Converting infix to RPN (shunting-yard algorithm)

October 5, 2010 (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/) Andrei Ciobanu (http://andreinc.net/author/admin/) Data Structures and Algorithms (http://andreinc.net/category/algorithms/), Java (http://andreinc.net/category/programming-languages/java-programming-languages/), Programming Languages (http://andreinc.net/category/programming-languages/), Python (http://andreinc.net/category/programming-languages/python-programming-languages/)

If you've tried to write your own calculator (something in the style of gcalctool (http://live.gnome.org/Gcalctool)) you've probably had to build a simple converter for your mathematical expressions from infix notation (http://en.wikipedia.org/wiki/Infix_notation) to RPN (http://en.wikipedia.org/wiki/Postfix_notation) (Reverse Polish Notation).

Inspired by this classical SPOJ (http://www.spoj.pl/problems/ONP/) challenge I wanted to write my own simplified version of an expression converter. If this topic is new for you, or you need to refresh your memory please don't skip the following paragraph.

Infix notation (http://en.wikipedia.org/wiki/Infix_notation) is the common arithmetic and logical formula notation, in which operators are written infix-style between the operands they act on (e.g. 2 **+** 5). Unfortunately what seems natural for us, is not as simple to parse by computers as prefix or RPN notations.

RPN (http://en.wikipedia.org/wiki/Postfix_notation) also known as the Reverse Polish Notation is mathematical notation wherein every operator (eg. + – * %) follows all of its operands. Examples:

Infix notation	Reverse Polish Notation
A + B	A B +
A ^ 2 + 2 * A * B + B ^ 2	A 2 ^ 2 A * B * + B 2 ^ +
((1 + 2) / 3) ^ 4	1 2 + 3 / 4 ^
(1 + 2) * (3 / 4) ^ (5 + 6)	1 2 + 3 4 / 5 6 + ^ *

In order to parse and convert a given infix mathematical expression to RPN we will use the ***shunting-yard algorithm (http://en.wikipedia.org/wiki/Shunting-yard_algorithm)*** . Just like the evaluation of RPN, the algorithm is stack-based . For the conversion we will use two buffers (one for input, and one for output). Additionally we will use a stack for operators that haven't been yet added to the output.

A **simplified** version of the **Shunting-yard algorithm** (complete version (http://en.wikipedia.org/wiki/Shunting-yard_algorithm#The_algorithm_in_detail))

- For all the input tokens [S1]:
  - Read the next token [S2];
  - If token is an operator **(x)** [S3]:
    - While there is an operator **(y)** at the top of the operators stack and either **(x)** is left-associative (http://en.wikipedia.org/wiki/Operator_associativity) and its precedence (http://en.wikipedia.org/wiki/Order_of_operations) is less or equal to that of **(y),** or **(x)** is right-associative (http://en.wikipedia.org/wiki/Operator_associativity) and its precedence is less than **(y)** [S4]:
      - Pop **(y)** from the stack [S5];
      - Add **(y)** output buffer [S6];
    - Push (x) on the stack [S7];
  - Else If token is left parenthesis, then push it on the stack [S8];
  - Else If token is a right parenthesis [S9]:
    - Until the top token (from the stack) is left parenthesis, pop from the stack to the output buffer [S10];
    - Also pop the left parenthesis but don't include it in the output buffer [S11];
  - Else add token to output buffer [S12].
- While there are still operator tokens in the stack, pop them to output [S13]

Note: [SN] Relate with code.

I) Java Implementation

The following implementation of the shunting-yard algorithm does not impose any validations. The input should be a valid mathematical expression or else the program may end abruptly or perform incorrectly.

Step 1 : Declaring and defining operators

```
// Associativity constants for operators
    private static final int LEFT_ASSOC = 0;
    private static final int RIGHT_ASSOC = 1;

    // Supported operators
    private static final Map<String, int[]> OPERATORS = new HashMap<String, int[]>();
    static {
        // Map<"token", []{precendence, associativity}>
        OPERATORS.put("+", new int[] { 0, LEFT_ASSOC });
        OPERATORS.put("-", new int[] { 0, LEFT_ASSOC });
        OPERATORS.put("*", new int[] { 5, LEFT_ASSOC });
        OPERATORS.put("/", new int[] { 5, LEFT_ASSOC });
        OPERATORS.put("%", new int[] { 5, LEFT_ASSOC });
        OPERATORS.put("^", new int[] { 10, RIGHT_ASSOC });
    }

    /**
     * Test if a certain is an operator .
     * @param token The token to be tested .
     * @return True if token is an operator . Otherwise False .
     */
    private static boolean isOperator(String token) {
        return OPERATORS.containsKey(token);
    }

    /**
     * Test the associativity of a certain operator token .
     * @param token The token to be tested (needs to operator).
     * @param type LEFT_ASSOC or RIGHT_ASSOC
     * @return True if the tokenType equals the input parameter type .
     */
    private static boolean isAssociative(String token, int type) {
        if (!isOperator(token)) {
            throw new IllegalArgumentException("Invalid token: " + token);
        }
        if (OPERATORS.get(token)[1] == type) {
            return true;
        }
        return false;
    }

    /**
     * Compare precendece of two operators.
     * @param token1 The first operator .
     * @param token2 The second operator .
     * @return A negative number if token1 has a smaller precedence than token2,
     * 0 if the precendences of the two tokens are equal, a positive number
     * otherwise.
     */
    private static final int cmpPrecedence(String token1, String token2) {
        if (!isOperator(token1) || !isOperator(token2)) {
            throw new IllegalArgumentException("Invalied tokens: " + token1
                    + " " + token2);
        }
        return OPERATORS.get(token1)[0] - OPERATORS.get(token2)[0];
    }
```

Step 2 : Parsing expression

The input in this case should an array of strings, where every little string is a token. The output will also be an array of strings but in RPN order. (Take a look at the code comments, and the algorithm references **[Sn]**).

```
    public static String[] infixToRPN(String[] inputTokens) {
        ArrayList<String> out = new ArrayList<String>();
        Stack<String> stack = new Stack<String>();
        // For all the input tokens [S1] read the next token [S2]
        for (String token : inputTokens) {
            if (isOperator(token)) {
                // If token is an operator (x) [S3]
                while (!stack.empty() && isOperator(stack.peek())) {
                    // [S4]
                    if ((isAssociative(token, LEFT_ASSOC) && cmpPrecedence(
                            token, stack.peek()) <= 0)
                            || (isAssociative(token, RIGHT_ASSOC) && cmpPrecedence(
                                    token, stack.peek()) < 0)) {
                        out.add(stack.pop());   // [S5] [S6]
                        continue;
                    }
                    break;
                }
                // Push the new operator on the stack [S7]
                stack.push(token);
            } else if (token.equals("(")) {
                stack.push(token);  // [S8]
            } else if (token.equals(")")) {
                // [S9]
                while (!stack.empty() && !stack.peek().equals("(")) {
                    out.add(stack.pop()); // [S10]
                }
                stack.pop(); // [S11]
            } else {
                out.add(token); // [S12]
            }
        }
        while (!stack.empty()) {
            out.add(stack.pop()); // [S13]
        }
        String[] output = new String[out.size()];
        return out.toArray(output);
    }
```

Step 3 : Testing the working converter

```
    public static void main(String[] args) {
        String[] input = "( 1 + 2 ) * ( 3 / 4 ) ^ ( 5 + 6 )".split(" ");
        String[] output = infixToRPN(input);
        for (String token : output) {
            System.out.print(token + " ");
        }
    }
```

And the output:

```
1 1 2 + 3 4 / 5 6 + ^ *
```

Step 4 : Putting all togheter

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Stack;

public class ReversePolishNotation {
    // Associativity constants for operators
    private static final int LEFT_ASSOC = 0;
    private static final int RIGHT_ASSOC = 1;

    // Supported operators
    private static final Map<String, int[]> OPERATORS = new HashMap<String, int[]>();
    static {
        // Map<"token", []{precendence, associativity}>
        OPERATORS.put("+", new int[] { 0, LEFT_ASSOC });
        OPERATORS.put("-", new int[] { 0, LEFT_ASSOC });
        OPERATORS.put("*", new int[] { 5, LEFT_ASSOC });
        OPERATORS.put("/", new int[] { 5, LEFT_ASSOC });
        OPERATORS.put("%", new int[] { 5, LEFT_ASSOC });
        OPERATORS.put("^", new int[] { 10, RIGHT_ASSOC });
    }

    /**
     * Test if a certain is an operator .
     * @param token The token to be tested .
     * @return True if token is an operator . Otherwise False .
     */
    private static boolean isOperator(String token) {
        return OPERATORS.containsKey(token);
    }

    /**
     * Test the associativity of a certain operator token .
     * @param token The token to be tested (needs to operator).
     * @param type LEFT_ASSOC or RIGHT_ASSOC
     * @return True if the tokenType equals the input parameter type .
     */
    private static boolean isAssociative(String token, int type) {
        if (!isOperator(token)) {
            throw new IllegalArgumentException("Invalid token: " + token);
        }
        if (OPERATORS.get(token)[1] == type) {
            return true;
        }
        return false;
    }

    /**
     * Compare precendece of two operators.
     * @param token1 The first operator .
     * @param token2 The second operator .
     * @return A negative number if token1 has a smaller precedence than token2,
     * 0 if the precendences of the two tokens are equal, a positive number
     * otherwise.
     */
    private static final int cmpPrecedence(String token1, String token2) {
        if (!isOperator(token1) || !isOperator(token2)) {
            throw new IllegalArgumentException("Invalied tokens: " + token1
                    + " " + token2);
        }
        return OPERATORS.get(token1)[0] - OPERATORS.get(token2)[0];
    }

    public static String[] infixToRPN(String[] inputTokens) {
        ArrayList<String> out = new ArrayList<String>();
        Stack<String> stack = new Stack<String>();
        // For all the input tokens [S1] read the next token [S2]
        for (String token : inputTokens) {
            if (isOperator(token)) {
                // If token is an operator (x) [S3]
                while (!stack.empty() && isOperator(stack.peek())) {
                    // [S4]
                    if ((isAssociative(token, LEFT_ASSOC) && cmpPrecedence(
                            token, stack.peek()) <= 0)
                            || (isAssociative(token, RIGHT_ASSOC) && cmpPrecedence(
                                    token, stack.peek()) < 0)) {
                        out.add(stack.pop());   // [S5] [S6]
                        continue;
                    }
                    break;
                }
                // Push the new operator on the stack [S7]
                stack.push(token);
            } else if (token.equals("(")) {
                stack.push(token);  // [S8]
            } else if (token.equals(")")) {
                // [S9]
                while (!stack.empty() && !stack.peek().equals("(")) {
                    out.add(stack.pop()); // [S10]
                }
```

```
91                 stack.pop(); // [S11]
92             } else {
93                 out.add(token); // [S12]
94             }
95         }
96         while (!stack.empty()) {
97             out.add(stack.pop()); // [S13]
98         }
99         String[] output = new String[out.size()];
100        return out.toArray(output);
101    }
102
103    public static void main(String[] args) {
104        String[] input = "( 1 + 2 ) * ( 3 / 4 ) ^ ( 5 + 6 )".split(" ");
105        String[] output = infixToRPN(input);
106        for (String token : output) {
107            System.out.print(token + " ");
108        }
109    }
110 }
```

II) Python Implementation

The python implementation is a complete "translation" of the previous Java implementation (only the syntax was changed ... in better).

Step 1 : Declaring and defining operators

```
1  #Associativity constants for operators
2  LEFT_ASSOC = 0
3  RIGHT_ASSOC = 1
4
5  #Supported operators
6  OPERATORS = {
7      '+' : (0, LEFT_ASSOC),
8      '-' : (0, LEFT_ASSOC),
9      '*' : (5, LEFT_ASSOC),
10     '/' : (5, LEFT_ASSOC),
11     '%' : (5, LEFT_ASSOC),
12     '^' : (10, RIGHT_ASSOC)
13 }
14
15 #Test if a certain token is operator
16 def isOperator(token):
17     return token in OPERATORS.keys()
18
19 #Test the associativity type of a certain token
20 def isAssociative(token, assoc):
21     if not isOperator(token):
22         raise ValueError('Invalid token: %s' % token)
23     return OPERATORS[token][1] == assoc
24
25 #Compare the precedence of two tokens
26 def cmpPrecedence(token1, token2):
27     if not isOperator(token1) or not isOperator(token2):
28         raise ValueError('Invalid tokens: %s %s' % (token1, token2))
29     return OPERATORS[token1][0] - OPERATORS[token2][0]
```

Step 2 : Parsing expression

Just like in the previous example, the algorithm does not impose any validation. The input expression should be composed of valid tokens, or else the program may malfunction or end abruptly.

```
1  #Transforms an infix expression to RPN
2  def infixToRPN(tokens):
3      out = []
4      stack = []
5      #For all the input tokens [S1] read the next token [S2]
6      for token in tokens:
7          if isOperator(token):
8              # If token is an operator (x) [S3]
9              while len(stack) != 0 and isOperator(stack[-1]):
10                 # [S4]
11                 if (isAssociative(token, LEFT_ASSOC)
12                     and cmpPrecedence(token, stack[-1]) <= 0) or
13                     (isAssociative(token, RIGHT_ASSOC)
14                     and cmpPrecedence(token, stack[-1]) < 0):
15                     # [S5] [S6]
16                     out.append(stack.pop())
17                     continue
18                 break
19             # [S7]
20             stack.append(token)
21         elif token == '(':
22             stack.append(token) # [S8]
23         elif token == ')':
24             # [S9]
25             while len(stack) != 0 and stack[-1] != '(':
26                 out.append(stack.pop()) # [S10]
27             stack.pop() # [S11]
28         else:
29             out.append(token) # [S12]
30     while len(stack) != 0:
31         # [S13]
32         out.append(stack.pop())
33     return out
```

Step 3 : Testing the converter

```
1 if __name__ == '__main__':
2     input = "( 1 + 2 ) * ( 3 / 4 ) ^ ( 5 + 6 )".split(" ")
3     output = infixToRPN(input)
4     print output
```

And the output:

```
['1', '2', '+', '3', '4', '/', '5', '6', '+', '^', '*']
```

Putting all togheter:

```
'''
Created on Oct 5, 2010

@author: nomemory
'''

#Associativity constants for operators
LEFT_ASSOC = 0
RIGHT_ASSOC = 1

#Supported operators
OPERATORS = {
    '+' : (0, LEFT_ASSOC),
    '-' : (0, LEFT_ASSOC),
    '*' : (5, LEFT_ASSOC),
    '/' : (5, LEFT_ASSOC),
    '%' : (5, LEFT_ASSOC),
    '^' : (10, RIGHT_ASSOC)
}

#Test if a certain token is operator
def isOperator(token):
    return token in OPERATORS.keys()

#Test the associativity type of a certain token
def isAssociative(token, assoc):
    if not isOperator(token):
        raise ValueError('Invalid token: %s' % token)
    return OPERATORS[token][1] == assoc

#Compare the precedence of two tokens
def cmpPrecedence(token1, token2):
    if not isOperator(token1) or not isOperator(token2):
        raise ValueError('Invalid tokens: %s %s' % (token1, token2))
    return OPERATORS[token1][0] - OPERATORS[token2][0]

#Transforms an infix expression to RPN
def infixToRPN(tokens):
    out = []
    stack = []
    #For all the input tokens [S1] read the next token [S2]
    for token in tokens:
        if isOperator(token):
            # If token is an operator (x) [S3]
            while len(stack) != 0 and isOperator(stack[-1]):
                # [S4]
                if (isAssociative(token, LEFT_ASSOC)
                    and cmpPrecedence(token, stack[-1]) <= 0) or
                    (isAssociative(token, RIGHT_ASSOC)
                    and cmpPrecedence(token, stack[-1]) < 0):
                    # [S5] [S6]
                    out.append(stack.pop())
                    continue
                break
            # [S7]
            stack.append(token)
        elif token == '(':
            stack.append(token) # [S8]
        elif token == ')':
            # [S9]
            while len(stack) != 0 and stack[-1] != '(':
                out.append(stack.pop()) # [S10]
            stack.pop() # [S11]
        else:
            out.append(token) # [S12]
    while len(stack) != 0:
        # [S13]
        out.append(stack.pop())
    return out

if __name__ == '__main__':
    input = "( 1 + 2 ) * ( 3 / 4 ) ^ ( 5 + 6 )".split(" ")
    output = infixToRPN(input)
    print output
```

Related Posts via Categories

- RPN Calculator (using Scala, python and Java) (http://andreinc.net/2011/01/03/rpn-calculator-using-python-scala-and-java/)
- Credit Card Validation (Python 3.x) (http://andreinc.net/2011/04/09/credit-card-validation/)
- Insertion Sort Algorithm (http://andreinc.net/2010/12/26/insertion-sort-algorithm/)
- Bottom-up Merge Sort (non-recursive) (http://andreinc.net/2010/12/26/bottom-up-merge-sort-non-recursive/)
- The merge sort algorithm (implementation in Java) (http://andreinc.net/2010/12/22/the-merge-sort-algorithm-implementation-in-java/)
- ROT13 (Caesar Cipher) (http://andreinc.net/2010/10/05/rot13/)
- Implementing a generic Priority Queue in C (using heaps) (http://andreinc.net/2011/06/01/implementing-a-generic-priority-queue-in-c/)
- Bytelandian gold coins (http://andreinc.net/2011/02/01/bytelandian-gold-coins/)
- Sieve of Eratosthenes (finding all prime numbers up to a specific integer) (http://andreinc.net/2010/12/12/sieve-of-eratosthenes-finding-all-prime-numbers-up-to-a-specific-integer/)
- Serialize java objects using GZip streams (GZIPInputStream and GZIPOutputStream) (http://andreinc.net/2010/12/12/serialize-java-objects-using-gzip-streams-gzipinputstream-and-gzipoutputstream/)

algorithm (http://andreinc.net/tag/algorithm/) algorithms (http://andreinc.net/tag/algorithms-2/) calculator (http://andreinc.net/tag/calculator/)
conversion (http://andreinc.net/tag/conversion/) converter (http://andreinc.net/tag/converter/) gcalctool (http://andreinc.net/tag/gcalctool/)
how to (http://andreinc.net/tag/how-to/) infix (http://andreinc.net/tag/infix/) infix to rpn (http://andreinc.net/tag/infix-to-rpn/)
infix to rpn conversion (http://andreinc.net/tag/infix-to-rpn-conversion/) interpretor (http://andreinc.net/tag/interpretor/)
java implementation (http://andreinc.net/tag/java-implementation/) mathematical expressions (http://andreinc.net/tag/mathematical-expressions/)
notation (http://andreinc.net/tag/notation/) operators (http://andreinc.net/tag/operators/) programming challenges (http://andreinc.net/tag/programming-challenges-2/)
programming exercise (http://andreinc.net/tag/programming-exercise/) python (http://andreinc.net/tag/python/)
reverse polish notation (http://andreinc.net/tag/reverse-polish-notation/) RPN (http://andreinc.net/tag/rpn/)
shunting-yard algorithm (http://andreinc.net/tag/shunting-yard-algorithm/) spoj (http://andreinc.net/tag/spoj/)

17 thoughts to “Converting infix to RPN (shunting-yard algorithm)”

GOOD (HTTP://WWW.GOOGLE.COM)
November 1, 2010 at 8:18 am (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-14)

good

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=14#RESPOND)

Pingback: RPN Calculator (using Scala, python and Java) « andreINC.net (http://andreinc.net/2011/01/03/rpn-calculator-using-python-scala-and-java/)

JAI
January 9, 2012 at 12:01 am (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-98)

Thanks for a well-written and very informative post!

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=98#RESPOND)

TOM
July 29, 2012 at 5:52 pm (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-128)

HI,
I have a C# implementation of this algorithm. If you are interested and want to publish, let me know and I will email you the code.

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=128#RESPOND)

LOGAN
August 2, 2013 at 5:15 pm (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-464)

Hi,

I’m really interested by your code in C. If you could send me it on my mail it will be really awesome : djapris@live.fr (mailto:djapris@live.fr)

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=464#RESPOND)

ANDREI CIOBANU
July 29, 2012 at 9:05 pm (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-129)

@Tom

Of course you can email me the code if you want, and I’ll publish the code (ofcourse giving you the credits).

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=129#RESPOND)

PAUL NEVE
August 1, 2012 at 2:12 pm (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-130)

Here’s a Javascript implementation ripped straight from your Java one. Scouring through t’interwebs there are previous little Javascript implementations that are any good out there, so hopefully this will be useful to others and save them the couple of hours I spent first looking for an existing one then hacking this one together.

```
// implementation of shunting yard
// converted from the Java version at http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/
function infixToRPN(expression)
{
    var LEFT_ASSOC = 0;
    var RIGHT_ASSOC = 1;
    var OPERATORS = {};
    OPERATORS['+'] = [ 0, LEFT_ASSOC ];
    OPERATORS['-'] = [ 0, LEFT_ASSOC ];
    OPERATORS['*'] = [ 5, LEFT_ASSOC ];
    OPERATORS['/'] = [ 5, LEFT_ASSOC ];
    OPERATORS['%'] = [ 5, LEFT_ASSOC ];
    OPERATORS['^'] = [ 10, RIGHT_ASSOC ];

    function isOperator(token)
    {
        if (OPERATORS[token]) return true;
        return false;
    }

    function isAssociative(token,type)
    {
        if (!isOperator(token)) {
            throw new Error("Invalid token: " + token);
        }
         if (OPERATORS[token][1] == type) {
            return true;
        }
        return false;
    }

    function cmpPrecedence(token1, token2)
    {
        if (!isOperator(token1) || !isOperator(token2)) {
            throw new Error("Invalid tokens: " + token1
                    + " " + token2);
        }
        return OPERATORS[token1][0] - OPERATORS[token2][0];
    }

    function peek(array)
    {
        if (array.length > 0) return (array[array.length-1]);
        return null;
    }

    // for now, split at space
    var inputTokens = expression.split(" ");

    var out = [];
    var stack = [];

    // For all the input tokens [S1] read the next token [S2]
    for (var i = 0; i  0 && isOperator(peek(stack)))
            {
                // [S4]
                if ((isAssociative(token, LEFT_ASSOC) && cmpPrecedence(
                    token, peek(stack)) <= 0)
                || (isAssociative(token, RIGHT_ASSOC) && cmpPrecedence(
                    token, peek(stack))  0 && peek(stack) != "(")
            {
                out.push(stack.pop()); // [S10]
            }
            stack.pop(); // [S11]
        }
        else
        {
            out.push(token); // [S12]
        }
    }

    while (stack.length > 0)
    {
        out.push(stack.pop()); // [S13]
    }
    return out;

}
```

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=130#RESPOND)

ANDREI CIOBANU

November 5, 2012 at 12:43 am (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-155)

Thanks for your comment and implementation. I've almost deserted this site. Recently I've decided to "revive" it, so that's the reason for this very late reply.

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=155#RESPOND)

Pingback: » AdvancedFilter: Parsing the Search String Spreadsheet Budget and Consulting (http://www.spreadsheetbudget.com/2012/08/06/advancedfilter-parsing-the-search-string/)

Pingback: Memorized my reading ;) « Parnurzeal's Weblog (http://parnurzeal.wordpress.com/2011/09/22/memorized-my-reading/)

VISHAL
January 2, 2013 at 10:22 am (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-436)

How will this algorithm handle a ternary operator? E.g. a?b:c

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=436#RESPOND)

ANDREI CIOBANU (HTTP://WWW.ANDREINC.NET)
January 2, 2013 at 11:50 am (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-437)

Right now it will not handle a ternary operator :).

You are basically trying to implement a “if then else” statement, and I am not very sure this algorithm was made for this.

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=437#RESPOND)

Pingback: Interesting sites | Latoto (http://latoto.cz/interesting-sites/)

SRINIVAS R
August 8, 2013 at 11:49 am (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-466)

Looking forward to the If Else implementation. Any Plans?..

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=466#RESPOND)

NIKOS M. (HTTP://NIKOS-WEB-DEVELOPMENT.NETAI.NET)
December 5, 2014 at 10:33 am (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-565)

The if-then-else, can be implemented as a 3-ary usual (function) operator, i.e “inline if”, provided the RPN implementation can parse (custom) functions

There was a similar project implementation as a sub-project for another project i was working on, some time ago, but decided to build a generic xpresion tool (in various platforms, currently js,python,php,actionscript) with variables and custom functions/operators support

here:

https://github.com/foo123/Xpresion (https://github.com/foo123/Xpresion)

Although it can be already covered by existing implementation, thought if adding, custom infix operators as well (additonaly to custom prefix/postfix operators, aka functions)

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=565#RESPOND)

NOBILIS
August 27, 2015 at 3:34 pm (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-840)

Hey, just wanted to say thanks for the implementation you’ve posted, it has been of tremendous help (specifically the Python version).

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=840#RESPOND)

TOTO (HTTP://TOTO-SHARE.COM)
April 14, 2016 at 10:17 pm (http://andreinc.net/2010/10/05/converting-infix-to-rpn-shunting-yard-algorithm/#comment-943)

Thank you very much for your tutorial. I am learn about shunting yard algorithm, your tutorial very help me to understanding this method

REPLY (HTTP://ANDREINC.NET/2010/10/05/CONVERTING-INFIX-TO-RPN-SHUNTING-YARD-ALGORITHM/?REPLYTOCOM=943#RESPOND)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Are we human, or are we dancer *

four × three =

POST COMMENT

Generic data structures in C (http://andreinc.net/2010/09/30/generic-data-structures-in-c/)

ROT13 (Caesar Cipher) (http://andreinc.net/2010/10/05/rot13/)

CATEGORIES

ARCHIVES

Select Month

sparkling Theme by Colorlib (http://colorlib.com/) Powered by WordPress (http://wordpress.org/)