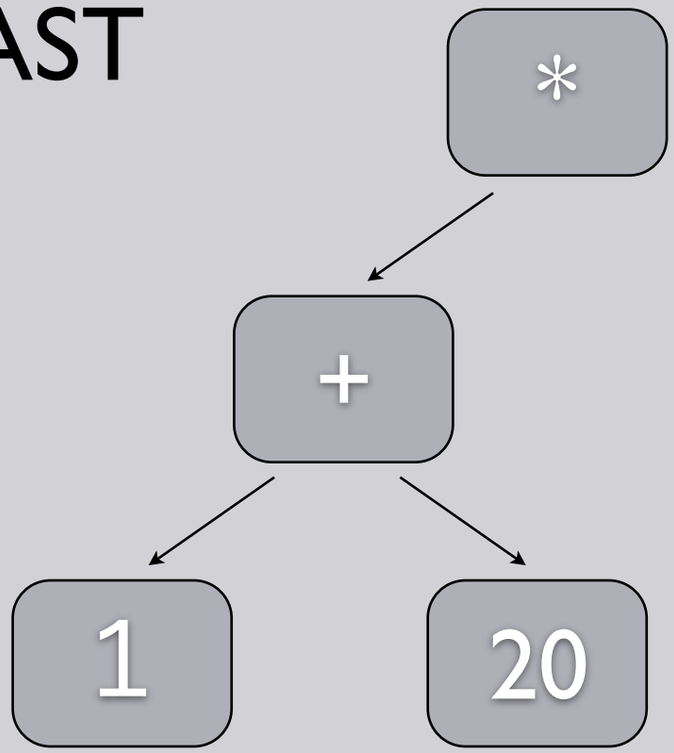
1. Addition
2. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

## AST




## Stack

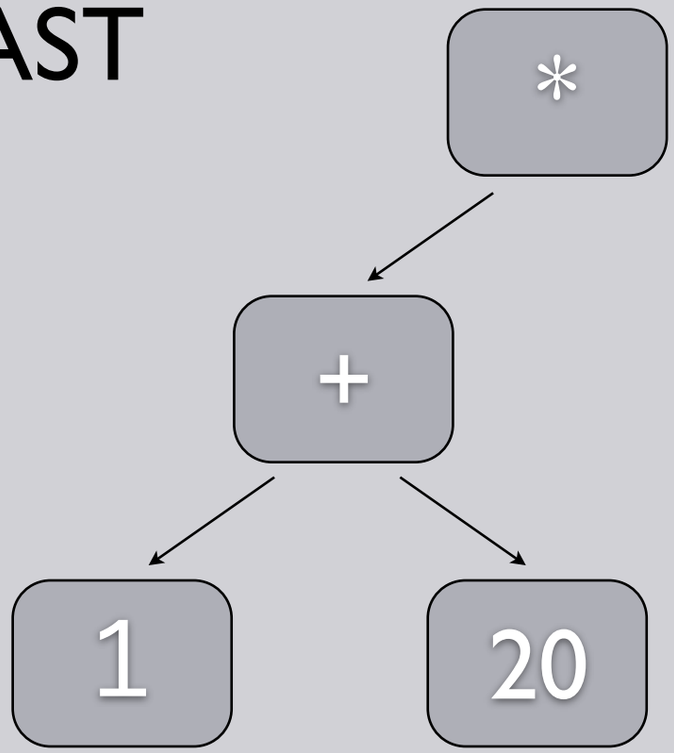
1. Addition
2. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** ( 1 + 20 ) * 2

**AST**




**Stack**

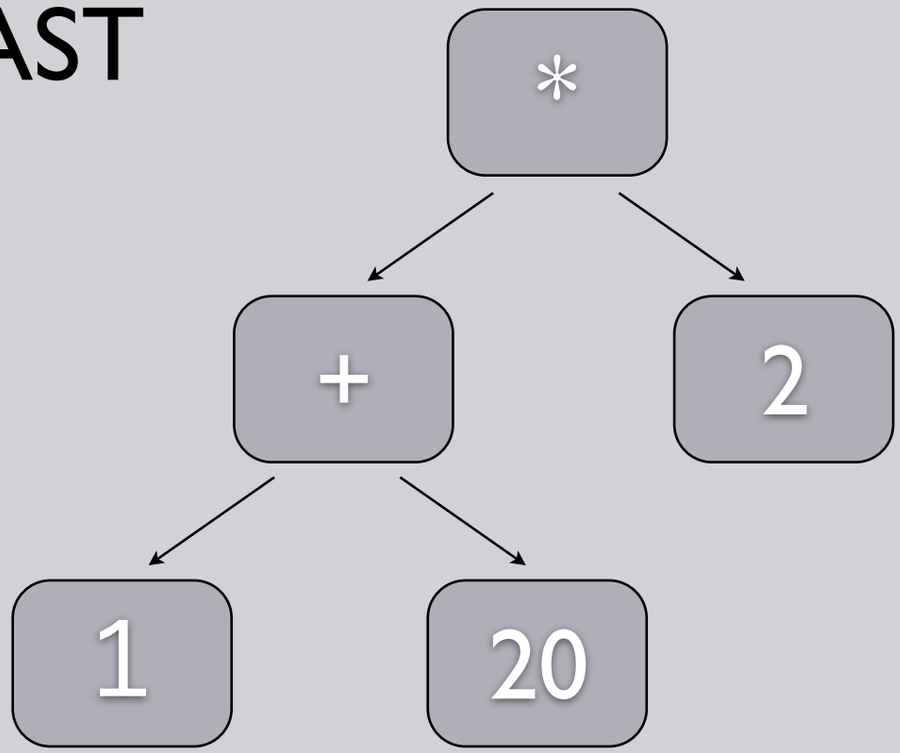
1. Addition
2. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2`

**AST**




**Stack**

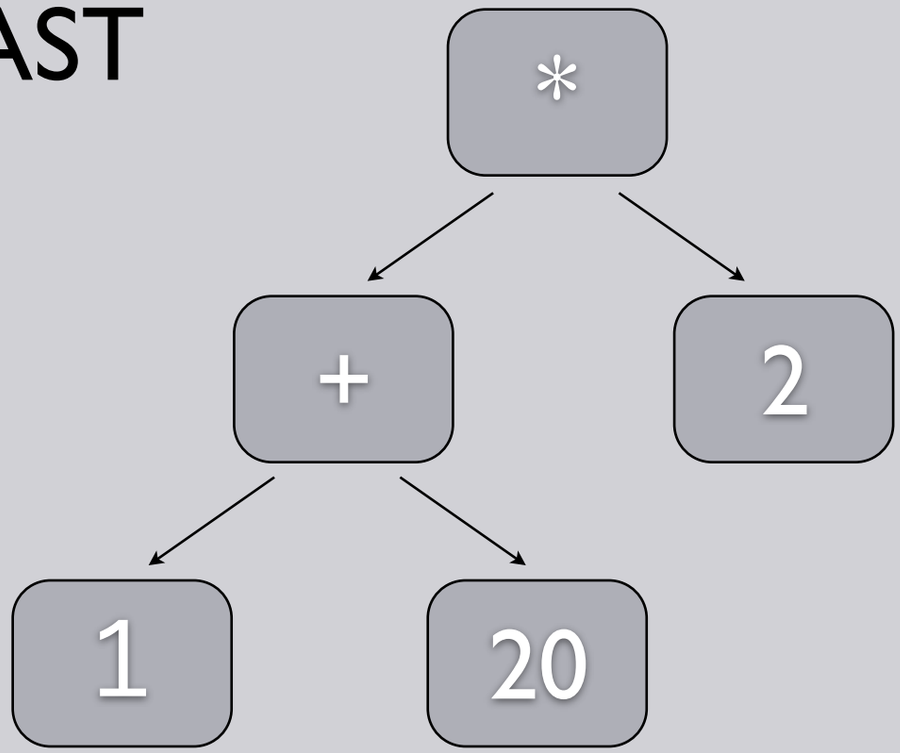
1. Addition
2. Multiplication
3. PrimaryExpression

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

Text ( 1 + 20 ) * 2 EOF

AST




Stack

1. Addition
2. Multiplication
3. PrimaryExpression

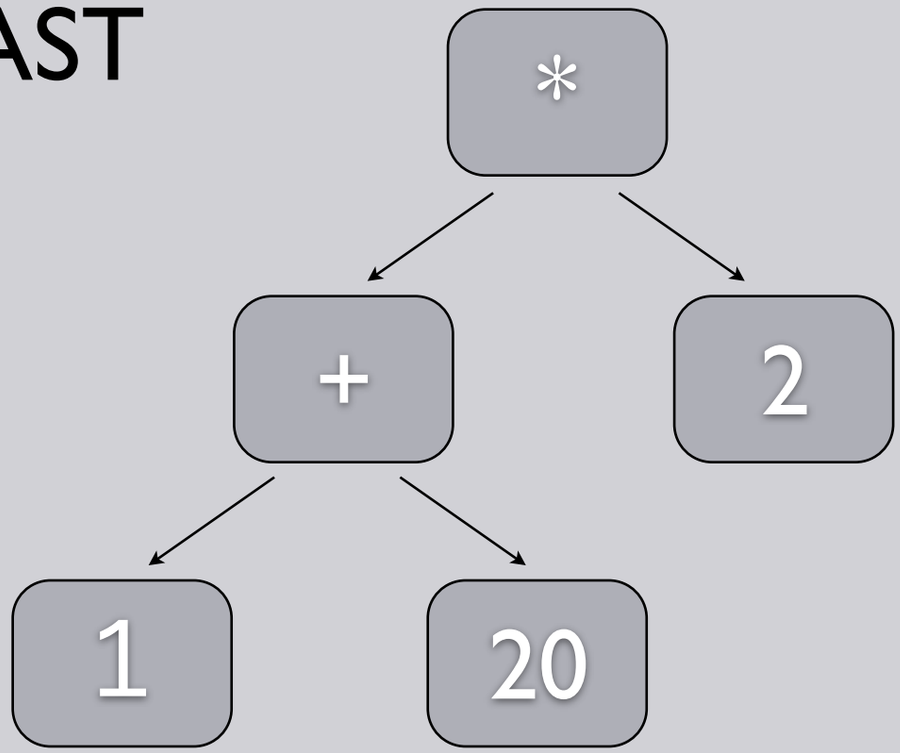
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

Text ( 1 + 20 ) * 2 EOF

Reduce!

AST




Stack

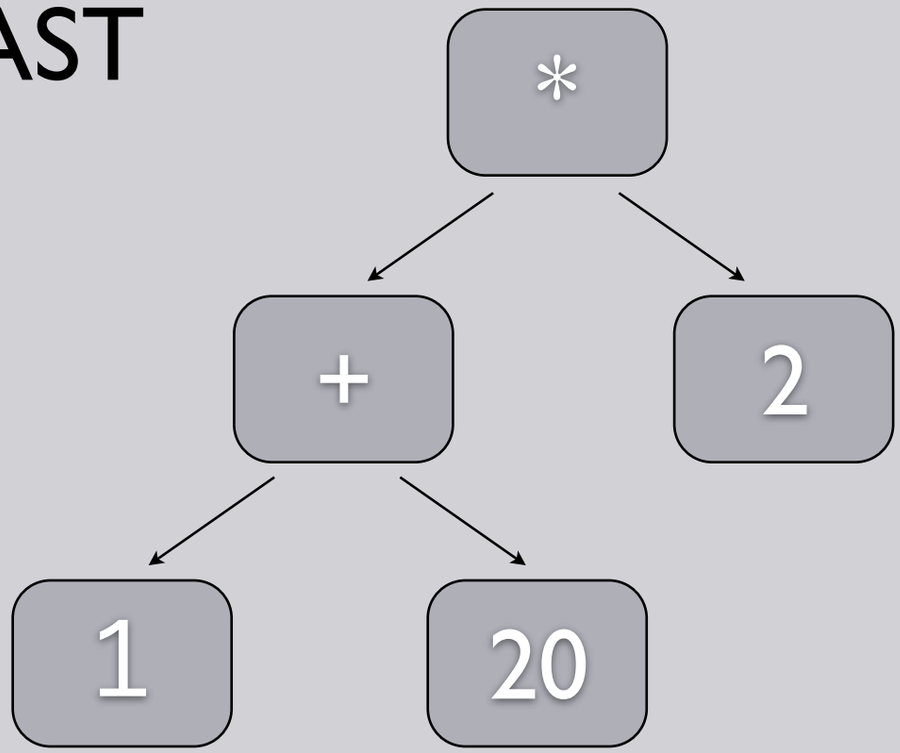
1. Addition
2. Multiplication

# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

**Text** `( 1 + 20 ) * 2` EOF

## AST




## Stack

1. Addition
2. Multiplication

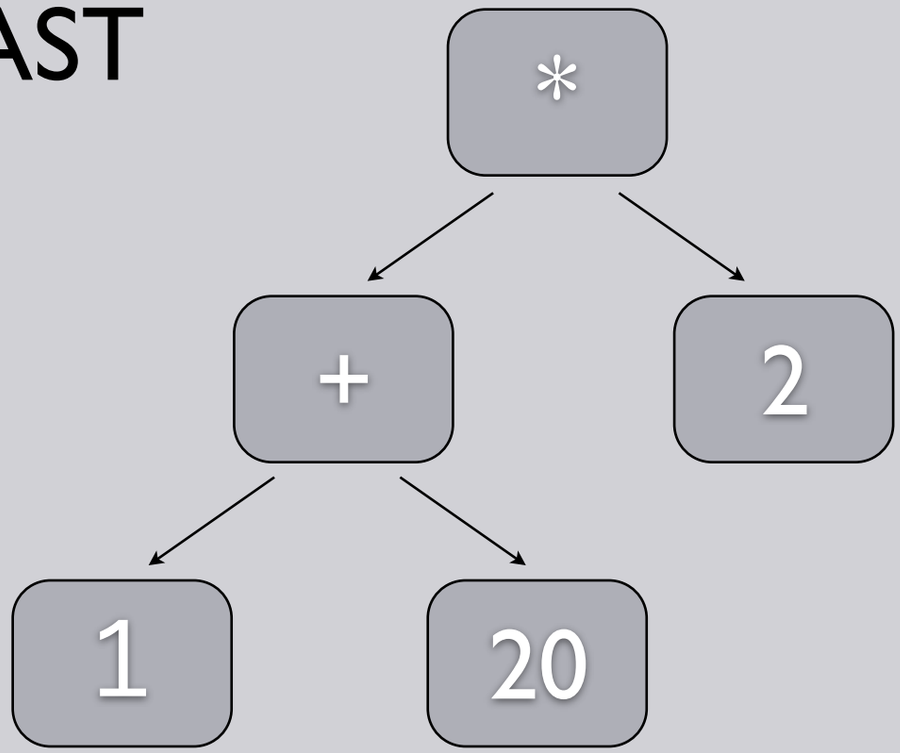
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

Text ( 1 + 20 ) * 2 EOF

Reduce!

AST




Stack

1.Addition

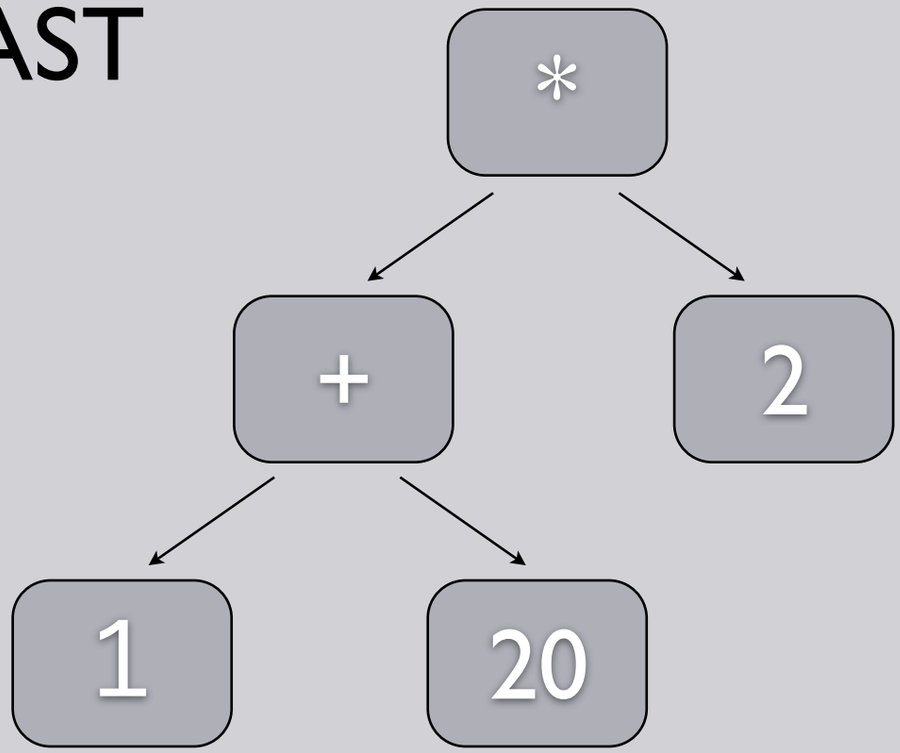
# Grammar

```
Addition returns Expression:
   Multiplication ({AdditionalExpression.left=current} '+' right=Multiplication)*;

Multiplication returns Expression:
   PrimaryExpression ({MultiplyExpression.left=current} '*' right=PrimaryExpression)*;

PrimaryExpression returns Expression:
  {NumberLiteral} value=INT |
  '(' Addition ')';
```

Text ( 1 + 20 ) * 2 EOF

Finished!

AST




Stack