2021

Infix to Postfix Parser

GROUP PROJECT

TEAM 1

ADAM JOST | NEHA METLAPALLI

Infix to Postfix Parser Group Project

Objective

The objective of the project was to write an infix expression parser using the stack data structure. The solution was to parse an infix expression given in string format and then efficiently evaluate it, printing the evaluated result to the console.

Our Team

Our team consists of two Java developers:

- Adam Jost
- Neha Metlapalli

Team Member Contributions

Project report:

- Adam Jost
- Neha Metlapalli

UML Diagram:

- Adam Jost

Source Code (Current Version):

- ExpressionEvaluator.java was coded by Adam Jost
- InfixParser.java was coded by Neha Metlapalli
    - Some code contained within was written by Adam Jost
- Postfix.java was coded by Adam Jost
- ListNode.java was coded by Adam Jost
- SinglyLinkedStack.java was coded by Adam Jost
- Printer.java was coded by Adam Jost

Debugging:

- Neha Metlapalli
- Adam Jost

System Functionality Overview

Our team developed a system that first prints a title heading containing a short description of the program. Next, the system reads in each infix expression from the input file. It then parses the expression given in string format and converts the formatted expression to a postfix expression. Next, the postfix expression is evaluated. Finally, the evaluated result is printed to the console. This process is repeated until all expressions contained within the input file have been evaluated. The system is flexible and gracefully handles all invalid expressions. If any of the input expressions are determined to be invalid, the printed result for that expression will be a message stating the expression was invalid. Also, if the infix expression contains an arithmetic error such as divide-by-zero or modulus-by-zero, an error message will be printed to the console informing the user of the arithmetic error.

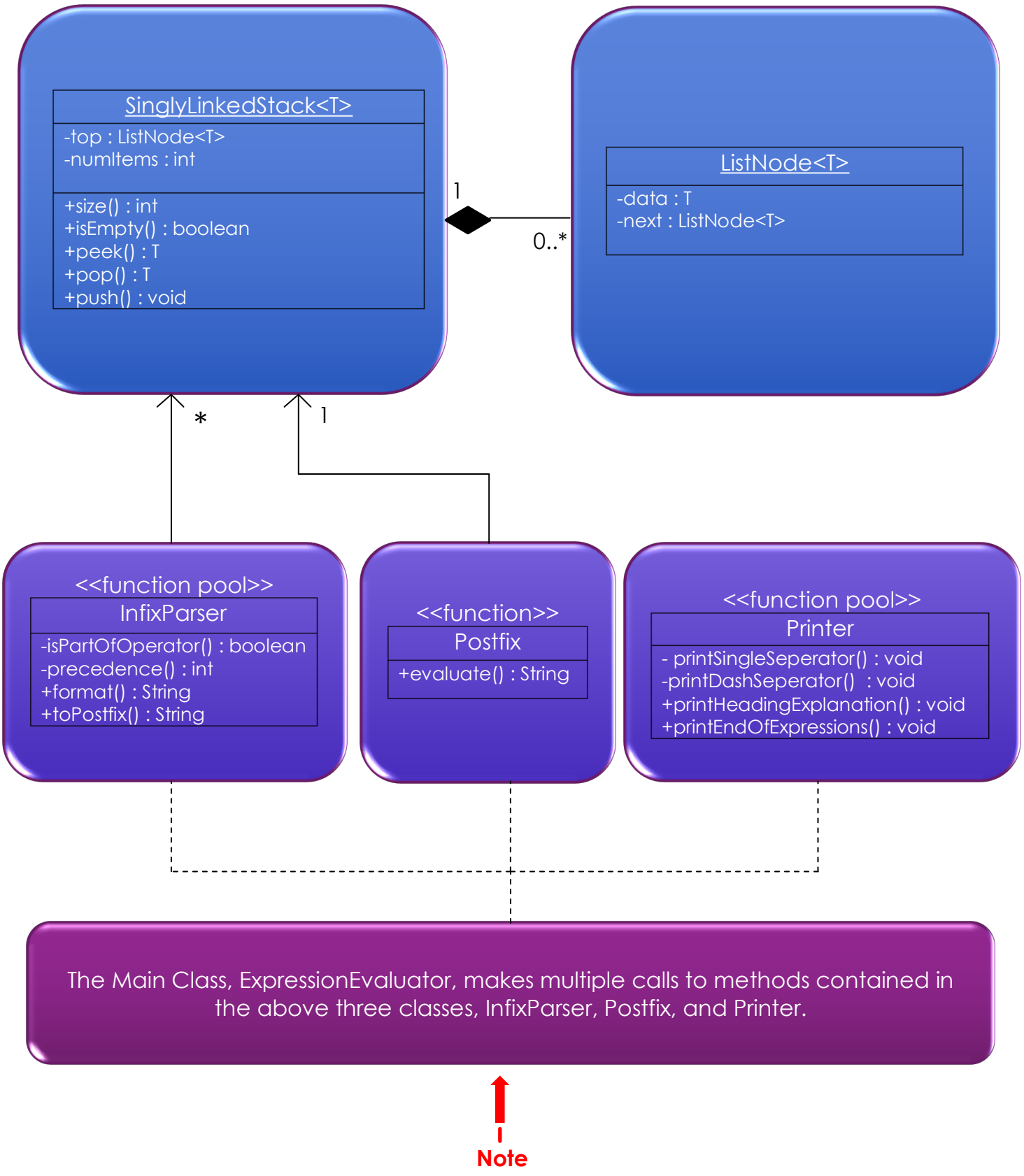
The Interface

```
=====================================================
         INFIX TO POSTFIX PARSER AND EVALUATOR
=====================================================
               Step-By-Step Explanation
-----------------------------------------------------
STEP 1: Read in infix expression from file.
STEP 2: Parse the expression given in String format.
STEP 3: Convert the formatted expression to a postfix
        expression.
STEP 4: Evaluate the postfix expression.
STEP 5: Print the evaluated result.
=====================================================
          Scroll Down to View All Expressions
=====================================================

Infix:   11+2*-33
Postfix: 11 2 -33 * +
Result:  -55

Infix:   2+2^2*3
Postfix: 2 2 2 ^ 3 * +
Result:  14
```

UML Class Diagram



System Design

Physical Design

Physical design relates to the actual input and output processes of the system such as how data is entered into a system, verified, processed, and displayed.

Input Design

File Input

At the start of the program, a one-time operation occurs where the system reads in a source document (.txt file) which contains a list of all of the infix expressions to be parsed and evaluated. Each line in the input file contains a single expression or an empty line. Expressions can be free of whitespace or have any amount of successive whitespace characters in between digits and operators. All valid infix expressions will be formatted, converted to postfix expressions, and then evaluated. All invalid expression will have an error message printed in place of the evaluated result.

Invalid Input Expressions

- Input expressions cannot contain any arithmetic errors such as a divide-by-zero error, or a modulus-b-zero error.
  - `-6 / 0`
  - `6 % 0`
- Input expressions cannot contain variables.
  - `1a + 2b`
  - `a % b`
- In some cases, for input expressions containing multiple subtraction or addition operators consecutively placed one after another the expression will be smartly converted to a valid expression void of the extra,

successive operators. For those cases that cannot be smartly converted the system will print an "Invalid Expression Error" to the console.

  - `1 - - - 1` will convert to `1 + -1`.
  - `1 + + + 1` will convert to `1 + 1`.
  - `1 + - + 1` will result in "Invalid Expression Error".
- Infix expressions containing more than two successive equality operators.
  - `1===1`
  - `1!==1`
- Infix expressions containing two or more successive greater than or less than comparison operators.
  - `1>>1`
  - `1<>1`
- Infix expression containing unsupported characters.
  - `1@#4`

Valid Input Expressions

- Expressions free of arithmetic errors.
- Contains only integers, operators, and whitespace.
- Can contain multiple whitespace characters in succession.
  - `20 - (    - 5)`
  - `-81 % 4 *(             -         23) /   4`
  - `1&&              1   &&  1   &&  0`
- Examples of input expressions considered to be "valid":
  - `2%2+2^2-5*(3^2)`
  - `11+ 2* - 33`
  - `7&&0`
  - `7||0`
  - `-(-6^2)`

- `1==1`
- `1!=1`
- `-1- -1(2)`

Output

After each expression is read-in, parsed, and evaluated the system will print the following three items to the console, the infix expression, the postfix expression, and the evaluated result. If the expression is found to be invalid the system will print "Invalid Expression Error" in place of the evaluated result. If the expression was found to contain an arithmetic error the system will print "Arithmetic Error" in place of the evaluated result.

Output Examples

- Valid expressions

```
Infix:   11+2*-33
Postfix: 11 2 -33 * +
Result:  -55
```

- Invalid expressions

```
Infix:   -+-+1
Postfix: - 1 -
Result:  Invalid Expression Error
Error: Invalid Infix Expression
```

- Arithmetic errors

```
Infix:   -3/0
Postfix: -3 0 /
Result:  Arithmetic Error
Error: Divide By Zero
```

Architectural Design

Architectural design focuses on the design of system architecture. It describes defines the structure and relationship between various parts of the systems.

Data Structures

- *ListNode*
    - Used to store individual expression tokens.
- *SinglyLinkedStack*
    - Data container for 0…* *ListNode(s)*.
    - Implemented using a Singly LinkedList.
    - User-defined implementation of a Stack data structure.

Major Classes

There is a total of 5 major classes:

- *ExpressionEvaluator*
- *InfixParser*
- *PostFix*
- *SinglyLinkedStack*
- *ListNode*

Relationship Between the Major Classes

- *ExpressionEvaluator* calls functions from the InfixParser class.
- *ExpressionEvaluator* calls the sole function in the Postfix class.
- *InfixParser* contains a pool of functions called by *ExpressionEvaluator*.
- *Postfix* contains a single function called by *ExpressionEvaluator*.
- *InfixParser* creates a *SinglyLinkedStack*.
- *Postfix* creates a *SinglyLinkedStack*.
- *SinglyLinkedStack* creates and contains 0…* *ListNode*(s).
- *ListNode* is used to create a *Singly-Linked-List* which is used to implement the *SinglyLinkedStack* data container.

Class Details

ListNode<T>

- Param:
  - *<T>* : *Object* type.
- Attributes:
  - *T* data – The data being stored.
  - *ListNode<T>* next – The next *ListNode* in the *Singly-Linked-List*.
- Created by *SinglyLinkedStack*.
- Data Structures:
  - *ListNode<T>* : Used to implement a *Singly-Linked-List*.

SinglyLinkedStack<T>

- Param:
  - *<T>* : *Object* type.
- Attributes:
  - *ListNode<T> top* – The top of the *SinglyLinkedStack*.
  - *Int numItems* – The number of items currently in the *SinglyLinkedStack*.
- Created by *InfixParser* and *Postfix*.
- Used as a data container to store items of infix and postfix expressions.
- Data Structures:
  - *ListNode<T>* : Used to implement a *Singly-Linked-List*.
  - Singly-Linked-List: Used to implement the *SinglyLinkedStack*.
  - *Stack: SinglyLinkedStack*, itself, is a *Stack* data structure.

InfixParser

Pool of functions used by ExpressionEvaluator. Formats, parses, and converts an infix expression to a postfix expression.

- Data Structures:
    - SinglyLinkedStack<String> stack
        - Contains *String* objects which are representations of individual integer values and operators contained within the current expression being parsed.

Postfix

Comprised of a single function used by ExpressionEvaluator that evaluates a postfix expression.

- Data Structures:
    - SinglyLinkedStack<Integer> stack
        - Contains *Integer* objects which are the integer values contained in the current postfix expression that is being evaluated.

ExpressionEvaluator

The "main program".

Test Cases

Test Case 1 : Unary and Subtraction Operators

This test was performed using the following infix expressions contained in an input file.

```
1 -(8-1)
2 1* -1
3 -6^6
4 1- -1
5 1---1
6 1----1
```

Expected output

- Infix expression on line 1:
    - Original input expression: -(8-1)
    - Infix expression: -(8-1)
    - Postfix: -1 8 1 - *
    - Result: -7
- Infix expression on line 2:
    - Original input expression: 1* -1
    - Infix expression: 1*-1
    - Postfix expression: -1 8 1 - *
    - Evaluated Result: -7
- Infix expression on line 3:
    - Original input expression:-6^2
    - Expected Infix expression: -6^2
    - Expected postfix expression: -1 6 6 ^ *
    - Expected result: -46656
- Infix expression on line 4:
    - Original input expression: 1- -1
    - Expected Infix expression: 1--1
    - Expected postfix expression: 1 1 +
    - Expected result: 2
- Infix expression on line 5:
    - Original input expression: 1---1
    - Expected Infix expression: 1---1
    - Expected postfix expression: 1 -1 +
        - Program is expected to convert the first two "-" operators to a "+" operator.
    - Expected result: 0
- Infix expression on line 6:
    - Original input expression: 1----1

- Expected Infix expression: 1----1
- Expected postfix expression: 1 + 1 +
  - Program is expected to attempt to correct the illegal use of operators but without success.
- Expected result: "Invalid Expression Error"

Actual Output

Actual output printed to the console

```
Infix:   -(8-1)
Postfix: -1 8 1 - *
Result:  -7

Infix:   1*-1
Postfix: 1 -1 *
Result:  -1

Infix:   -6^6
Postfix: -1 6 6 ^ *
Result:  -46656

Infix:   1--1
Postfix: 1 1 +
Result:  2

Infix:   1---1
Postfix: 1 -1 +
Result:  0

Infix:   1----1
Postfix: 1 + 1 +
Result:  Invalid Expression Error
Error: Invalid Infix Expression
```

Differences: Expected vs. Actual Output

This test was a success. The correct output was printed as expected with no differences for all input expressions.

Test Case 2 : Logic, Equality, and Comparison Operators

This test was performed using the following infix expressions contained in an input file.

```
1 1 && 0
2 1 || 1
3 1 &&& 1
4 1 >= 1
5 -1 <= 1
6 2 == -2
7 2 != -2
8 4 === 4
9 5 !=== 5
10 1 <> 1
11 1 >< 1
```

Expected Output

Logic, equality, and comparison expressions are evaluated to either true or false. False is represented by '0' and true is represented by '1'.

- Infix expression on line 1:
  - Original input expression: 1 && 0
  - Infix expression: 1&&0
  - Postfix: 1 0 &&
  - Result: 0
- Infix expression on line 2:
  - Original input expression: 1 || 1
  - Infix expression: 1||1
  - Postfix expression: 1 1 ||
  - Evaluated Result: 1
- Infix expression on line 3:
  - Original input expression1&&&1
  - Expected Infix expression: 1&&&1
  - Expected postfix expression: Invalid Expression Error

    - Expected result: Invalid Expression Error
- Infix expression on line 4:
    - Original input expression: 1 >= 1
    - Expected Infix expression: 1>=1
    - Expected postfix expression: 1 1 >=
    - Expected result: 1
- Infix expression on line 5:
    - Original input expression: -1 <= 1
    - Expected Infix expression: -1<=1
    - Expected postfix expression: -1 1 <=
    - Expected result: 1
- Infix expression on line 6:
    - Original input expression: 2 == -2
    - Expected Infix expression: 2==-2
    - Expected postfix expression: 2 -2 ==
    - Expected result: 0
- Infix expression on line 7:
    - Original input expression: 2 != -2
    - Expected Infix expression: 2!=-2
    - Expected postfix expression: 2 -2 !=
    - Expected result: 1
- Infix expression on line 8:
    - Original input expression: 4 === 4
    - Expected Infix expression: 4===4
    - Expected postfix expression: Invalid Expression Error
    - Expected result: Invalid Expression Error
- Infix expression on line 9:
    - Original input expression: 5 !=== 5
    - Expected Infix expression: 5!===5

    - Expected postfix expression: Invalid Expression Error
    - Expected result: Invalid Expression Error
- Infix expression on line 10:
    - Original input expression: 1 <> 1
    - Expected Infix expression: 1<>1
    - Expected postfix expression: Invalid Expression Error
    - Expected result: Invalid Expression Error
- Infix expression on line 11:
    - Original input expression: 1 >< 1
    - Expected Infix expression: 1><1
    - Expected postfix expression: Invalid Expression Error
    - Expected result: Invalid Expression Error

Actual Output

```
Infix:   1&&0
Postfix: 1 0 &&
Result:  0

Infix:   1||1
Postfix: 1 1 ||
Result:  1

Infix:   1&&&1
Postfix: Invalid Expression Error
Result:  Invalid Expression Error
Error: Invalid Infix Expression

Infix:   1>=1
Postfix: 1 1 >=
Result:  1

Infix:   -1<=1
Postfix: -1 1 <=
Result:  1
```

```
Infix:   2==-2
Postfix: 2 -2 ==
Result:  0

Infix:   2!=-2
Postfix: 2 -2 !=
Result:  1

Infix:   4===4
Postfix: Invalid Expression Error
Result:  Invalid Expression Error
Error: Invalid Infix Expression

Infix:   5!===5
Postfix: Invalid Expression Error
Result:  Invalid Expression Error
Error: Invalid Infix Expression

Infix:   1<>1
Postfix: Invalid Expression Error
Result:  Invalid Expression Error
Error: Invalid Infix Expression

Infix:   1><1
Postfix: Invalid Expression Error
Result:  Invalid Expression Error
Error: Invalid Infix Expression
```

Differences: Expected vs. Actual Output

This test was a success. The correct output was printed as expected with no differences for all input expressions.

Future Improvements

This program could be improved by adding more explanation to the output pertaining to the exact reason why the expression was determined to be invalid opposed to simply printing the "Invalid Infix Expression" error. Another improvement could be adding the support for expressions containing variables. The program could also be improved by prompting the user to enter one of several commands that would allow them to edit expressions, add new

expressions and delete expressions contained within the input file once all of the expressions contained within the input file have been successfully completed being evaluated.