Arithmetic Notation, Shunting Yard Algorithm, Binary Expression Tree

Introduction

Important programming concepts for this lab are Arithmetic Algebraic Expressions, the different expression notations like Prefix, Postfix and Infix, the evaluation order of the expressions (precedence), and how to convert an expression from one notation to another.

Each concept is backed by algorithms, and Pre-Visualizations illustrative examples to understand concepts clearly.

We will be using the concepts of Stacks using a shunting yard algorithm to build a postfix expression string.

Arithmetic Algebraic Expressions and Notation

An algebraic expression is made up of a legal combination of operands and operators.
A stand-alone operand is also a valid expression.
Tokens are the symbols that make up an expression.
Tokens can be operands or operators.

Operators operate on operands and binary operators operate on two operands.
Binary operators are tokens that signify a mathematical operation between the two operands.

The operands represent the quantity (unit of data) on which an arithmetical operation is performed.

For our purpose for this lab, the algebraic expressions are restricted to the following:

- Each token is a single character
- For example, A is represented as the character 'A'
- There are no blanks (spaces) allowed in the expression string to be parsed
- Operand tokens can be single letter unsigned variables alphabet characters (x, y, z, A, B, C)
- All expressions are binary
- All operators are the binary operators: +, -, *, /
- Parentheses, the [( ] and [ )] characters, are valid expression tokens that will guide formation of the postfix expression
- The input infix String expression is assumed to follow the rules and does not require rules checking of the input expressions

Considering the aforementioned definitions for a binary expression, we can write an example of an infix expression as:

x+y*z.

Algebraic Expressions can be represented using three different common notations.

INFIX

Expressions in which operands surround the operator, e.g., x+y, 6*3 etc. This way of writing the Expressions is called infix notation. It may require use of parentheses to dictate the evaluation order.

PREFIX:

Prefix notation, also known as Polish notation, is a symbolic logic invented by Polish mathematician Jan Lukasiewicz in the 1920s. In the prefix notation, the operator comes before the operands, e.g., +by, *+xyz … etc.
No parentheses are required to dictate evaluation order.

POSTFIX:

Postfix notation is also known as Reverse Polish notation.
In this notation, the postfix notation, the operator comes after the operands, e.g., xy+, xyz+* etc.
No parentheses are required to dictate evaluation order.

Evaluation Ordering

Infix notations, although familiar to us, are not as simple as they seem, especially while evaluating them. To evaluate an infix expression, we need to consider Operators' Priority and Associativity.

For example, the expression:
3 + 5 * 4
Evaluates to 32 i.e., 3 + (5 * 4).

The infix operator precedence governs evaluation order. Therefore, infix notation requires parentheses if you want exact control of overriding the precedence order of the operators in the expression.

The following is the order of precedence for infix notation binary operators.

()
/, *
+, -

What about the expression 6 / 3 * 2?
This expression evaluates to 4 i.e. (6/3)*2.
As both * and the / have same priorities, use Operators Associativity property to properly evaluate.

Operator Associativity governs the evaluation order of the operators of same priority. For an operator with left-Associativity, evaluation is from left to right.

All the binary operators we use in this assignment have left associativity.

Since parentheses, operator priority and operator left associativity must be considered while evaluating an infix expression, converting the infix expression to postfix notation will eliminate these criteria. Both prefix and postfix notations have an advantage over infix notation in that while evaluating an expression in prefix or postfix form, you do not need to use parentheses, operator priority, and operator left associativity, because prefix and postfix notations do not require such criteria to direct evaluation order.

E.g., x/(y*z) becomes:
/*xyz in prefix
xyz*/ in postfix.

Both prefix and postfix notations make expression evaluation much easier, both are used by compilers and Floating-Point Units on most modern CPUs. The expression is scanned in infix form; it is converted into prefix or postfix form with parentheses no longer required and then evaluated.

Pre-Lab Visualization

Converting Expression from Infix to Postfix using Stack Shunting Yard Algorithm

To convert an expression from infix to postfix, we are going to use a stack.

Shunting Yard Algorithm

The **shunting-yard algorithm** is a method for parsing mathematical expressions specified in infix notation.
It can be used to produce output in post fix notation.

The algorithm was invented by Edger Dijkstra (1961), named the "shunting yard" algorithm, because its operation resembles that of railroad shunting yards commonly used in Europe for train car assembly order management. In a general analogy, shunt yards were used to assemble trains consisting of a certain train car ordering.

For our assignment, all the operators are assumed to be binary left associative.
The input tokens (train cars) are processed one symbol at a time.

Shunting Yard Algorithm

When an input token operand is processed, the input token operand is always put directly to the output.
All operator and parenthesis input tokens operate with the stack.

If the stack is empty and the input token is an operator, the operator symbol is put on the stack.

If the stack is not empty and the input token is an operator with a precedence that is less than or equal to the shunt stack top operator token precedence, then that stack top operator token is popped off the stack and sent to the output. This placing of stack top operators with a greater precedence than the current input operator token will continue until a stack top operator of greater precedence occurs or the stack becomes empty. When this stack processing is done for the input operator token, then the input operator token is put on the stack.

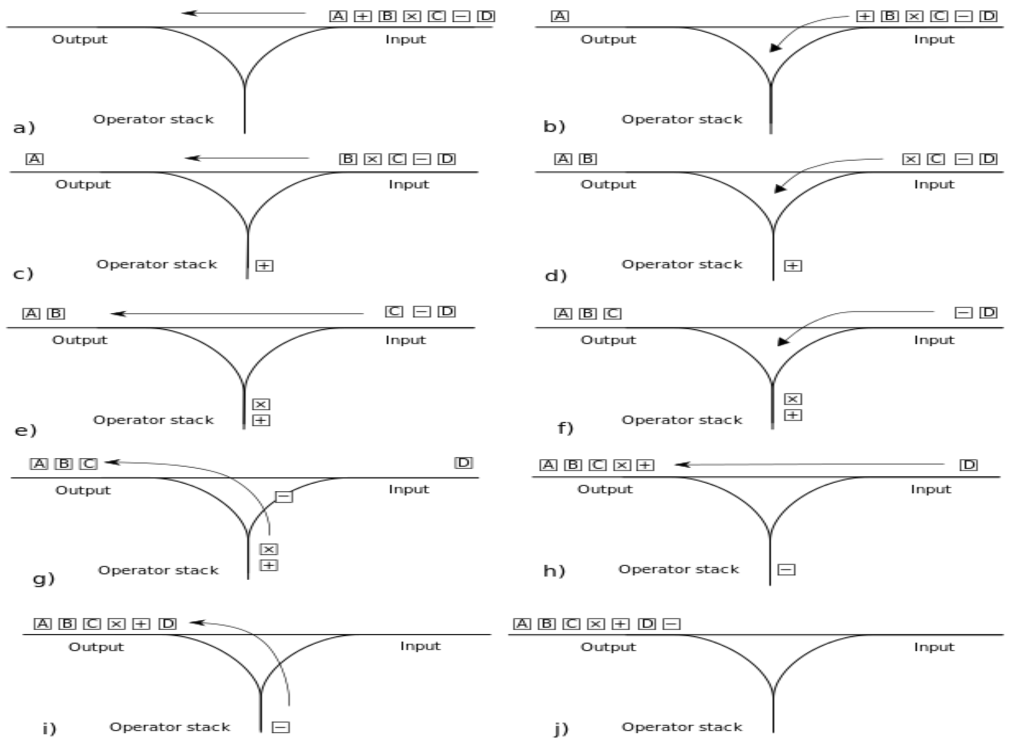
If the input token is an open parenthesis [(], it is always put on the stack.

If the input token is a closed parenthesis [)], the tokens on the stack are put to the output until an open parenthesis [(] is encountered on the stack, then the open parenthesis [{] is then discarded off the stack and the input token processing continues.

When all input tokens are done being processed, any remaining operators on the stack are popped and sent to the output to finish.

Arithmetic Notation, Shunting Yard Algorithm, Binary Expression Tree

Shunting Yard Algorithm Graphical Illustration Example



a + b * c into Postfix form:

Infix Token	Stack Contents	Postfix Expression
a	Empty	a
+	+	a
b	+	ab
*	* +	ab
c	* +	abc
Empty	Empty	abc*+

A + B * C - D into Postfix form:

Infix Token	Stack Contents	Postfix Expression
A	Empty	A
+	+	A
B	+	AB
*	* +	AB
C	* +	ABC
-	-	ABC*+
D	-	ABC*+D
Empty	Empty	ABC*+D-

A * (B - C) into Postfix form:

Infix Token	Stack Contents	Postfix Expression
A	Empty	A
*	*	A
(	(*	A
*	*	AB
B	(*	AB
-	- (*	AB
C	- (*	ABC
)	*	ABC-
Empty	Empty	ABC-*

Directed Lab Work

Use a stack shunting yard algorithm and the STL stack to create a postfix expression string.

Read Into a Stream from an Input File

You will create a text file that will consist of infix expression strings, one expression string per line.
Assume that the file contents are all valid infix expressions follow the rules established in this assignment.
Do not have the program check for arithmetical syntax errors that may exist in the file.
Remove such erroneous items or fix these items in the file.

You must include infix expressions where some of the expressions contain parentheses.
You must include at least 10 lines of expressions that test various cases to make sure the program is acting correctly.

Use the C++ streams capability to read the file contents into a string-based stream object.

Your program must:

- Read a line from the input file that has a valid expression into an infix string, then process that string
    - The file must be processed 1 line at a time and the whole file should not be read into a buffer containing the whole file
- Display the inputted infix string
- Process the infix string to a postfix string using a shunting yard algorithm
- Display the final resulting postfix string.
- The code should then go back and read another infix expression line from the input until the whole file has been read and processed.

The input expression file must have a valid expression using the rules in this document. You need to show that you are testing valid expressions for a variety of expressions that covers all operators, operator precedence and use parentheses.
You will be graded on the expressions you create in the input file to thoroughly test your program.

The program must use:

C++ string class: for all input and output expression strings

Application class named: ShuntingYardClass

The program must be modular, well structured, well organized and follow the best programming practices document.

See the associated cpp file included with this assignment to aid you in your design.

Binary Expression Trees

The post fix expression should be used to build a binary expression tree.

Your program should do a pre-order, in-order and post order to display of each of the trees that have been built.

Follow the guide below to correctly build the binary expression tree for the post-fix expression.

```
// Construction of an expression tree uses a stack to hold operand nodes
//
// To construct an expression tree:
//
//    Loop through the postfix expression for every token
//       If the token is an operand
//         Put the token into a node and push the operand token node onto the stack
//          Else
//         token is an operator
//         Place/Pop 2 token nodes from stack, make these tokens the children for the current token node
//
//    When loop is done processing tokens,
//    The stack node will be the tree root
```

Arithmetic Notation, Shunting Yard Algorithm, Binary Expression Tree

Below is a sample output for a simple expressions file.

```
InFix   Expression : a+b*c-d
PostFix Expression : abc*+d-

Tree  pre-order expression is
-+a*bcd

Tree   in-order expression is
(a+b*c)-d

Tree post-order expression is
abc*+d-



InFix   Expression : A+B*(C-D)
PostFix Expression : ABCD-*+

Tree  pre-order expression is
+A*B-CD

Tree   in-order expression is
A+B*(C-D)

Tree post-order expression is
ABCD-*+



InFix   Expression : a*(b+c)-d/e
PostFix Expression : abc+*de/-

Tree  pre-order expression is
-*a+bc/de

Tree   in-order expression is
a*(b+c)-d/e

Tree post-order expression is
abc+*de/-



Press the enter key once or twice to end
```