Infix, Prefix, Postfix Expressions

Dr. Thomas E. Hicks
Computer Science Dept
Trinity University
Objectives

1. Evaluate Mathematical Expressions

2. Infix Expression → Operators, Operands, Scope Openers, Scope Closers

3. Mathematical Order Of Operations

4. Generate Postfix From Infix

5. Generate Prefix From Infix

6. Evaluate Infix Expression With Respect To Parentheses, Brackets, & Braces

7. Create Prefix Expression From Infix Expression

8. Evaluate Postfix Expression
UT Health Science Center

Artificial Heart

1 Hr 20 Min Compile Time

Many Fortran Expressions

Only Compile & Test 6 or 7 a Day!
"We Know How To Program Expressions!"

\[ A = B \times C - 1.2 \div (D + E) \]

"We Don’t Know How To Evaluate Expressions!"
Interactive Application

Enter Expression 1: __________________
Enter Expression 2: __________________

Would Not Have To Compile Each Time
Simply Enter The Sets Of Expressions
Have An Algorithm Worked Out Before You Start Coding
Step 1
Valid Infix!
Evaluation of Mathematical Expression

**Step 1 – Is It A Valid Infix**

Is It Valid With Respect To Parentheses, Brackets, & Braces

<table>
<thead>
<tr>
<th>#</th>
<th>Expression</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>(2 + 3) * (5 - 2)</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>3 * (2 + [12 / [4 * {5 - 4} -1 ] - 1])</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

What makes a Mathematical Expression valid with respect to Parentheses, Brackets, and Braces?

<table>
<thead>
<tr>
<th>#</th>
<th>Expression</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3 * )2 + 3 (</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>3 * (2 + 12 / {4 * [5 - 4] -1 ])</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>( 3 + 2 ^ 3 ^ 2 / {4 * [5 - 4] * 1 } * 2</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Now that we can manually verify valid infix parentheses, brackets, and braces, how do we get the computer to do so?
Verify Infix Parentheses, Brackets, & Braces # 0
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

“2 + (3 * 5)”

Define Scope Openers: ( { [ Define Scope Closers: ) } ]

```
Stack <char>
ValidParen(5);
```

Traverse Infix String One Character At A Time

• If the character is a Scope Opener Push It To The Stack
• If the character is a Scope Closer, Pop the Stack; if the Closer and the top element of the stack are a pair, continue; otherwise return Invalid!

• When done, return Valid if stack is empty; otherwise return Invalid!
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

“2 + (3 * 5)”

Stack <char>
ValidParen(5);

What do we do with the 2?

Skip Constants/Operands
Nothing – Skip to next Character
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

“2 + (3 * 5)”

What do we do with the +?

Skip Operators
Nothing – Skip to next Character

Stack <char>
ValidParen(5);

Info
Max
-1
Top 13
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

“2 + (3 * 5)”

What do we do with the (?

Push Scope Openers

Stack <char>

ValidParen(5);
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

“2 + (3 * 5)”

What do we do with the 3?

Skip Constants/Operands
Nothing – Skip to next Character
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
"2 + (3 * 5 )"
```

What do we do with the *?

Skip Operators
Nothing – Skip to next Character

Stack <char>
ValidParen(5);
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

“2 + (3 * 5 )”

Stack <char>
ValidParen(5);

What do we do with the 5?

Skip Constants/Operands
Nothing – Skip to next Character
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
"2 + (3 * 5 )"
```

What do we do with the )? 

Pop The Stack And See If Set ) Goes With ( If the Set Matches Continue Return INVALID If No Match

Stack <char>
ValidParen(5);
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 + (3 * 5)</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

“2 + (3 * 5)"

Done With The Infix String

If Stack Empty Return VALID

If Stack Not Empty Return INVALID

Stack <char>
ValidParen(5);
Verify Infix Parentheses, Brackets, & Braces #1
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

```
“5 + 2 ^ 3 ^ 2”
```

What do we do with the 5?

Skip Constants/Operands

Nothing – Skip to next Character

Stack <char>

ValidParen(5);
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>

```
“5 + 2 ^ 3 ^ 2”

Stack <char>
ValidParen(5);
```

**What do we do with the +?**

**Skip Operators**

Nothing – Skip to next Character
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

```
"5 + 2 ^ 3 ^ 2"
```

Stack <char>
ValidParen(5);

What do we do with the 2?

Skip Constants/Operands
Nothing – Skip to next Character
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

“5 + 2 ^ 3 ^ 2” Stack <char>
ValidParen(5);

What do we do with the ^?

Skip Operators
Nothing – Skip to next Character

^ or $ often represents exponentiation
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression</th>
<th>[Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td>&quot;5 + 2 ^ 3 ^ 2&quot;</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

"5 + 2 ^ 3 ^ 2"

What do we do with the 3?

Skip Constants/Operands
Nothing – Skip to next Character

Stack <char>
ValidParen(5);
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(5 + 2 \wedge 3 \wedge 2)</td>
<td>(\times)</td>
<td></td>
</tr>
</tbody>
</table>

What do we do with the \(^\wedge\)?

Skip Operators

Nothing – Skip to next Character
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

```
"5 + 2 ^ 3 ^ 2"
ValidParen(5);
```

What do we do with the 2?

Skip Constants/Operands
Nothing – Skip to next Character
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 + 2 ^ 3 ^ 2</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

```
“2 + 3 * 5 - 2”
```

ValidParen(5);

Infix Expression Complete

What Is Returned?

VALID – Stack is Empty!
Verify Infix Parentheses, Brackets, & Braces #2
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(2 + 3) * (5 - 2)</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

```
“(2 + 3) * (5 - 2)”
```

What do we do with the (?

Push Scope Openers

Stack <char>

ValidParen(5);
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(2 + 3) * (5 - 2)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
“(2 + 3) * (5 - 2)”
```

Stack <char>
ValidParen(5);

```
Stack:
( 5
Info:
5
Max:
0
Top:
31
```

Skip
Pop Scope Closer
Continue
Set Match ()
Skip
Skip
Skip
Skip
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(2 + 3) * (5 - 2)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
“(2 + 3) * (5 - 2)”
```

Stack <char>
ValidParen(5);

```
Push
```

Skip
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(2 + 3) * (5 – 2)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
“(2 + 3) * (5 – 2)”
```

Stack <char>
ValidParen(5);

```
Stack Info
5
4
3
2
1
0

( )

Set Match ()
Continue

Skip
Skip
Skip

Pop
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(2 + 3) * (5 – 2)</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

“(2 + 3) * (5 – 2)”

ValidParen(5);

Infix Expression Complete

What Is Returned?

VALID – Stack is Empty!
Verify Infix Parentheses, Brackets, & Braces #3
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 * (2 + [ 12 / [ 4 * {5 − 4} -1 ] -1 ] )</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

“3 *(2 + [ 12 / [ 4 * {5 − 4} -1 ] −1 ] )”
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 * (2 + [ 12 / [ 4 * {5 – 4} -1 ] – 1 ] )</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

“3 * (2 + [ 12 / [ 4 * {5 – 4} -1 ] – 1 ] )”

Skip
Skip
Push
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$3 \times (2 + [12 / [ 4 * {5 - 4} -1 ] - 1 ] )$</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

“$3 \times (2 + [12 / [ 4 * \{5 - 4\} -1 ] - 1 ] )$”

![Diagram showing the evaluation of the expression with push and skip actions]
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 * (2 + [ 12 / [ 4 * {5 – 4} -1 ] -1 ] )</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

"3 * (2 + [ 12 / [ 4 * {5 – 4} -1] -1 ] )"
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 * (2 + [ 12 / [ 4 * {5 – 4} -1 ] -1 ] )</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

“3 * (2 + [ 12 / [ 4 * {5 – 4} -1 ] -1 ] )"
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 * (2 + [ 12 / [ 4 * {5 - 4} -1 ] - 1 ] )</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$3 \times (2 + [ 12 / [ 4 \times { 5 - 4 } - 1 ] - 1 ] )$</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

“$3 \times (2 + [ 12 / [ 4 \times \{ 5 - 4 \} - 1 ] - 1 ] )$”
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$3 \times (2 + \left[ \frac{12}{\left[ 4 \times {5 - 4} - 1 \right] - 1} \right] - 1)$</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

“$3 \times (2 + \left[ \frac{12}{\left[ 4 \times \{5 - 4\} - 1 \right] - 1} \right] - 1)$”

Set Match ()
Continue

Pop

Max

Top

Info

5

0

43
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 * (2 + [ 12 / [ 4 * {5 – 4} -1 ] - 1 ] )</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

"3 * (2 + [ 12 / [ 4 * { 5 – 4 } - 1 ] - 1 ] )"

Infix Expression Complete

What Is Returned?

VALID – Stack is Empty!
Verify Infix Parentheses, Brackets, & Braces #4
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3 * ) 2 + 3 (</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3 * (2 + 12 / {4 * [5 – 4} -1 ]))</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

"3 * ( 2 + 12 / { 4 * [ 5 – 4 } -1 ] )"
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3 * (2 + 12 / {4 * [5 - 4] -1 ])</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

"3 * ( 2 + 12 / { 4 * [ 5 – 4 } -1 ] )"
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3 * ( 2 + 12 / { 4 * [ 5 – 4 } -1 ] )</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

“3 * ( 2 + 12 / { 4 * [ 5 – 4 } -1 ] )"
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>3 * ( 2 + 12 / { 4 * [ 5 - 4 } -1 ] )</code></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

“`3 * ( 2 + 12 / { 4 * [ 5 - 4 } -1 ] )`”

[ ] Not Match
Return INVALID

Skip Skip Skip Pop
Verify Infix Parentheses, Brackets, & Braces #6
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>(( 3 + 2 \uparrow 3 \uparrow 2 / { 4 * [5 - 4] * 1 } * 2)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
“( 3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } * 2”
```

```
5 |
4 |
3 |
2 |
1 |
0 |
   Info
    5
    Max
    -1
    Top 53
```
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>( 3 + 2 ^ 3 ^ 2 / {4 * [5 – 4] -1 } * 2</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>( 3 + 2 ^ 3 ^ 2 / {4 * [5 - 4] -1 } * 2</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
"( 3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] *1 } ) * 2"
```
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>( 3 + 2 ^ 3 ^ 2 / {4 * [5 - 4] -1 } * 2</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>( 3 + 2 ^ 3 ^ 2 / {4 * [5 - 4] -1 } * 2</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

```
“( 3 + 2 ^ 3 ^ 2 / {4 * [5 - 4] -1 } * 1 } * 2”
```
We are going to evaluate expressions with constants – Extrapolate it to variables!

<table>
<thead>
<tr>
<th>#</th>
<th>Expression [Infix String]</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>( 3 + 2 ^ 3 ^ 2 / {4 * [5 - 4] -1 } * 2</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

If Stack Empty Return VALID

If Stack Not Empty Return INVALID

"( 3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } * 2"
What type of stack was to Verify A Valid Infix?
Is It Correct Parentheses, Brackets, and Braces?

Stack <char>
ValidParen(5);
Step 2
Create Postfix From Infix
To Manually Form Prefix & Postfix From Valid Infix Must Know Order Operations
Must Get Same Answer As Manual Solution!

Preserve The Mathematical Order Of Operations

\[5 + 2 \times 3 \times 2 = \ ?\]

\[\text{Other} \ ?\]

\[69 \ ?\]

\[517 \ ?\]

Have An Algorithm Worked Out Before You Start Coding
Maybe A Spreadsheet Will Help?

\[ 5 + 2^3^2 = \boxed{517} \]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Excel Evaluation Of Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^4)</td>
<td>16</td>
</tr>
<tr>
<td>(5 + 2^3^2)</td>
<td>Maybe Not!</td>
</tr>
<tr>
<td>(5 + (2^3)^2)</td>
<td>69</td>
</tr>
<tr>
<td>(5 + 2^{(3^2)})</td>
<td>517</td>
</tr>
</tbody>
</table>

69 ?

517 ?
Create Valid Postfix Expression

Step 3

Preserve The Mathematical Order Of Operations

**# 1 Innermost First**
Parenthes, Brackets, Braces

# 4 Left To Right

**# 2 Right To Left**

**# 3 Left To Right**

**# 4 Left To Right**
## Create Valid Postfix Expression  **Step 2**

Preserve The Mathematical Order Of Operations

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5 + 2 ^ 3 ^ 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( 2 + 3 ) * (5 – 2)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 * 2 + 3 – 4 / 2 / 2 - 3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( 3 + 2 ^ 3 ^ 2 / {4 * [ 5 – 4 ] –1 } ) * 2</td>
<td></td>
</tr>
</tbody>
</table>

---

### Create Postfix By Hand – Must Be Able To Do So To Verify Correctness Of Algorithm

<table>
<thead>
<tr>
<th>Infix</th>
<th>5 + 2 ^ 3 ^ 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>5 2 3 2</td>
</tr>
<tr>
<td>Postfix</td>
<td>5 2 3 2</td>
</tr>
</tbody>
</table>
Manual
Form Prefix & Postfix
From Valid Infix #1
<table>
<thead>
<tr>
<th>Infix</th>
<th>5 + 2 ^ 3 ^ 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>5 2 3 2</td>
</tr>
<tr>
<td>Postfix</td>
<td>5 2 3 2</td>
</tr>
</tbody>
</table>
2 – Solve As If Doing Mathematically – Order Of Operations

Infix: \(5 + 2^3^2\)

Prefix: \(5 2 ^3 2\)

Postfix: \(5 2 ^3 2^\)

Put Operator In Front Of Respective Operands For Prefix

Put Operator In Rear Of Respective Operands For Prefix
### 2 – Solve As If Doing Mathematically – Order Of Operations

<table>
<thead>
<tr>
<th>Infix</th>
<th>Prefix</th>
<th>Postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 + 2 ^ 3 ^ 2</td>
<td>5 ^ 2 ^ 3 2</td>
<td>5 + 2 3 2 ^</td>
</tr>
</tbody>
</table>

Infix: 5 + 2 ^ 3 ^ 2
Prefix: 5 ^ 2 ^ 3 2
Postfix: 5 + 2 3 2 ^
### 2 – Solve As If Doing Mathematically – Order Of Operations

<table>
<thead>
<tr>
<th>Infix</th>
<th>5 + ( \boxed{2 ^ 3 ^ 2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>+5 ( ^2 ^3 2 )</td>
</tr>
<tr>
<td>Postfix</td>
<td>5 2 3 2 ( ^{^\dddot^\dddot} ) +</td>
</tr>
</tbody>
</table>
Manual
Form Prefix & Postfix
From Valid Infix #2
Solve As If Doing Mathematically – Order Of Operations

<table>
<thead>
<tr>
<th>Infix</th>
<th>( 2 + 3 ) * ( 5 – 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>+ 2 3 5 2</td>
</tr>
<tr>
<td>Postfix</td>
<td>2 3 + 5 2</td>
</tr>
</tbody>
</table>
### Solve As If Doing Mathematically – Order Of Operations

<table>
<thead>
<tr>
<th>Infix</th>
<th>Prefix</th>
<th>Postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2 + 3 ) * ( 5 - 2 )</td>
<td>+ 2 3 5 2</td>
<td>2 3 + 5 2-</td>
</tr>
</tbody>
</table>

- **Infix** notation, where operators are placed between operands.
- **Prefix** notation, where operators are placed before operands.
- **Postfix** (or Reverse Polish) notation, where operators are placed after operands.
### Solve As If Doing Mathematically – Order Of Operations

**Infix**

\[
(2 + 3) \times (5 - 2)
\]

**Prefix**

\[
*+23-52
\]

**Postfix**

\[
23+52-*
\]

\[
\begin{align*}
+ & 2 & 3 & -5 & 2 \\
* & + & 2 & 3 & -5 & 2 \\
2 & 3 & + & 5 & 2 & -
\end{align*}
\]
Form Prefix & Postfix
From Valid Infix #3
Solve As If Doing Mathematically – Order Of Operations

Infix

\[
\begin{array}{c}
3 \times 2 + 3 - 4 / 2 / 2 - 3 \\
\end{array}
\]

Prefix

\[
\begin{array}{c}
* 3 2 + 3 - 4 / 2 / 2 - 3 \\
\end{array}
\]

Postfix

\[
\begin{array}{c}
3 2* + 3 - 4 / 2 / 2 - 3 \\
\end{array}
\]
### Solve As If Doing Mathematically – Order Of Operations

#### Infix

| 3 * 2  | + 3  | - | 4 / 2 | / 2 | - 3 |

#### Prefix

| * 3 2 | 3 4 2 2 3 |

#### Postfix

| 3 2* 3 4 2 2 3 |
Solve As If Doing Mathematically – Order Of Operations

Infix: \[3 \times 2 + 3 - \frac{4}{2} / 2 - 3\]

Prefix: \[* 3 2 \div / 4 2 2 3\]

Postfix: \[3 2* 3 4 2/ 2/ 3\]
Solve As If Doing Mathematically – Order Of Operations

**Infix**

\[ 3 \times 2 + 3 - \frac{4}{2} / 2 - 3 \]

**Prefix**

\[
\begin{array}{cccc}
+ & * & 3 & 2 \\
3 & / / & 4 & 2 & 2 & 3
\end{array}
\]

**Postfix**

\[
\begin{array}{cccc}
3 & 2 & * & 3+ \\
4 & 2 & / & 2 & / & 3
\end{array}
\]
Solve As If Doing Mathematically – Order Of Operations

**Infix**

\[ 3 \times 2 + 3 - \frac{4}{2} / 2 = -3 \]

**Prefix**

\[- + \times 3 2 3 // 4 2 2 3 \]

**Postfix**

\[3 2 \times 3 + 4 2 / 2 / 3 \]
Solve As If Doing Mathematically – Order Of Operations

Infix

3 * 2 + 3 – 4 / 2 / 2

Prefix

– – +* 3 2 3 //4 2 2 3

Postfix

3 2* 3+ 4 2/ 2/- 3
Manual
Form Prefix & Postfix
From Valid Infix #4
Solve As If Doing Mathematically – Order Of Operations

Infix: $\left( 3 + 2 ^ 3 ^ 2 / \{ 4 * \left[ 5 - 4 \right] * 1 \} \right) * 2$

Prefix: $3 2 3 2 4 5 4 1 2$

Postfix: $3 2 3 2 4 5 4 1 2$
Solve As If Doing Mathematically – Order Of Operations

Infix: \(( 3 + 2 ^ 3 ^ 2 / \{ 4 \ast [ 5 - 4 ] \ast 1 \} ) \ast 2\)

Prefix: \(3\ 2\ 3\ 2\ 4\ -5\ 4\ 1\ 2\)

Postfix: \(3\ 2\ 3\ 2\ 4\ 5\ 4-\ast\ 1\ 2\)
Solve As If Doing Mathematically – Order Of Operations

Infix: \((3 + 2^3^2 / \{4 * [5 - 4] * 1\}) * 2\)

Prefix: 3 2 3 2 **4 -5 4 1 2

Postfix: 3 2 3 2 4 5 4-* 1* 2
**Solve As If Doing Mathematically – Order Of Operations**

Infix: \( \left( 3 + 2 \right)^{3^2} / \left\{ 4 \times \left[ 5 - 4 \right] \times 1 \right\} \times 2 \)

<table>
<thead>
<tr>
<th>Infix</th>
<th>Prefix</th>
<th>Postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \left( 3 + 2 \right)^{3^2} / \left{ 4 \times \left[ 5 - 4 \right] \times 1 \right} \times 2 )</td>
<td>(3 \ 2 \ \wedge3 \ 2 \ **4 \ -5 \ 4 \ 1 \ 2)</td>
<td>(3 \ 2 \ \wedge3 \ 2 \ **4 \ -5 \ 4 \ 1 \ 2)</td>
</tr>
</tbody>
</table>

3 2 3 2 **4 -5 4 1 2
Solve As If Doing Mathematically – Order Of Operations

Infix: \((3 + 2 \ ^3 \ ^2 / \ {4 \ * \ [5 - 4] \ * \ 1}) \ * \ 2\)

<table>
<thead>
<tr>
<th>Infix</th>
<th>Prefix</th>
<th>Postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 + 2 ^3 ^2 / {4 * [5 - 4] * 1} * 2</td>
<td>3 ^2 ^3 ^2 * *4 -5 4 1 2</td>
<td>3 2 3 ^2 ^3 2 ^ * *4 -5 4 1 2</td>
</tr>
<tr>
<td>3 + 2 ^3 ^2 / {4 * [5 - 4] * 1} * 2</td>
<td>3 ^2 ^3 ^2 * *4 -5 4 1 2</td>
<td>3 2 3 ^2 ^3 2 ^ * *4 -5 4 1 2</td>
</tr>
<tr>
<td>3 + 2 ^3 ^2 / {4 * [5 - 4] * 1} * 2</td>
<td>3 ^2 ^3 ^2 * *4 -5 4 1 2</td>
<td>3 2 3 ^2 ^3 2 ^ * *4 -5 4 1 2</td>
</tr>
</tbody>
</table>

87
Solve As If Doing Mathematically – Order Of Operations

Infix: \(( 3 + \boxed{2 \ ^3 \ ^2} \ / \ \boxed{4 \ * \ [5 \ - \ 4] \ * \ 1} ) \ * \ 2\)

Prefix: 

\[
\begin{array}{c}
3 \\
/^{^2} \\
/^{^3} \\
2 \\
^{**4} \\
-5 \\
4 \\
1 \\
2
\end{array}
\]

Postfix: 

\[
\begin{array}{c}
3 \\
2 \\
3 \\
2^{^\^} \\
4 \\
5 \\
4-^{*} \\
1^{*} \\
2
\end{array}
\]
### Infix Form

\[( 3 + \frac{2^3}{2} ) \times 2 \]

### Prefix Form

\[+3 \div 2^3 \times 2 \times 4 - 5 \times 4 \div 2\]

### Postfix Form

\[3 \ 2 \ 3 \ \ 2^{^3} \ 4 \ 5 \ 4 \times 1 \times / \]

**Note:** The prefix and postfix forms are used to represent the order of operations. Each form rearranges the operators and operands differently to simplify the calculation process.
Solve As If Doing Mathematically – Order Of Operations

Infix: \((3 + 2^3^2 / \{4 * [5 - 4] * 1\}) * 2\)

Prefix: \(*+3/\^2\^3\ 2\ **4\ -5\ 4\ 1\ 2\)

Postfix: \(3\ 2\ 3\ 2^\^\ 4\ 5\ 4-*\ 1*+/\ 2^\)

Result: 90
Now that we can manually create the postfix expressions, how do we do this with a computer program?
Step 2
Computer Algorithm
Postfix Calculator

Postfix Calculator. Stack: Accumulator: DPS. This calculator uses postfix notation. To use the calculator your browser requires JavaScript support. ...

MIT OpenCourseWare | Civil and Environmental Engineering | 1.00 ...
Finite State Machines, Postfix Calculator. Lecture 24 PDF. Source Code. All code produced with Java™ software. Java™ is a trademark of Sun Microsystems Inc. ...

Postfix calculator ...
... Postfix calculator. Write a program that prompts the user for a mathematical expression in postfix and that evaluates the expression and prints the result. ...

CS2 Assignment 4. Postfix Calculator
Form Postfix
From Valid Infix #2
To Form Postfix
• Examine one Infix character at a time – left to right
  • If Operand, Write it to the Postfix String
  • If Scope Opener or Operator
    • If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    • If ICP < ISP Pop – Write Values To Postfix Until Can Push
    • If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
• Empty Stack – Write Values To Postfix

2 3 + 5 2 - *

( 2 + 3 ) * ( 5 - 2 )

( ICP = 7 \Rightarrow Always PUSH opener to stack)

Stack <char>
Postfix(5);

( ISP of Empty Stack = -1)
To Form Postfix
• Examine one Infix character at a time – left to right
  • If Operand, Write it to the Postfix String
  • If Scope Opener or Operator
    • If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    • If ICP < ISP Pop – Write Values To Postfix Until Can Push
    • If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
• Empty Stack – Write Values To Postfix

2 3 + 5 2 - *

( 2 + 3 ) * ( 5 - 2 )

Write Operand To Postfix String

Postfix = 2

( ISP = That Of Top Char = 0
To Form Postfix

- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

<table>
<thead>
<tr>
<th>Operator</th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>(((</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>)))</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

\[ (2 + 3) \times (5 - 2) \]

ICP = 1 > ISP = 0 \(\Rightarrow\) Push Operator

Postfix = 2

 ISP = That Of Top Char = 0
To Form Postfix:
- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

<table>
<thead>
<tr>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
</tr>
<tr>
<td>{{[</td>
<td>7</td>
</tr>
<tr>
<td>]]</td>
<td>-</td>
</tr>
</tbody>
</table>

2 3 + 5 2 - *

( 2 + 3 ) * ( 5 - 2 )

Write Operand To Postfix String

Postfix = 2 3

( ISP = That Of Top Char = 2
To Form Postfix
• Examine one Infix character at a time – left to right
  • If Operand, Write it to the Postfix String
  • If Scope Opener or Operator
  • If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
  • If ICP < ISP Pop – Write Values To Postfix Until Can Push
  • If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
• Empty Stack – Write Values To Postfix

\[
\begin{align*}
2 & \ 3 & + & 5 & 2 & - & * \\
\end{align*}
\]

\[
(2 + 3) \times (5 - 2)
\]

Scope Closer – Pop & Add To Postfix

Postfix = 2 3 +

\[
(\text{ISP} = \text{That Of Top Char} = 2)
\]
To Form Postfix
- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

2 3 + 5 2 - *

\[(2 + 3) \times (5 - 2)\]

ICP = 3 > ISP = -1 \(\rightarrow\) Push

Postfix = 2 3 +
To Form Postfix

• Examine one Infix character at a time – left to right
  • If Operand, Write it to the Postfix String
  • If Scope Opener or Operator
    • If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    • If ICP < ISP Pop – Write Values To Postfix Until Can Push
    • If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
• Empty Stack – Write Values To Postfix

2 3 + 5 2 - *

( 2 + 3 ) * ( 5 – 2 )

( ICP = 7 \rightarrow Always PUSH opener to stack)

Postfix = 2 3 +
To Form Postfix
- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

2 3 + 5 2 - *

( 2 + 3 ) * ( 5 - 2 )

Write Operand To Postfix String

Postfix = 2 3 + 5
To Form Postfix

- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

**Examples:**

- **Expression 1:** 
  
  \[(2 + 3) \times (5 - 2)\]

  **Steps:**
  - ICP = 1 > ISP = 0 → Push
  - Postfix = 2 3 + 5

- **Expression 2:**
  
  \[(5 - 1) \div (3 + 3)\]

  **Steps:**
  - ICP = 2 > ISP = 3 → Push
  - Postfix = 5 1 - 3 3 +
To Form Postfix

- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

<table>
<thead>
<tr>
<th>Operator</th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>*, /</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>{, [</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>}}, ]</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

2 3 + 5 2 - *

\[ (2 + 3) \times (5 - 2) \]

Write Operand To Postfix String

Postfix = 2 3 + 5 2
To Form Postfix
- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

```
2 3 + 5 2 - *
```

```
( 2 + 3 ) * ( 5 - 2 )
```

Scope Closer – Pop & Add To Postfix

```
Postfix = 2 3 + 5 2 -
```
To Form Postfix

- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

<table>
<thead>
<tr>
<th>Operator</th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>[[</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>]]</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

2 3 + 5 2 - *

(2 + 3) * (5 - 2)

Done – Pop Stack & Add To Postfix

Postfix = 2 3 + 5 2 - *
To Form Postfix

- Examine one Infix character at a time – left to right
  - If Operand, Write it to the Postfix String
  - If Scope Opener or Operator
    - If the In-Coming Priority (ICP) is greater than the In-Stack Priority (ISP) Push
    - If ICP < ISP Pop – Write Values To Postfix Until Can Push
    - If Scope Closer, Pop till find opener – Write values to Postfix – Throw out scope opener!
- Empty Stack – Write Values To Postfix

<table>
<thead>
<tr>
<th></th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>( [</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>] )</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

2 3 + 5 2 - *

( 2 + 3 ) * ( 5 - 2 )

It is finished – Success! – Stack Is Empty!

Postfix = 2 3 + 5 2 - *

( ISP 2

Info

Max

Top

5

4

3

2

1

0

-1

107
Form Prefix & Postfix
From Valid Infix #1
Infix = \(5 + 2 \wedge 3 \wedge 2\)

Write Operand To Postfix String

Postfix = 5

ISP = -1
Infix = 5 + 2 ^ 3 ^ 2

ICP = 1 > ISP -1 ➔ PUSH

Postfix = 5
Infix = 5 + 2 ^ 3 ^ 2

Write Operand To Postfix String

Postfix = 5 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>*</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>/</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>^</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>(</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>)</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>]</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

ISP = 1

Max Top

Info

5
Infix = 5 + 2 \wedge^3 \wedge^2

ICP = 6 > ISP 2 \Rightarrow \text{PUSH}

Postfix = 5 2

<table>
<thead>
<tr>
<th></th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>\wedge</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>([ [</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>]] ]</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

ISP = 2
Infix = 5 + 2 \^3 \^2

Write Operand To Postfix String

Postfix = 5 2 3

ISP = 5
Infix = 5 + 2 ^ 3 ^ 2

ICP = 6 > ISP 5 → Push

Postfix = 5 2 3
Infix = 5 + 2 ^ 3 ^ 2

Write Operand To Postfix String

Postfix = 5 2 3 2

ISP = 5
Infix = 5 + 2 ^ 3 ^ 2

Postfix = 5 2 3 2 ^ ^ +
Form Prefix & Postfix
From Valid Infix #3
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

Write Operand To Postfix String

Postfix = 3

ISP = -1
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

ICP = 3 > ISP = -1 → Push *

Postfix = 3

ISP = -1
Infix = \(3 * 2 + 3 - 4 / 2 / 2 - 3\)

Write Operand To Postfix String

Postfix = 3 2

ISP = 4
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

ICP = 1 < ISP = 4 → Pop * Add To Postfix & Try Again! Push +

Postfix = 3 2 *

ISP = 4
\[ \text{Infix} = 3 \times 2 + 3 - 4 / 2 / 2 - 3 \]

Write Operand To Postfix String

\[ \text{Postfix} = 3 \ 2 \times \ 3 \]

ISP = 2
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

ICP = 1 < ISP = 2 → Pop + Add To Postfix & Try Again! Push -

Postfix = 3 2 * 3 +
Infix $= 3 \times 2 + 3 - \frac{4}{2} / 2 - 3$

Write Operand To Postfix String

Postfix $= 3 \ 2 \times 3 + 4$

ISP $= 2$
Infix = $3 \times 2 + 3 - 4 / 2 / 2 - 3$

ICP = 3 > ISP = 2 $\Rightarrow$ Push $/$

Postfix = $3 \ 2 \ * \ 3 \ + \ 4$

ISP = 2
Infix = $3 \times 2 + 3 - 4 / 2 / 2 - 3$

Write Operand To Postfix String

Postfix = 3 \ 2 \ * \ 3 \ + \ 4 \ 2$

ISP = 2
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

ICP = 3 < ISP = 4 ➔ Pop / Add To Postfix & Try Again! Push /

Postfix = 3 2 * 3 + 4 2 /
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

Write Operand To Postfix String

Postfix = 3 2 * 3 + 4 2 / 2

ISP = 4
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

ICP = 1 < ISP = 4  ➔  Pop / Add To Postfix & Try Again! Pop ➔ Add To Postfix & Try Again! Push -

Postfix = 3 2 * 3 + 4 2 / 2 / -

<table>
<thead>
<tr>
<th></th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>[[[</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>]]</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

ISP = 4
Infix: $3 \times 2 + 3 - 4 \div 2 \div 2 - 3$

Write Operand To Postfix String

Postfix: $3 \ 2 \ * \ 3 \ + \ 4 \ 2 \ / \ 2 \ / \ - \ 3$

ISP: 2
Infix = 3 * 2 + 3 - 4 / 2 / 2 - 3

Postfix = 3 2 * 3 + 4 2 / 2 / - 3.

ISP = 2
Form Prefix & Postfix
From Valid Infix #4
( 3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

ICP = 7 ➞ Always PUSH opener to stack

Postfix = 

ISP = -1
Write Operand To Postfix String

\[
(3 + 2^3^2 / \{4 \times [5 - 4] \times 1\}) \times 2
\]

Postfix = 3

<table>
<thead>
<tr>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
</tr>
<tr>
<td>(</td>
<td>7</td>
</tr>
<tr>
<td>)</td>
<td>-</td>
</tr>
</tbody>
</table>

ISP = 0
( 3 + 2 \uparrow\uparrow 3 \uparrow 2 / \{ 4 \ast \left[ 5 - 4 \right] \ast 1 \} \ast 2 )

ICP = 1 > ISP = 0 \rightarrow \text{Push +}

Postfix = 3
( 3 + 2 ^ 3 ^ 2 / { 4 * [ 5 − 4 ] * 1 } ) * 2

Write Operand To Postfix String

Postfix = 3 2

ISP = 2
\[
( 3 + 2 ^ 3 ^ 2 / \{ 4 * [ 5 - 4 ] * 1 \} ) * 2
\]

**ICP = 6 > ISP = 2 \rightarrow** Push \(^{\wedge}\)

**Postfix = 3 2**
(3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

Write Operand To Postfix String

Postfix = 3 2 3

ISP = 5
(3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

ICP = 6 > ISP = 5 ➔ Push ^

Postfix = 3 2 3
(3 + 2 ^ 3 ^ 2 / {4 * [5 - 4] * 1}) * 2

Postfix = 3 2 3 2

ISP = 5
\[
(3 + 2 \, ^\wedge \, 3 \, ^\wedge \, 2 \, / \, \{4 \, \times \, [5 \, -\, 4] \, \times \, 1\}) \, \times \, 2
\]

**ICP = 3 < ISP = 5 \Rightarrow Pop \, ^\wedge Add \, To**

**Postfix & Try Again! Pop \, ^\wedge Add \, To**

**Postfix & Try Again! Push /**

**Postfix = 3 \, 2 \, 3 \, 2 \, ^\wedge \, ^\wedge**

<table>
<thead>
<tr>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
</tr>
<tr>
<td>([[])]</td>
<td>-</td>
</tr>
</tbody>
</table>

ISP = 5
(3 + 2 ^ 3 ^ 2 / { 4 * [ 5 – 4 ] * 1 } ) * 2

Postfix = 3 2 3 2 ^ ^

ICP ISP
+ - 1 2
* / 3 4
^ 6 5
([ [ 7 0
)] ] -
Write Operand To Postfix String

Postfix = 3 2 3 2 ^ ^ 4
( 3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

ICP = 3 > ISP = 0 $\Rightarrow$ Push *

Postfix = 3 2 3 2 ^ ^ 4
(3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

Postfix = 3 2 3 2 ^ ^ 4

ICP ISP
+ - 1 2
* / 3 4
^ 6 5
(([[ 7 0 ))) -

ISP = 4
Write Operand To Postfix String

Postfix = 3 2 3 2 ^ ^ 4 5

<table>
<thead>
<tr>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
</tr>
</tbody>
</table>

ISP = 0
(3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

ICP = 1 > ISP = 0 → Push
Always PUSH opener to stack

Postfix = 3 2 3 2 ^ ^ 4 5
\[ (3 + 2^{3}^{2} / \{4 \times [(5 - 4) \times 1]\}) \times 2 \]

**Write Operand To Postfix String**

**Postfix** = 3 2 3 2 ^ ^ 4 5 4
\[(3 + 2 ^ 3 ^ 2 / \{ 4 * [ 5 - 4 ] * 1 \} ) * 2\]

**Pop Stack Till Find Matching Opener → Add - To Postfix**

**Postfix** = 3 2 3 2 ^ ^ 4 5 -

**ISP = 1**
(3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

Postfix = 3 2 3 2 ^ ^ 4 5 - *

ISP = 4
Write Operand To Postfix String

Postfix = 3 2 3 2 ^ ^ 4 5 4 - * 1 * / + 2 *
The given expression is:

\((3 + 2^3^2 / \{4 * [5 - 4] * 1\}) * 2\)

The steps shown in the diagram are:

1. Pop Stack Till Find Matching Opener \(\Rightarrow\) Add - To Postfix

The postfix expression is:

\(3 2 3 2 ^ ^ 4 5 4 - * 1 * / + 2 *\)

The ISP value is:

\(ISP = 2\)
\[(3 + 2 ^ 3 ^ 2 / \{ 4 * [ 5 - 4 ] * 1 \}) * 2\]

**Pop Stack Till Find Matching Opener \(\rightarrow\) Add - To Postfix**

**Postfix** =

\[3 2 3 2 ^ ^ 4 5 4 * 1 * / + 2 *\]

**ISP = 4**

**ISP**

<table>
<thead>
<tr>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1 2</td>
</tr>
<tr>
<td>* /</td>
<td>3 4</td>
</tr>
<tr>
<td>^</td>
<td>6 5</td>
</tr>
<tr>
<td>([[[</td>
<td>7 0</td>
</tr>
<tr>
<td>]]])</td>
<td>-</td>
</tr>
</tbody>
</table>
Expression: ( (3 + 2) ^ 3 ^ 2 / \{ 4 * [ 5 - 4 ] * 1 \} ) * 2

Postfix: 3 2 3 2 ^ ^ 4 5 - * 1 - / +

ISP = -1

ISP Table:

<table>
<thead>
<tr>
<th>Operator</th>
<th>ICP</th>
<th>ISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>* /</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>^</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>( ( ( [ ] ) ) )</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>
Write Operand To Postfix String

Postfix =

3  2  3  2  ^  ^  4  5  4  -  *  1  *  /  +  2  *

ISP = 4
(3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

Postfix =

3 2 3 2 ^ ^ 4 5 - * 1 - / + 2 *
What type of stack was to create the Postfix from the Infix?

Stack <char>
Postfix(5);
Step 3
Evaluate
Postfix Expression
Algorithm? Must Be Able To Manually Evaluate Infix

Order Operations Still Important
Create Valid Postfix Expression

Step 3

Preserve The Mathematical Order Of Operations

Preserve The Mathematical Order Of Operations

- Parenthes, Brackets, Braces
- # 4 Left To Right
- # 1 Innermost First
- # 3 Left To Right
- # 4 Left To Right
- # 2 Right To Left
- # 3 Left To Right
- +
- *
Evaluate Postfix Expression  Step 2
Preserve The Mathematical Order Of Operations

1 | 5 + 2 ^ 3 ^ 2
   5 2 3 2 ^ ^ +
   = ______ 517

2 | ( 2 + 3 ) * (5 - 2)
   2 3 + 5 2 - *
   = ______ 15

3 | 3 * 2 + 3 - 4 / 2 / 2 - 3
   3 2 * 3 + 4 2 / 2 / - 3 -
   = ______ 5

4 | ( 3 + 2 ^ 3 ^ 2 / {4 * [ 5 - 4 ] * 1 } ) * 2
   3 2 3 2 ^ ^ 4 5 4 - * 1 * / + 2 *
   = ______ 262
Evaluate Postfix #1
Stack <double>
Eval (5);

Push Operands To Stack
Stack <double>
Eval (5);

Push Operands To Stack
1 | 5 + 2 ^ 3 ^ 2

5 2 3 2 ^ ^ +

5 2 3 2 ^ ^ +

Push Operands To Stack

Stack <double>
Eval (5);

= 517
1 | 5 + 2 ^ 3 ^ 2

Stack <double>
Eval (5);

Push Operands To Stack

= 517
Stack: <double>

Eval (5);

Operator – Pop Stack Twice

Apply Operator

Evaluate Expression & Push

Push 9
1 | 5 + 2 ^ 3 ^ 2

5 2 3 2 ^ ^ +

Stack <double>
Eval (5);

Operator – Pop Stack Twice

Pop – Evaluate 2 ^ 9
Push 512
1 | 5 + 2 ^ 3 ^ 2

= 517

Stack <double>
Eval (5);

Operator – Pop Stack Twice

Pop – Evaluate 5 + 512
Push 517
1 | 5 + 2 ^ 3 ^ 2

= 517

Stack <double>
Eval (5);

Done – Only Item In Stack Contains Evaluated Expression

5 2 3 2 ^ ^ +
Evaluate Postfix #2
2 \times (2 + 3) = 15
\[(2 + 3) \times (5 - 2)\]

\[= 15\]
2 | ( 2 + 3 ) * (5 – 2)

= 15

2 3 + 5 2 – *

Pop → Evaluate 2 + 3 → Push 5
2 | (2 + 3) * (5 - 2) = 15

2 3 + 5 2 - *

Push

Info
Max
Top
\[(2 + 3) \times (5 - 2)\]

\[= 15\]
Evaluate 5 - 2
Push 3
$2 \mid (2 + 3) \times (5 - 2)$

= 15

Pop – Evaluate $5 \times 3$
Push 15
\[(2 + 3) \times (5 - 2) = 15\]
Evaluate Postfix #3
The expression given is: 3 * 2 + 3 - 4 / 2 / 2 - 3

3 * 2 + 3 + 4 / 2 / - 3 -

3

2 * 3 + 4 2 / 2 / - 3 -

Push

= 5

Info
Max
Top

181
\[3 \times 2 + 3 - 4 / 2 / 2 - 3\]

\[= 5\]
$$3 \mid 3 \times 2 + 3 - 4 / 2 / 2 - 3$$

$$3 \ 2 \times 3 + 4 \ 2 / 2 / - 3 -$$

Pop ➔ Evaluate 3 * 2 ➔ Push 6
3 | 3 * 2 + 3 – 4 / 2 / 2 – 3

= 5
3 | 3 * 2 + 3 - 4 / 2 / 2 - 3

= 5

Pop → Evaluate 6 + 3 → Push 9
3 | 3 * 2 + 3 – 4 / 2 / 2 – 3

= 5

3 2 * 3 + 4 2 / 2 / - 3 -
3 | 3 * 2 + 3 – 4 / 2 / 2 – 3

= 5

3 2 * 3 + 4 2 / 2 / - 3 -

Push

Info
Max
Top

187
3 | 3 * 2 + 3 – 4 / 2 / 2 − 3

= 5

3 2 * 3 + 4 2 / 2 / − 3 −

Pop → Evaluate 4 / 2 → Push 2
3 | 3 * 2 + 3 - 4 / 2 / 2 - 3
= 5

3 2 * 3 + 4 2 / 2 / - 3 -

Push

Max

Top

Info

5

5

2

9
3 | 3 * 2 + 3 - 4 / 2 / 2 - 3

= 5

3 2 * 3 + 4 2 / 2 / - 3 -

Pop → Evaluate 2 / 2 → Push 1
3 | 3 * 2 + 3 - 4 / 2 / 2 - 3

3 2 * 3 + 4 2 / 2 / - 3 -

= 5

Pop → Evaluate 9 - 1 → Push 8
<table>
<thead>
<tr>
<th></th>
<th>3 * 2 + 3 – 4 / 2 / 2 – 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 2 * 3 + 4 2 / 2 / - 3 -</td>
</tr>
</tbody>
</table>

= 5

Push

<table>
<thead>
<tr>
<th>Info</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>8</td>
</tr>
<tr>
<td>Top</td>
<td>0</td>
</tr>
</tbody>
</table>
Evaluate 8 - 3 → Push 5
3 | 3 * 2 + 3 – 4 / 2 / 2 – 3

= 5

3 2 * 3 + 4 2 / 2 / – 3 –

Done – Only Item In Stack Contains Evaluated Expression
Evaluate Postfix #4
\[ 4 \mid (3 + 2 \, ^\, 3 \, ^\, 2 \, / \, \{4 \, \star \, [\, 5 \, - \, 4 \, ] \, \star \, 1 \, \} \, ) \, \star \, 2 \]

\[ = \ 262 \]
\[
4 \mid (3 + 2 ^ 3 ^ 2 / \{4 \ast [5 - 4] \ast 1\}) \ast 2
\]

\[
= 262
\]
4 | ( 3 + 2 ^ 3 ^ 2 / { 4 * [ 5 - 4 ] * 1 } ) * 2

= 262
\[ 4 \mid (3 + 2^3^2) / (4 \times [5 - 4] \times 1) \times 2 = 262 \]

3 2 3 2 \^ \^ 4 5 4 - * 1 * / + 2 *

Pop \rightarrow \text{Evaluate } 3^2 \rightarrow \text{Push } 9
4 \times \left( 3 + 2 \times 3^{2} / \{4 \times [5 - 4 \times 1]\} \right) \times 2 = 262
\[
4 \mid (3 + 2 ^ 3 ^ 2 / \{4 * \lfloor 5 - 4 \rfloor * 1 \} ) * 2
\]

\[
= 262
\]
\[
(3 + 2 \cdot 3^2 \div (4 \cdot (5 - 4) \cdot 1)) \cdot 2 = 262
\]
4  |  ( 3 + 2 ^ 3 ^ 2 / {4 * [ 5 − 4 ] * 1} ) * 2

\[= 262\]

Pop → Evaluate 5 - 4 → Push 1
Evaluate 4 * 1 → Push 4
\[ 4 \mid (3 + 2^{3^2} / \{4 \times \lfloor 5 - 4 \rfloor \times 1 \}) \times 2 \]

\[ = 262 \]
Evaluate 4 * 1 → Push 4
4 \left(3 + 2^3 + 2 \right) \div \left(4 \times \left[5 - 4 \right] \times 1 \right) \times 2 = \boxed{262}

\begin{align*}
3 & \quad 2 & \quad 3 & \quad 2 & \quad ^{\wedge} & \quad 4 & \quad 5 & \quad 4 & \quad - & \quad 1 & \quad * & \quad / & \quad + & \quad 2 & \quad * \\
\end{align*}

Pop \rightarrow \text{Evaluate } \frac{512}{4} \rightarrow \text{Push } 128
Evaluate 3 + 128 → Push 131
\[
(3 + 2^3)^2 / (4 \times [5 - 4] \times 1) \times 2
\]

\[
= 262
\]
\[
4 \times (3 + 2^3^2 / \{4 \times [5 - 4 \times 1] \} \times 2)
\]

\[
= 262
\]

Pop \rightarrow \text{Evaluate } 131 \times 2 \rightarrow \text{Push } 262
The expression is evaluated as follows:

\[ 4 \mid (3 + 2^3)^2 / \{4 \times [5 - 4] \times 1\} \times 2 \]

Result: 262

Done – Only Item In Stack Contains Evaluated Expression
What type of stack was to Evaluate The Postfix?

Stack <double>
Eval(5);
In Summary
Extra Credit Lab
Not A Team - Individual
You will have lots of programming to do this semester, but you may program this and submit it by the first reading day to replace your two lowest quizzes with 100%.

Main Program is to continue to prompt the user for an Infix until the user enters QUIT.
If the infix is Invalid, say so. If it is Valid, Evaluate it and display the evaluated value.

Templated Stacks
This Lab Can Be Easily Expanded To Process Multiple Digit Floating Point Numbers: 1.2 * 5.43

This Lab Can Be Expanded To Process Variables → More Difficult!

Lots of string processing practice!

Right to left string processing will enable you to create algorithm to evaluate prefix!
Software Engineering I

Be able to create an algorithm that you can walk through manually prior to sitting at the computer.
Create and test well designed ADT’s, such as the Stack, that can be used with multiple datatypes in many programs.
Implement & Test one part of a program at a time. Major components for this lab might be to (1) verify the Infix, (2) create the Postfix, and (3) evaluate the Postfix.