# Infix & Postfix Notation	start time:

Computing mathematical values is one of the most basic operations of a computer, but can be a difficult problem as the expressions become more complex. How you arrange the parts of an expression can change how difficult it is to process with a program. This kind of arrangement of expressions and statements is especially useful in mathematical programming and when writing compilers. In this activity, you will work with two different notations and examine the ways the different notations can be processed.

Learning Objectives (for content & process)

After completing this activity, learners should be able to:

- Explain the complexities of writing programs to evaluate infix expressions.
- Evaluate postfix expressions.
- Convert infix expressions to postfix.
- Identify the differences and similarities between infix and postfix expressions.

Before you start, complete the form below to assign a role to each member.
If you have 3 people, combine Speaker & Reflector.

Team	**Date**
Team Roles	**Team Member**
Recorder: records all answers & questions, and provides copies to team & facilitator.	
Speaker: talks to facilitator and other teams.	
Manager: keeps track of time and makes sure everyone contributes appropriately.	
Reflector: considers how the team could work and learn more effectively.	

Reminders:

1. *Note the time whenever your team starts a new section or question.*
2. *Write legibly & neatly so that everyone can read & understand your responses.*

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

(13 min) A. Infix	start time:

```
4 * ( 2 + 8 - 1 ) + 3 - 6 * 4
```

Evaluating mathematical expressions can be difficult for both humans and computers. Humans usually prefer the notation above, but it poses several problems when trying to write a computer program to evaluate it. This notation is referred to as **infix** because the operator is in between the operands.

1. (3 min) Describe (with sentences or pseudocode) an algorithm for evaluating a single-operator expression like `2 + 8` or `3 - 6` if the operator and operands were **stored in an array** in order.

2. (3 min) Explain (with sentences or pseudocode) what you could do **to extend your algorithm** to evaluate longer expressions using + or −, such as `2 + 8 - 1` or `7 + 4 + 9`.

3. (4 min) Explain (with sentences or pseudocode) why your algorithm could handle any length of expression using + or −, or explain how you could extend it to do so.

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

4. (2 min) Explain why your algorithm would not correctly evaluate `3 - 6 * 4` or `2 + 8 / 2`.

5. (1 min) How do you (not a program) know how to correctly evaluate these expressions or the one at the start of this section?

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

(12 min) B. Postfix	start time:

`4 2 8 + 1 - * 3 4 * +`

The standard notation (such as `3 * 4`) is known as **infix**, because the operator is in the middle of the two operands. While humans may find this easier to process visually, it can be easier when writing computer programs to use **postfix** notation (such as `3 4 *`).

1. (2 min) Describe (with sentences or pseudocode) an algorithm for evaluating a single-operator **postfix** expression like `2 8 +` or `3 4 *` if the operator and operands were **stored in an array** in order.

2. (3 min) Explain (with sentences or pseudocode) what you could do **to extend your algorithm** to evaluate longer expressions such as `2 8 + 1 -` or `5 7 + 3 + 9 + 10 -`. (Note: These should equal `9` and `14`.)

3. (2 min) Explain why `4 2 8 + 1 - *` (equivalent to `4 * (2 + 8 - 1)`) is more difficult to evaluate than the ones in the previous question.

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

You can use a stack to hold on to operands until you encounter an operator, as well as storing the results of each operation to be used as operands later.

```
create an empty stack for operands
for each token in expression
  if token is an operand
    stack.push( operand )
  else // it's an operator
    right = stack.pop( )
    left = stack.pop( )
    result = operation performed on left and right
    stack.push( result )
// stack should have one value at end (the final result)
```

4. (3 min) Using the algorithm above, show the contents of the **operand stack** for each step of evaluating the expression at the start of this section (`4 2 8 + 1 - * 3 4 * +`). Confirm that your result matches the value of the original infix version in section A.

Token	4	2	8	+	1	-	*	3	4	*	+
Stack	4	2 4									

5. (2 min) Evaluate the expression `9 2 12 * 3 / +` using the algorithm above.

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

(13 min) C. Infix to Postfix Conversion	start time:

```
4 * ( 2 + 8 - 1 ) + 3 * 4
```

```
4 2 8 + 1 - * 3 4 * +
```

Converting infix expressions to postfix is at least as difficult as evaluating infix expressions. However, it can be used for expressions with variables that will be evaluated many times with different values, and similar processes are sometimes used by compilers to process code.

1. (2 min) List the operators from the infix expression above in the order that they would be evaluated according to the standard precedence rules.

2. (1 min) Examine the list above and then the postfix expression. What can you see about the operators in the postfix?

3. (1 min) Examine the operands in both the infix and postfix. What relationship does the order of one have with the order of the other?

4. (2 min) Explain why `2 6 + 5 4 * +` is a valid postfix translation of the infix expression `2 + 6 + 5 * 4`, even though it does not fit the same pattern in terms of operators.

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

You can also use a stack to hold onto operators until you need to add them to the postfix expression, much like the stack that held the operands for the evaluation algorithm.

```
create an empty stack for operators
create an empty postfix string
for each token in infix
  if token is an operand
    add token to postfix
  else    // it's an operator
    while( stack not empty &
           precedence(stack.top()) >= precedence(token) )
      add stack.pop( ) to postfix
    stack.push( token )
// copy any remaining operators to postfix
while ( stack not empty )
  add stack.pop( ) to postfix
```

5. (3 min) Using the algorithm above, show the contents of the **operator stack** for each step of converting the expression `10 + 5 - 8 * 2 + 1` to postfix. Also record the final postfix version.

Token	10	+	5	-	8	*	2	+	1	
Stack		+								

Postfix:

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

6. (4 min) Using the infix/postfix conversion at the start of this section and your knowledge of parentheses, fill in a procedure to handle start and end parentheses when converting infix expressions to postfix. Use sentences or pseudocode.

```
create an empty stack for operators
create an empty postfix string
for each token in infix
  if token is an operand
    add token to postfix
  if token is "("




  else if token is ")"





  else // it's an operator
    while( stack not empty &
           precedence(stack.top()) >= precedence(token) )
      add stack.pop( ) to postfix
    stack.push( token )
// copy any remaining operators to postfix
while ( stack not empty )
  add stack.pop( ) to postfix
```

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Reflector: Get an 'Evidence of Competencies' form and provide one piece of positive evidence for each member of the group. Then, complete the section for the 'Assessment of Group.' Share the document with each of your group members. All students should be collecting pieces of evidence for their mid-semester and end of semester reflections.

© 2018-2020 by Sofia Lemons, sofia.lemons@cs.unh.edu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.