

# CS302 - Data Structures

## *using C++*

Topic: Using the ADT Stack

Kostas Alexis

# Algebraic Expression

- Operator Precedence
  - Parenthesis ( )
  - Exponentiation  $\wedge$  (not a C++ expression)
  - Multiplication and Division  $*$  /
  - Addition and Subtraction  $+$  -

# Algebraic Expression

- Operator Precedence

- Parenthesis ( )
- Exponentiation ^ (not a C++ expression)
- Multiplication and Division \* /
- Addition and Subtraction + -

$$20 - 2 * 2 ^ 3$$

# Algebraic Expression

- Operator Precedence

- Parenthesis ( )
- Exponentiation ^ (not a C++ expression)
- Multiplication and Division \* /
- Addition and Subtraction + -

$$20 - 2 * 8$$

# Algebraic Expression

20 – 16

- Operator Precedence
  - Parenthesis ( )
  - Exponentiation  $\wedge$  (not a C++ expression)
  - Multiplication and Division  $*$  /
  - Addition and Subtraction  $+$  -

# Algebraic Expression

4

- Operator Precedence
  - Parenthesis ( )
  - Exponentiation  $\wedge$  (not a C++ expression)
  - Multiplication and Division  $*$  /
  - Addition and Subtraction  $+$  -
- Binary Operators
  - Require two operands
  - $4 + 5$
- Unary Operators
  - Single operand
  - $-6$

# Algebraic Expression

- Operator Precedence
  - Parenthesis ( )
  - Exponentiation  $\wedge$  (not a C++ expression)
  - Multiplication and Division  $*$  /
  - Addition and Subtraction  $+$  -
- Binary Operators
  - Require two operands
  - $4 + 5$
- Unary Operators
  - Single operand
  - $-6$
- **Infix**
- Common notation
  - $5 + 6$
  - $5 + 6 * 7$
  - $(5 + 6) * 7$
- **Prefix**
- Functional languages
  - $+ 5 6$
  - $+ * 7 6 5$
  - $* + 5 6 7$
- **Postfix**
- Reverse Polish Notation
  - $5 6 +$
  - $5 6 7 * +$
  - $7 5 6 + *$

# Algebraic Expression

- Operator Precedence

- Parenthesis  $()$
- Exponentiation  $\wedge$  (not a C++ expression)
- Multiplication and Division  $* /$
- Addition and Subtraction  $+ -$

- Binary Operators

- Require two operands
- $4 + 5$

- Unary Operators

- Single operand
- $-6$

- **Infix**

- Common notation

$5 + 6$

- $5 + 6$
- $5 + 6 * 7$
- $(5 + 6) * 7$

- **Prefix**

- Functional languages

- $+ 5 6$
- $+ * 7 6 5$
- $* + 5 6 7$

- **Postfix**

- Reverse Polish Notation

- $5 6 +$
- $5 6 7 * +$
- $7 5 6 + *$



# Algebraic Expression

- Operator Precedence

- Parenthesis  $()$
- Exponentiation  $^$  (not a C++ expression)
- Multiplication and Division  $* /$
- Addition and Subtraction  $+ -$

- Binary Operators

- Require two operands
- $4 + 5$

- Unary Operators

- Single operand
- $-6$

- **Infix**

- Common notation

$+ 6$

- $5 + 6$
- $5 + 6 * 7$
- $(5 + 6) * 7$

- **Prefix**

- Functional languages

- $+ 5 6$
- $+ * 7 6 5$
- $* + 5 6 7$

- **Postfix**

- Reverse Polish Notation

- $5 6 +$        $5$
- $5 6 7 * +$
- $7 5 6 + *$

# Algebraic Expression

- Operator Precedence

- Parenthesis  $()$
- Exponentiation  $^$  (not a C++ expression)
- Multiplication and Division  $* /$
- Addition and Subtraction  $+ -$

- Binary Operators

- Require two operands
- $4 + 5$

- Unary Operators

- Single operand
- $-6$

- **Infix**

- Common notation

6

- $5 + 6$
- $5 + 6 * 7$
- $(5 + 6) * 7$

- **Prefix**

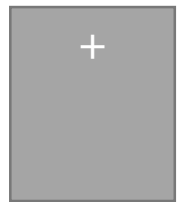
- Functional languages

- $+ 5 6$
- $+ * 7 6 5$
- $* + 5 6 7$

- **Postfix**

- Reverse Polish Notation

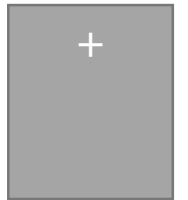
- $5 6 +$       5
- $5 6 7 * +$
- $7 5 6 + *$



operatorStack

# Algebraic Expression

- Operator Precedence
  - Parenthesis ( )
  - Exponentiation  $\wedge$  (not a C++ expression)
  - Multiplication and Division  $*$  /
  - Addition and Subtraction  $+$  -
- Binary Operators
  - Require two operands
  - $4 + 5$
- Unary Operators
  - Single operand
  - $-6$
- **Infix**
- Common notation
  - $5 + 6$
  - $5 + 6 * 7$
  - $(5 + 6) * 7$
- **Prefix**
- Functional languages
  - $+ 5 6$
  - $+ * 7 6 5$
  - $* + 5 6 7$
- **Postfix**
- Reverse Polish Notation
  - $5 6 +$        $5 6$
  - $5 6 7 * +$
  - $7 5 6 + *$



operatorStack

# Algebraic Expression

- Operator Precedence

- Parenthesis  $()$
- Exponentiation  $^$  (not a C++ expression)
- Multiplication and Division  $* /$
- Addition and Subtraction  $+ -$

- Binary Operators

- Require two operands
- $4 + 5$

- Unary Operators

- Single operand
- $-6$

- **Infix**

- Common notation

- $5 + 6$
- $5 + 6 * 7$
- $(5 + 6) * 7$

- **Prefix**

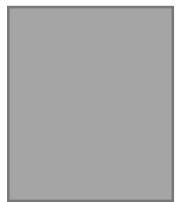
- Functional languages

- $+ 5 6$
- $+ * 7 6 5$
- $* + 5 6 7$

- **Postfix**

- Reverse Polish Notation

- $5 6 +$        $5 6 +$
- $5 6 7 * +$
- $7 5 6 + *$



operatorStack

# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression
- Use stacks to do so

# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix

# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix

infix

$5 * ( 4 + 2 ) ^ 3$

postfix

# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression

infix

5 \* ( 4 + 2 ) ^ 3

postfix

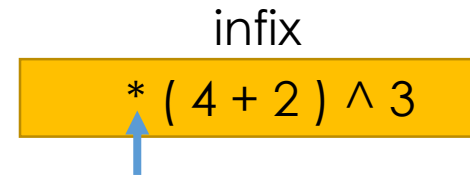


# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - **Pop** operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - **Push** operator onto the operatorStack

infix

\* ( 4 + 2 ) ^ 3



postfix

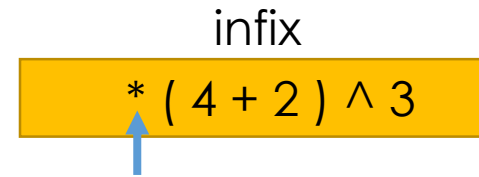
5




# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - **Pop** operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - **Push** operator onto the **operatorStack**

infix  
\* ( 4 + 2 ) ^ 3



postfix  
5



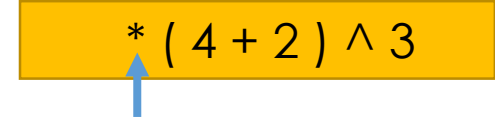
operatorStack

# Evaluating Infix Expressions

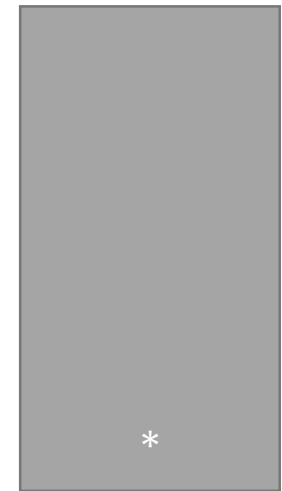
- Converting Infix Expressions to Postfix

- **Operand** – append to the postfix expression
- **Operators** \* / + - etc
  - If **operatorStack** is not empty
    - **Pop** operators and append to the postfix expression
    - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
    - **Push** operator onto the **operatorStack**

infix  
\* ( 4 + 2 ) ^ 3



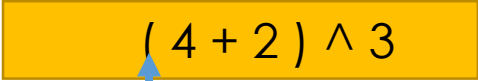
postfix  
5



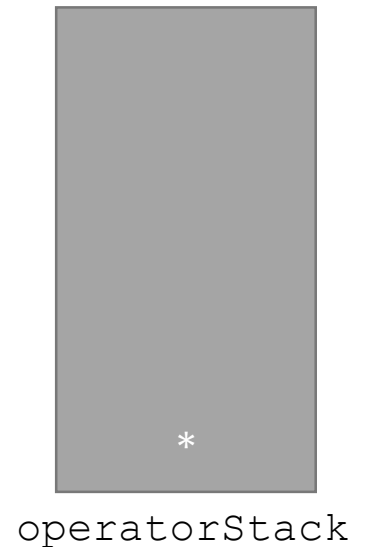
operatorStack

# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - **Pop** operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - **Push** operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix  


postfix  

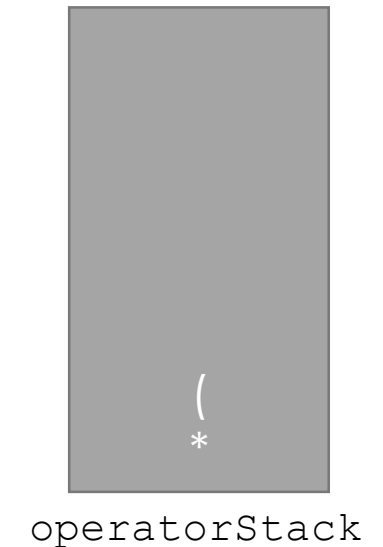



# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix  
4 + 2 ) ^ 3

postfix  
5

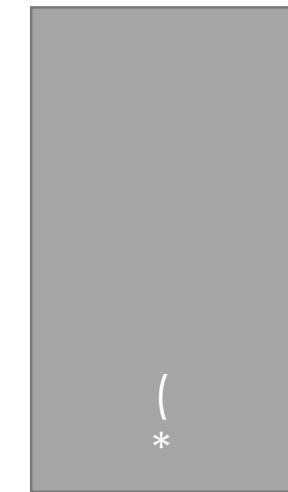


# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix  
4 + 2 ) ^ 3

postfix  
5

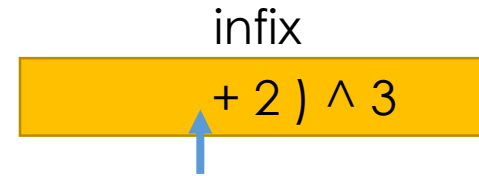


operatorStack

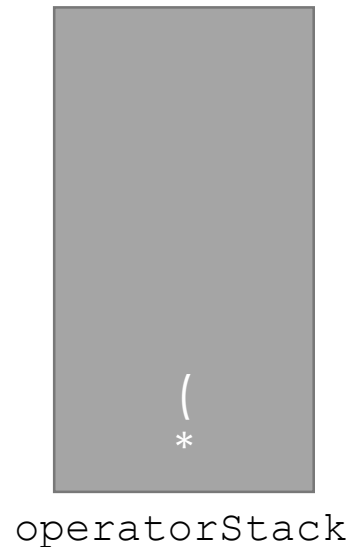
# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix  
+ 2 ) ^ 3



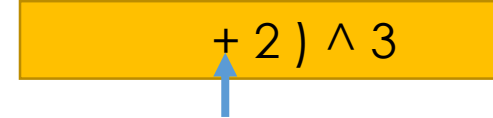
postfix  
5 4



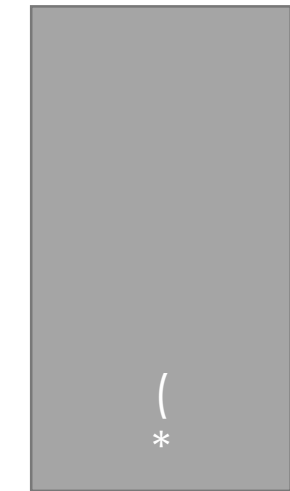
# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix  
+ 2 ) ^ 3



postfix  
5 4



operatorStack

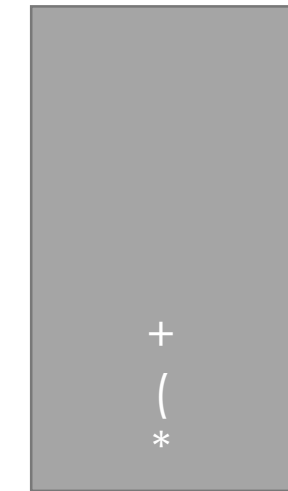


# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix  
2 ) ^ 3

postfix  
5 4



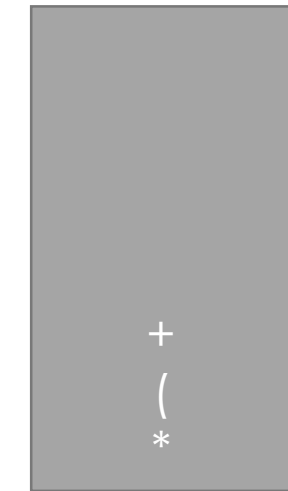
operatorStack

# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix  
2 ) ^ 3

postfix  
5 4




operatorStack

# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix

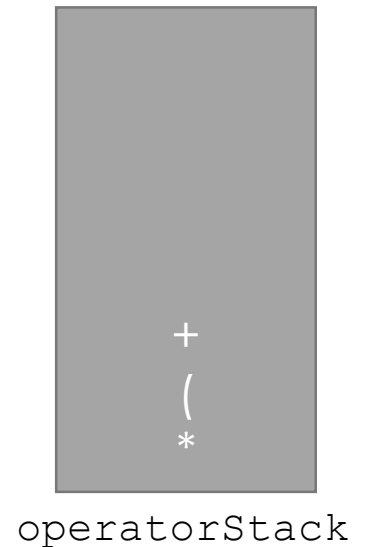


) ^ 3

postfix



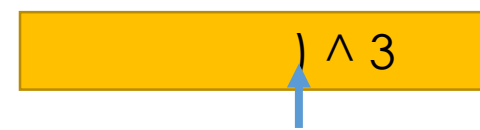
5 4 2



# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
    - ( – push onto **operatorStack**
    - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix

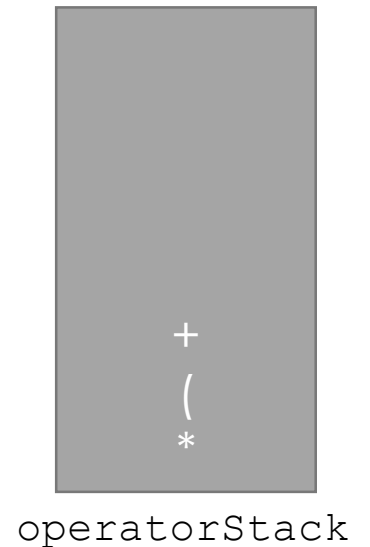


) ^ 3

postfix




5 4 2



# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix

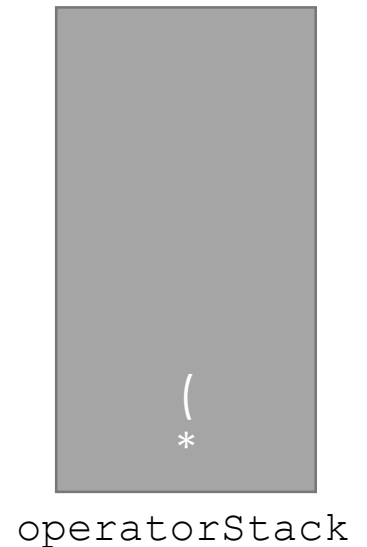


) ^ 3

postfix

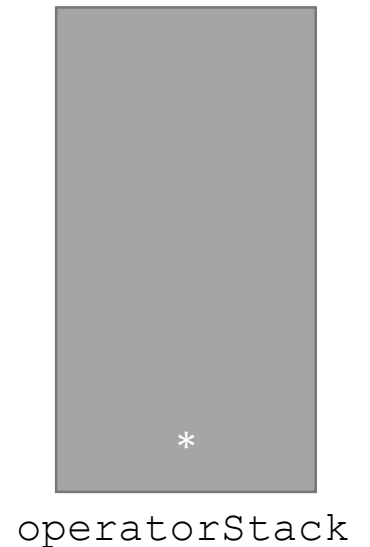
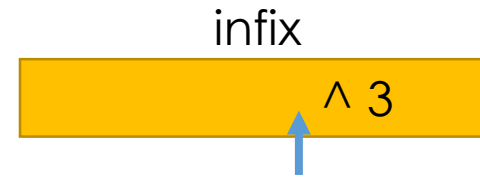


5 4 2 +



# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped



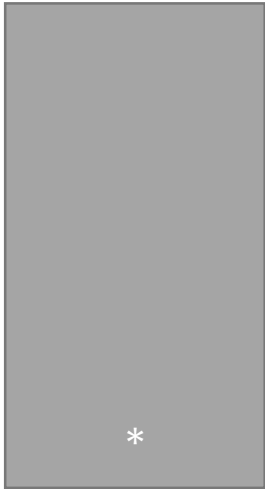
# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix



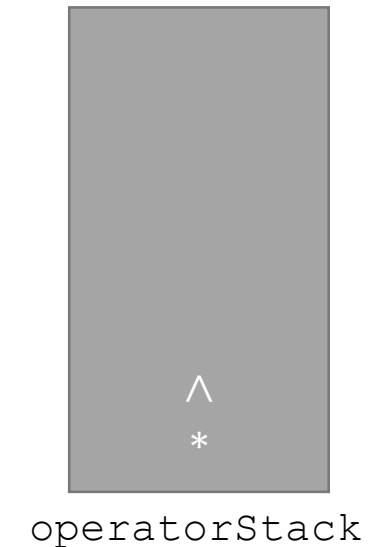
postfix



operatorStack

# Evaluating Infix Expressions

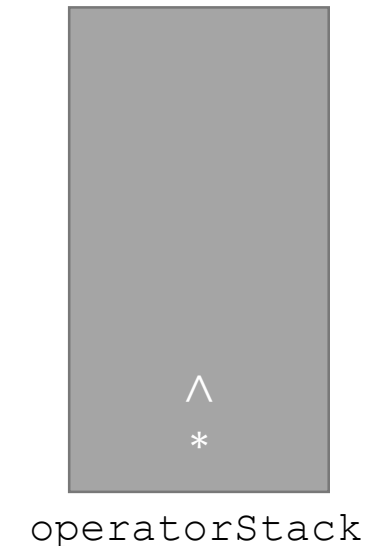
- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped





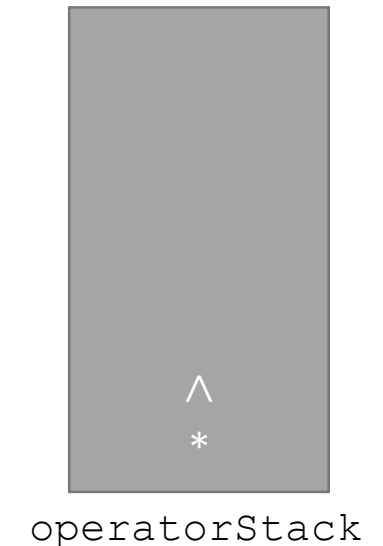
# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped



# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped



# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix



postfix

5 4 2 + 3 ^



operatorStack

# Evaluating Infix Expressions

- Converting Infix Expressions to Postfix
  - **Operand** – append to the postfix expression
  - **Operators** \* / + - etc
    - If **operatorStack** is not empty
      - Pop operators and append to the postfix expression
      - If their precedence  $\geq$  that of the operator in the infix expression until operatorStack is empty or a ( is reached
      - Push operator onto the **operatorStack**
  - ( – push onto **operatorStack**
  - ) – pop operators from **operatorStack** and append to postfix expression until ( is popped

infix



postfix

5 4 2 + 3 ^ \*



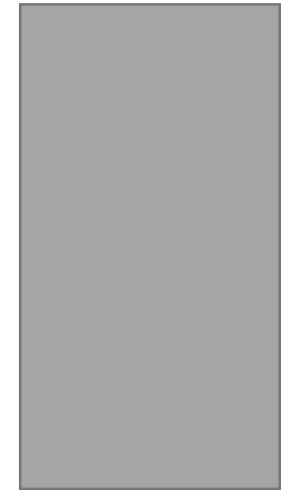
operatorStack

# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the `operandStack`
      - **pop** the operands from the `operandStack`
    - **push** the result of the operation onto the `operandStack`

postfix

5 4 2 + 3 ^ \*

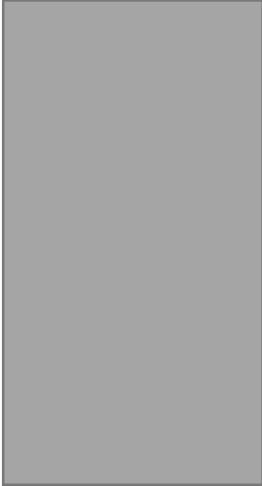


`operandStack`

# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**

postfix  
5 4 2 + 3 ^ \*

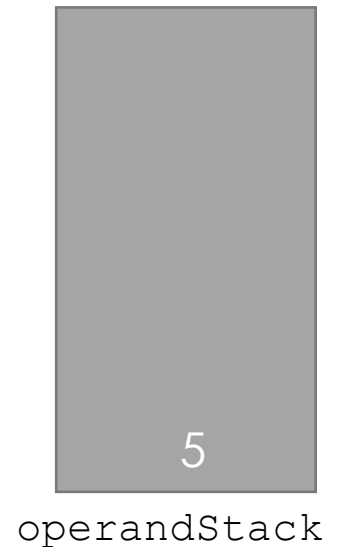


operandStack

# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**

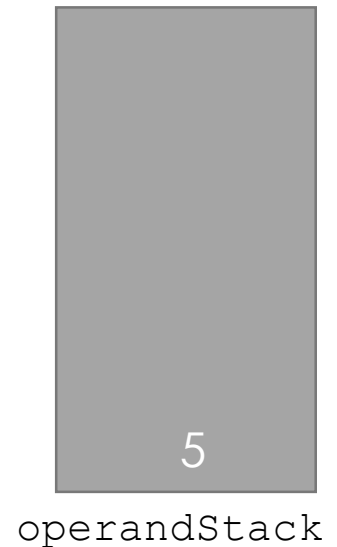
postfix  
4 2 + 3 ^ \*



# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**

postfix  
4 2 + 3 ^ \*

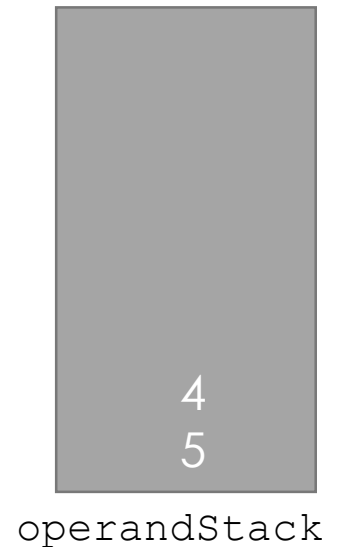




# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**

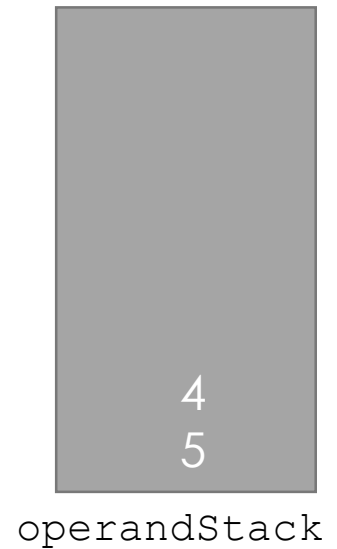
postfix  
2 + 3 ^ \*



# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**

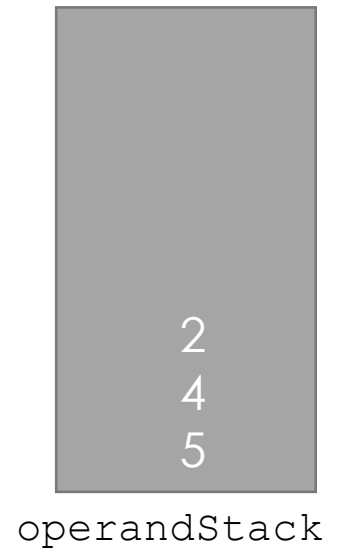
postfix  
2 + 3 ^ \*



# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the `operandStack`
      - **pop** the operands from the `operandStack`
    - **push** the result of the operation onto the `operandStack`

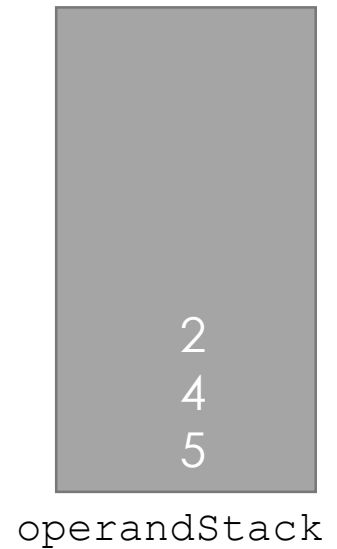
postfix  
+ 3 ^ \*



# Evaluating Postfix Expressions

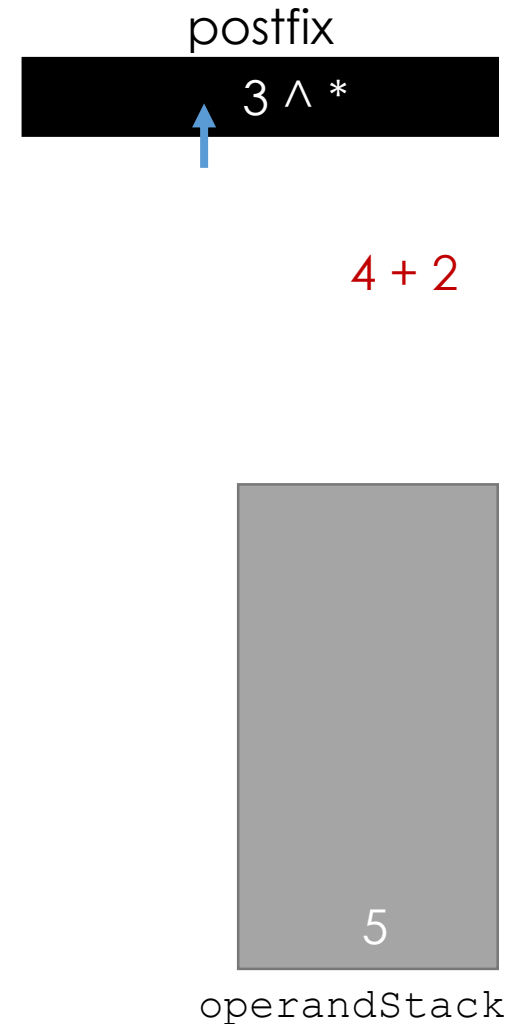
- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**

postfix  
+ 3 ^ \*



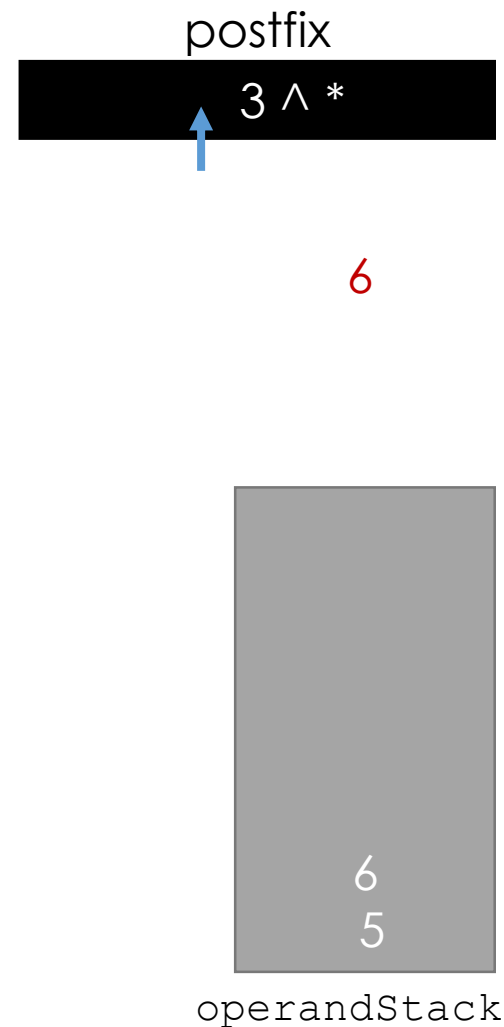
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



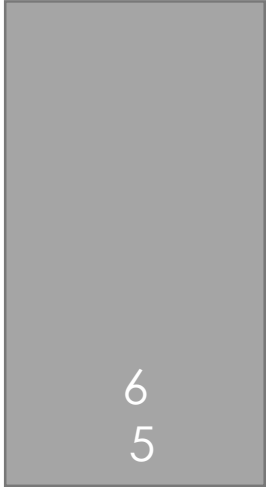
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**

postfix



3 ^ \*

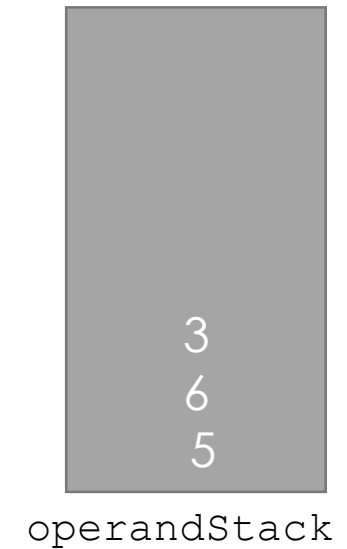


operandStack

6  
5

# Evaluating Postfix Expressions

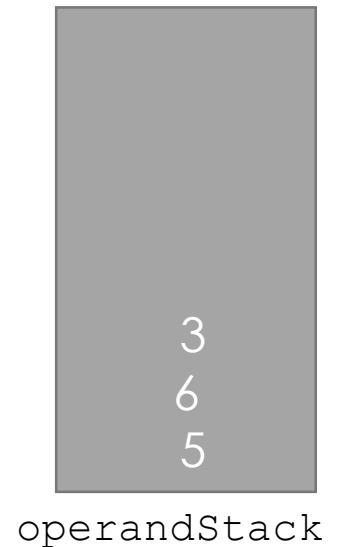
- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**





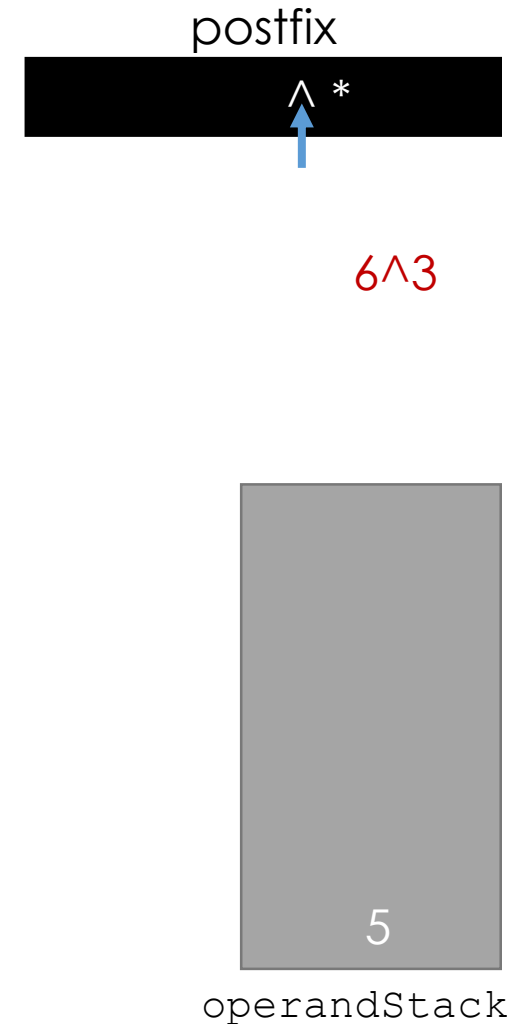
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



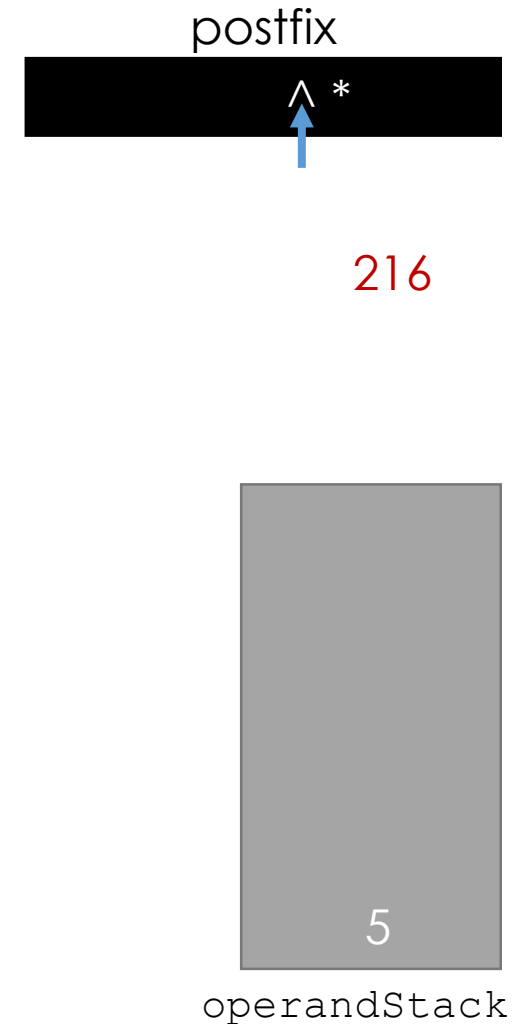
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



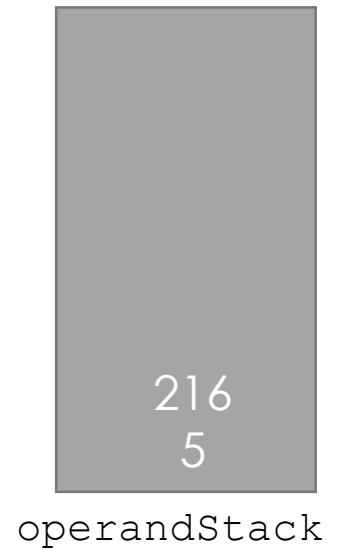
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



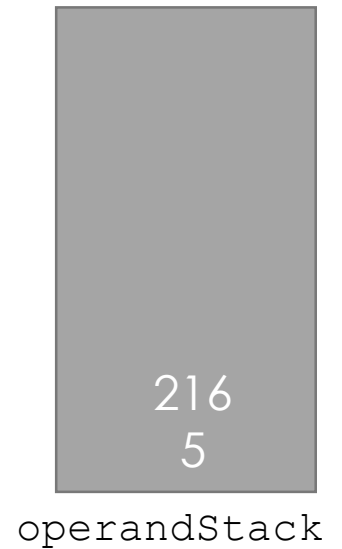
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



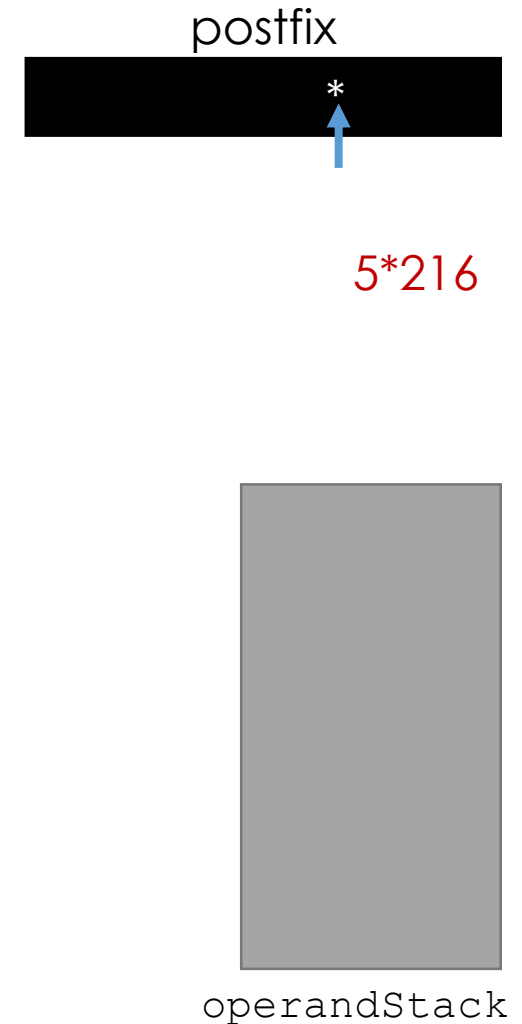
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



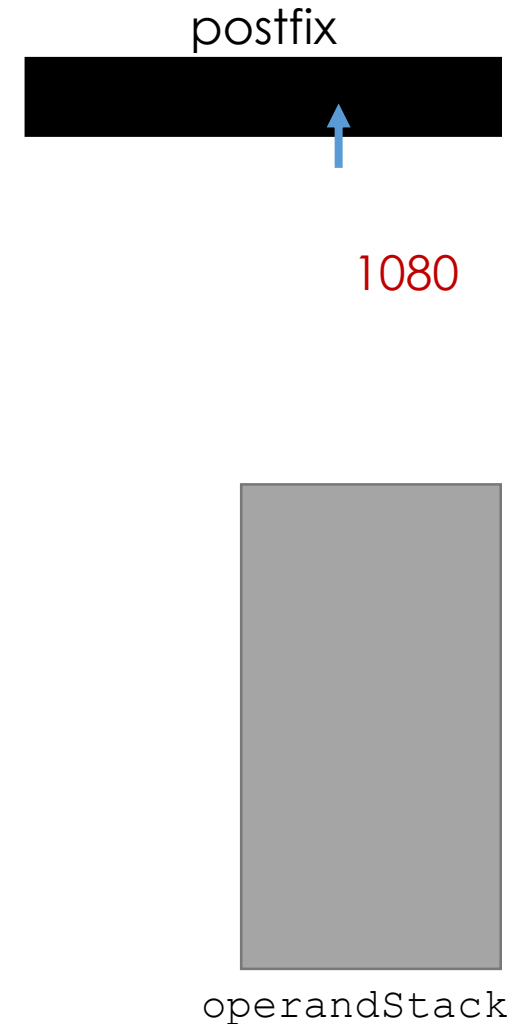
# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



# Evaluating Postfix Expressions

- Scan characters in the postfix expression
  - When an operand is entered
    - **push** it onto the `operandStack`
  - When an operator is entered
    - Apply it to the top two operands of the **operandStack**
      - **pop** the operands from the **operandStack**
    - **push** the result of the operation onto the **operandStack**



# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression

```
Algorithm convertToPostfix(infix)
// Converts an infix expression to an equivalent postfix expression

infixExpression = infix expression to process
operatorStack = a new empty stack
postfixExpression = a new empty string
```



# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression

```
Algorithm convertToPostfix(infix)
// Converts an infix expression to an equivalent postfix expression

infixExpression = infix expression to process
operatorStack = a new empty stack
postfixExpression = a new empty string

while (infixQueue has characters left to parse)
{
    nextCharacter = next non-blank infixExpression character
    process(nextCharacter)
}
while (!operatorStack.isEmpty())
{
    topOperator = operatorStack.pop()
    append topOperator to postfixExpression
}
return postfixExpression
```

# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression

```
// Process nextCharacter algorithm

switch (nextCharacter)
{
    case variable:
        append nextCharacter to postfixExpression
    case '^':
        operatorStack.push(nextCharacter)
    case '(':
        operatorStack.push(nextCharacter)
```

# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression

```
// Process nextCharacter algorithm

switch (nextCharacter)
{
    case variable:
        append nextCharacter to postfixExpression
    case '^':
        operatorStack.push(nextCharacter)
    case '(':
        operatorStack.push(nextCharacter)
    case ')':
        topOperator = operatorStack.pop()
        while(topOperator != '(')
            append topOperator to postfixExpression
        topOperator = operatorStack.pop()
    default:
        break;
}
```

# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression

```
// Process nextCharacter algorithm

switch (nextCharacter)
{
    case variable:
        append nextCharacter to postfixExpression

    case '^':
        operatorStack.push(nextCharacter)

    case '(':
        operatorStack.push(nextCharacter)

    case ')':
        topOperator = operatorStack.pop()
        while(topOperator != '(')
            append topOperator to postfixExpression
            topOperator = operatorStack.pop()

    case '+': case '-': case '*': case '/':
        while(!operatorStack.isEmpty() and precedence of
nextCharacter <= precedence of operatorStack.peek())
            topOperator = operatorStack.pop()
            append topOperator to postfixExpression
            topOperator = operatorStack.pop()
        peratorStack.push(nextCharacter)

    default:
        break;
}
```

# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression

```
Algorithm convertToPostfix(infix)
// Converts an infix expression to an equivalent postfix expression

infixExpression = infix expression to process
operatorStack = a new empty stack
postfixExpression = a new empty string

while (infixQueue has characters left to parse)
{
    nextCharacter = next non-blank infixExpression character
    process(nextCharacter)
}
while (!operatorStack.isEmpty())
{
    topOperator = operatorStack.pop()
    append topOperator to postfixExpression
}
return postfixExpression
```

# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
  - Evaluate the postfix expression

```
Algorithm evaluatePostfix(postfix)
// evaluate a postfix expression
valueStack = a new empty stack
while(postfixExpression has characters left to process)
{
    nextCharacter = next non blank character of postfixExpression
    switch (nextCharacter)
    {
        case variable:
            valueStack.push(value of var nextCharacter)
            break
        case '+': case '-': case '*': case '/': case '^':
            operandTwo = valueStack.peek()
            valueStack.pop()
            operandOne = valueStack.peek()
            valueStack.pop()
            result = apply operation in nextCharacter to its
            operands operandOne and operandTwo
            break
        default: break
    } // end switch
} // end while
```

# Evaluating Infix Expressions

- To evaluate an infix expression
  - Convert the infix expression to postfix form
    - Evaluate the postfix expression

```
Algorithm evaluatePostfix(postfix)
// evaluate a postfix expression
valueStack = a new empty stack
while(postfixExpression has characters left to process)
{
    nextCharacter = next non blank character of postfixExpression
    switch (nextCharacter)
    {
        case variable:
            valueStack.push(value of var nextCharacter)
            break
        case '+': case '-': case '*': case '/': case '^':
            operandTwo = valueStack.peek()
            valueStack.pop()
            operandOne = valueStack.peek()
            valueStack.pop()
            result = apply operation in nextCharacter to its
            operands operandOne and operandTwo
            break
        default: break
    } // end switch
} // end while
return valueStack.peek()
```

**Thank you**