An Exhaustive Review for Infix to Postfix Conversion with Applications and Benefits

Rohit Rastogi
Sr. Asst. Professor,
CSE Dept.ABES Engg. College,
Ghaziabad (U.P.), INDIA.
Email Id:rohit.rastogi@abes.ac.in

Pinki Mondal
ABES Engg. College, Ghaziabad (U.P.), INDIA, **Email Id:**
pinki.pinki.mondal8@gmail.com

Kritika Agarwal
ABES Engg. College, Ghaziabad (U.P.), INDIA
Email Id:kritika.12it009@abes.ac.in

Abstract-In Computer Science, Reverse Polish notation has simplified calculations and has benefitted a new face to technology. Since 1960, RPN is used in calculators because its implementation is very easy and simple as well as it gives good performance. In this paper, we have introduced a new approach for infix to postfix expressions conversion by following some rules and highlighted some of the applications and advantages of existing methods. The proposed algorithm differs from other existing algorithms through its human readability. We have discussed a comparative analysis of RPN, Shunting Yard algorithm and a new approach to Post fix conversion named as PKR algorithm.

Keywords–Infix Notation, Postfix Notation, Data Structure, Keystrokes.

I. INTRODUCTION

Reverse Polish Notation is an algorithm for representing expressions in which the operator is placed after the arguments/operands being operated on. It was invented in the 1920s by Polish mathematician Jan Lukasiewicz [1].
Second method is Shunting Yard algorithm which was given by Edsger Dijkstra. Shunting-Yard algorithm is an algorithm in which parsing of mathematical expression is done on specified infix expression. It produces output either in RPN or as an AST. This algorithm performs its operations on stack and queue. Stack holds operators in it and output is added into the Queue. The generalized form of shunting-yard algorithm is operator-precedence parsing [2].

II. INFIX TO POST-FIX NOTATION

Infix notation is an arithmetic and logical notation which represents the operator placed between two operands. It is not easy to parse by the computer. In infix notation, the order of operations is mandatory to indicate and operands and operators must surrounded by parentheses [3].
Post-fix notation is a mathematical notation which is used to parse a machine. Post-fix notation is also called Reverse Polish notation (RPN) and it involves operands followed by operators. In RPN, use of parentheses is not necessary for calculations [1].

III. WHY CONVERSION IS NEEDED FROM INFIX TO POST-FIX NOTATION?

Unlike Infix expression, RPN has the property that brackets are not required to represent the grouping of the terms or order of evaluation [4]. Post-fix expression can be easily obtained by push - pop operation on STACK. This greatly simplifies the expression's computation within computer programs [5]. The big advantage of RPN is that it is extremely easy and fast, for a computer to analyze. Infix notation is easier to read for humans, whereas a machine can easily parse postfix notation. In postfix expression we can obtain the original parse tree without original knowledge, but the same is not applied to infix expressions.

IV. HISTORICAL IMPORTANCE AND APPLICATION AREAS

A. History of Implementations

Hewlett-Packard Engineers designed first calculator which used RPN, HP9100A in 1968 which is regarded as Desktop Calculator [12]. Second is HP-35, which is the first handheld scientific calculator in world, used RPN in 1972 [8]. HP introduced LCD-based calculator in the 1980s such as HP 10C, HP 11C, HP 15C, HP 16C and HP 12C which is famous financial calculator. From 1990 to 2003, RPN calculators with graphing includes HP-48 series and in 2006 HP-50g manufactured. In 2011, HP manufactured 12C, 12C platinum, 17 BII, 20 B (financial), 30 B (business), 33S, 35S, 48 GII and 50G (scientific) [10].
PROGRAM is Programmable Scientific Calculator which was developed by Prinztronic brand. Soviet programmable calculators such as MK-52, MK-61, B3-34 and earlier B3-21, used RPN for both automatic mode and programming [11].

B.Current Implementations

Exiting implementations using RPN include:

- Stack oriented programming languages like Forth, Factor, and PostScript page description.

- Hardware calculators include HP science/ engineering, business/ finance and Semico calculators.
- Software calculators include Mac OS X calculator, Apple's "reverse polish notation calculator", Android's "RealCalc", UNIX system calculator program dc, etc.

V. EXISTING ALGORITHMS TO CONVERT INFIX EXPRESSIONS INTO POSTFIX EXPRESSIONS

(a). Reverse Polish Notation Algorithm:

RPN (I, P)

Suppose I is the Infix notation arithmetic expression. This algorithm gives Postfix expression P.

```
1.  Push "("onto Stack and add ")" to the end of I.
2.  Scan the expression I from Left to Right and repeat from
    step number 3 to 6 for every element of I until stack is
    empty :
3.  If an operand is read, add it to P.
4.  If a left parenthesis is read, push onto Stack.
5.  If an operator, say $, is read, then :
(a) Repeatedly, each operator is added to P by popping from
    Stack which has the same/higher precedence than the
    operator encountered $.
(b) Add $ to Stack.
6.  If a right parenthesis ")" is read, then:
(a) Repeatedly, each operator is added to P by popping from
    Stack until a left parenthesis "("is read.
(b) Remove left parentheses "(". [Do not add it to P].
7.  Exit.
```

Numerical Example:

I: A + (B * C - (D / E ^ F) * G) * H

Step 1: A + (B * C - (D / E ^ F) * G) * H, read expression from left to right.

TABLE I. CONVERSION OF INFIX TO POSTFIX EXPRESSION BY USING RPN ALGORITHM.

Symbol Scanned		STACK	Expression P
(1)	**A**	(	**A**
(2)	**+**	(+	**A**
(3)	**(**	(+ (	**A**
(4)	**B**	(+ (	**AB**
(5)	*****	(+ (*	**AB**
(6)	**C**	(+ (*	**ABC**
(7)	**–**	(+ (-	**ABC***
(8)	**(**	(+ (- (	**ABC***
(9)	**D**	(+ (- (	**ABC*D**
(10)	**/**	(+ (- (/	**ABC*D**
(11)	**E**	(+ (- (/	**ABC*DE**
(12)	**^**	(+ (- (/ ^	**ABC*DE**
(13)	**F**	(+ (- (/ ^	**ABC*DEF**
(14)	**)**	(+ (-	**ABC*DEF^/**
(15)	*****	(+ (- *	**ABC*DEF^/**
(16)	**G**	(+ (- *	**ABC*DEF^/G**
(17)	**)**	(+	**ABC*DEF^/G*-**
(18)	*****	(+ *	**ABC*DEF^/G*-**
(19)	**H**	(+ *	**ABC*DEF^/G*-H**
(20)	**)**	**EMPTY**	**ABC*DEF^/G*-H*+**

(b).Shunting-yard algorithm [2]

Algorithm

Postfix expression is added to output QUEUE. Consider there is an Infix expression then the algorithm says that:
Read a token.

```
1.1. If it is an operand, then add it to the Queue.
1.2. If it is an operator, then push it onto the Stack.
1.3. If it is an argument separator (e.g., a comma):
     (a) Then add operators to Queue from Stack until a
         left parenthesis is at the top of Stack. If no left
         parenthesis is there, either parentheses were
         mismatched or separator was misplaced.
1.4. If it is an operator, $, then:
     (a) while there is an operator, #, at the top of the
         Stack, and
         1.4.a.1. either $ is left-associative and its
                  precedence is less than or equal to that
                  of #, or $ has low precedence than that
                  of #, then
                  1.4.a.1.1. pop # off the Stack, onto the
                             output Queue;
                  1.4.a.1.2. Push $ onto the Stack.
1.5. If it is a "("left parenthesis, then push it to Stack.
1.6. If it is a ")"right parenthesis:
```

(a) Pop operators from Stack to Queue until left parenthesis is read at the top of Stack.
(b) The left parenthesis is popped but not added to the Queue.
(c) If the token is operator, pop it to Queue.
(d) If no left parenthesis is inside the top of Stack, then there are mismatched parentheses.

1.7. When no other token left to read:
    (a) While there are operators in the Stack
        1.7.a.1. If operator is a parenthesis, then mismatched parentheses error is there.
        1.7.a.2. Pop the operator onto Queue.
1.8. Exit.

Numerical Example on Shunting-Yard Algorithm

Consider the Infix Expression: A + (B * C - (D / E ^ F) * G) * H

Transform given in-fix into its post-fix expression in OUTPUT QUEUE.

Step 1: A + (B * C - (D / E ^ F) * G) * H, read expression from left to right.

TABLE II. CONVERSION OF INFIX TO POSTFIX EXPRESSION BY USING SHUNTING-YARD ALGORITHM.

Token		Action	Output (in RPN)	Operator Stack	Notes
(1)	A	Add A to output	A		
(2)	+	Push + to stack	A	+	
(3)	(	Push (to stack	A	+ (	
(4)	B	Add B to output	AB	+ (	
(5)	*	Push * to stack	AB	+ (*	
(6)	C	Add C to output	ABC	+ (*	
(7)	–	Pop stack to output Push - to stack	ABC*	+ (-	- has less precedence than *
(8)	(	Push (to stack	ABC*	+ (- (	
(9)	D	Add D to output	ABC*D	+ (- (	
(10)	/	Push / to stack	ABC*D	+ (- (/	
(11)	E	Add E to output	ABC*DE	+ (- (/	
(12)	^	Push ^ to stack	ABC*DE	+ (- (/ ^	
(13)	F	Add F to output	ABC*DEF	+ (- (/ ^	
(14)	)	Pop stack to output Pop stack	ABC*DEF^/ ABC*DEF^/	+ (- (+ (-	Repeated until "(" found Discard matching parenthesis
(15)	*	Push * to stack	ABC*DEF^/	+ (- *	* has high precedence than -
(16)	G	Push G to stack	ABC*DEF^/G	+ (- *	
(17)	)	Pop) to output Pop stack	ABC*DEF^/G*- ABC*DEF^/G*-	+ (+	Repeated until "(" found Discard matching parenthesis
(18)	*	Push * to stack	ABC*DEF^/G*-	+ *	
(19)	H	Add H to output	ABC*DEF^/G*-H	+ *	
(20)	*End*	Pop entire stack and add to output	ABC*DEF^/G*-H*+		

VI. OUR PROPOSED PKR APPROACH FOR TRANSFORMING INFIX TO POSTFIX EXPRESSIONS

PKR algorithm uses either of the two rules to compare two operators**:**

1. BEMD%AS refers to Bracket Exponent Multiply Divide Modulus Addition Subtract

←Yes if order of operator results in Yes then operator can stay inside stack

No → if order of operator results in No then operator which is pushed first, is popped and added to output expression.

2. BNAO refers to Bracket NOT (Binary) AND (Binary) OR (Binary)

←Yes if order of operator results in Yes, the operator can stay inside stack

No → if order of operator results in No, the operator which is pushed first, is popped and added to output expression.

Algorithm:

Let P is Infix arithmetic expression. This algorithm will give the equivalent Post-fix expression in OUTPUT.

1. SCAN P from Left to Right, the encountered symbol is pushed onto STACK and if
   (a) Operand encountered, add it to OUTPUT.
   (b) Operator encountered, push onto STACK.
   (c) If open parenthesis encountered "(", push onto STACK.
2. If closed parenthesis ")" encountered then pop operators in LIFO order and add it to OUTPUT and it will also cancel the "(" opening parenthesis.

3. If two operators encountered continuously the follow BEMD%AS rule -
   (a) If sequence of operators is OCCURING from Right to Left in BEMD%AS then no popping.
   (b) If sequence of operators is OCCURING from Left to Right then pop the operator which is inserted first onto OUTPUT.
   (c) Same operators cannot stay together in STACK so pop the operator which is pushed first onto OUTPUT.
4. If more than two operators are encountered in STACK then repeat Step-3 on the two topmost Operators present in STACK.
5. Exit.

Numerical Example on PKR approach

Consider the same Example: P: A + (B * C - (D / E ^ F) * G) * H

Transform P into equivalent post-fix expression in OUTPUT.

Step 1: A + (B * C - (D / E ^ F) * G) * H, read the expression from left to right.

TABLE III. CONVERSION OF INFIX TO POSTFIX EXPRESSION BY USING PKR ALGORITHM.

Scan	STACK	Action	OUTPUT
(1) A		**POP**	**A**
(2) +	**+**	**PUSH**	**A**
(3) (	**+ (**	**PUSH**	**A**
(4) B	**+ (**	**POP**	**AB**
(5) *	**+ (***	**PUSH**	**AB**
(6) C	**+ (***	**POP**	**ABC**
(7) -	**+ (* -**	**PUSH**	**ABC**
APPLY RULE	**←yes BEMD%AS rule no→**	**NO ⇨ Pop (*)**	
	+ (-	**POP**	**ABC***
(8) (	**+ (- (**	**PUSH**	**ABC***
(9) D	**+ (- (**	**POP**	**ABC*D**
(10) /	**+ (- (/**	**PUSH**	**ABC*D**
(11) E	**+ (- (/**	**POP**	**ABC*DE**
(12) ^	**+ (- (/ ^**	**PUSH**	**ABC*DE**
APPLY RULE	**^ has high priority than /**	**YES ⇨ NOP**	
(13) F	**+ (- (/ ^**	**PUSH**	**ABC*DEF**
(14)			
(15))	**+ (- ***	**POP**	**ABC*DEF^/**
(16) *	**+ (- ***	**PUSH**	**ABC*DEF^/**
(17) G	**+ (- (/ ^**	**POP**	**ABC*DEF^/G**
APPLY RULE	**←yes BEMD%AS rule no→**	**YES ⇨ NOP**	
(18))	**+**	**POP**	**ABC*DEF^/G*-**
(19) *	**+ ***	**PUSH**	**ABC*DEF^/G*-**
APPLY RULE	**←yes BEMD%AS rule no→**	**YES ⇨ NOP**	
(20) H	**+ ***	**POP**	**ABC*DEF^/G*-H**
(21)	**EMPTY**	**POP**	**ABC*DEF^/G*-H*+**

VII. ANALYSIS BASED ON SEVERAL PARAMETERS

(a).Performance Analysis:

TABLE IV. PERFORMANCE ANALYSIS OF EXISTING ALGORITHMS AND PKR APPROACH

S No.	Name of the Algorithm	Number of Iterations Required	Number of the Steps needed (according to the above taken infix expression)	Complexity Discussion
1.	Reverse Polish Notation (RPN)	(as per the length of string) n	20	Each token of infix expression will be parsed exactly once.

2.	Shunting Yard Algorithm	(as per the length of string) n	20	Each token of infix expression will be parsed exactly once.
3.	Our PKR Approach	(as per the length of string) n	20	Each token of infix expression will be parsed exactly once.

(b).Comparison Analysis:

TABLE V. COMPARISON ANALYSIS OF EXISTING ALGORITHMS AND PKR APPROACH.

Name of the Algorithm	**Data Structure**	**Space and Time Complexity**	**Nature (P/NP/NP-hard /NP-Complete) And Type (Recursive/Non- Recursive)**	**Result Display (For a particular expression) Input → Infix Expression Result → Postfix Expression**
Reverse Polish Notation(RPN)	Stack and array or queue	Space- θ(n) Time- θ (n)	P-Time / Non-Recursive	ABC*DEF^/G*-H*+
Shunting Yard Algorithm	Stack and queue	Space- θ (n) Time- θ (n)	P-Time / Non-Recursive	ABC*DEF^/G*-H*+
Our proposed PKR Approach	Stack and array or queue	Space- θ (n) Time- θ (n)	P-Time / Non-Recursive	ABC*DEF^/G*-H*+

VIII. PERFORMANCE

Thus, the Running-time Complexity is θ (n) and Space Complexity is also θ (n) for the Algorithm. Thus we can say that our approach gives linear performance as well as it takes less memory space for conversion.

IX. ALGORITHM OPTIMIZATION

Since there are many high level languages like C, C++, Java and .Net, which can be used to optimize this procedure as per their optimization techniques.

X. BENEFITS

This approach is user friendly as it reduces the complexity of human effort. Because we just have to remember the rules to solve the expressions and it requires very less efforts. This whole review will help us to add better understanding of technical concept of Infix to postfix conversion. People can take decisions to opt better algorithms for their target operations.

This is a very efficient way to compute complicated calculations of expressions very easily with less number of keystrokes, by using this procedure [7]. This approach is convenient as it has readability for humans, especially when there is a need to solve large expressions, and it is also very convenient representation for machines.

XI. RECOMMENDATIONS

The proposed PKR's conversion method is not only optimized and easier method for infix to postfix conversion but also helpful for computation of expression by computer programs and it is human friendly too. So, this algorithm can be used for easily implementation and evaluation of postfix expressions and also in areas like performing calculation, parsing, logic, linguistics and lexical analysis [9].

XII. LIMITATIONS

This paper discussed about proposed algorithm on the basis of few expressions only. Though our method is simple but technically, it is limited to only postfix conversion. Also our algorithm does not include many logical and other operators. And proposed algorithm cannot compare logical and arithmetic operators together.

XIII. GENERAL FUTURE SCOPE

The above proposed algorithm can be implemented by using several Programming languages available like C, C++, Java and .Net as per their optimization policies so as to optimize. We can bring new methods in which we can invent hybrid of two data structure so as to make it simpler in implementation in computers, thus making it computer readable. And it can also be represented as a rooted tree like Abstract Syntax Tree, used in parsing in compilers.

XIV. CONCLUSION

This paper gives a new method for Infix to Postfix conversion by applying two simple rules on infix expression which is highly human readable and the time and space complexity is also linear, simpler to execute. In addition to this, our approach also simplifies the conversion of expression using computer program as we have to feed only two conditions to compare operators. Hence computer complexity is also reduced to an extent. Thus the proposed algorithm is simple and it can be useful in theory and practical implementation because it is human as well as computer optimized.

XV. NOVELTY IN OUR PAPER

This paper not only presents the new method to convert the infix to postfix expression in a simple way but also gives defined set of rules which is easily understandable and remembered by the people. As Reverse Polish Notation was very influential in the early days of computing so as our approach's implementation on computer is very easy that it requires minimal demands on memory [6].

ACKNOWLEDGEMENT

The completion of this paper would not have been successful without the help and guidance of our respected HOD Prof. Dr. R. Radhakrishnan who was always there to support us and correct us at every step. We would like to thank Mr. Rajat Sharma of CSE-IV year who helped us on this work and who has given this idea. Next, we would like to show our gratitude towards family and friends and our college ABES EC for their motivation, encouragement and providing us with great resources. Last but definitely not the least we would thank the almighty god without whose grace this paper would not have achieved success.

REFERENCES

[1] Reverse Polish notation - Wikipedia, the free encyclopedia http://en.wikipedia.org/ wiki/ Reverse_Polish_notation

[2] Shunting-yard algorithm - Wikipedia, the free encyclopedia http://en.wikipedia.org/ wiki/ Shunting_yard_algorithm

[3] Infix notation - Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/ Infix_notation

[4] Burks, A. W.; Warren, D. W.; Wright, J. B. (1954). "An Analysis of a Logical Machine Using Parenthesis-Free Notation". Mathematical Tables and Other Aids to Computation 8 (46): 53. doi:10.2307/2001990. JSTOR 2001990

[5] "Reverse Polish Notation.", From MathWorld -A Wolfram Web Resource, created by Eric W. Weisstein.

[6] Reverse Polish Notation_10 Innovations http://www.10innovations.unsw.edu.au/reverse-polish-notation

[7] RPN or DAL? - The X-Number Way, www.xnumber.com/xnumber/rpn_or_adl.htm

[8] Computer Science beta, http://cs.stackexchange.com/questions/4666/what-is-the-significance-of-reverse-polish-notation

[9] The Future of RPN Calculators, http://hardware.slashdot.org/story/04/06/03/2122226/the-future-of-rpn-calculators

[10] HP Calculators

[11] Elektronika B3-21 page on RSkey.org

[12] Kasprzyk, D. M.; Drury, C. G.; Bialas, W. F. (1979), "Human behaviour and performance in calculator use with Algebraic and Reverse Polish Notation", Ergonomics 22 (9): 1011, doi: 10.1080/00140137908924675.

AUTHORS' PROFILE

Mr. Rohit Rastogi received his B.E. degree in Computer Science and Engineering from C.C.S.Univ. Meerut in 2003, the M.E. degree in CS from NITTTR-Chandigarh (National Institute of Technical Teachers Training and Research-affiliated to MHRD, Govt. of India), Punjab Univ. Chandigarh in 2010. He was Asst. Professor at IMS College, Ghaziabad in computer Sc. Dept.

He has authored/co-authored, participated and presented research papers in various Science and Management areas in around 40 International Journals and International conferences including prestigious IEEE and Springer and 10 national conferences including SRM Univ., Amity Univ., JP Univ. and Bharti Vidyapeetha etc.

He has guided 5 ME students in their thesis work and students of UG and PG in around 100 research papers. He has developed many commercial applications and projects and supervised around 30 B.E. students at graduation level projects.

At present, he is a Sr. Asst. Professor of CSE Dept. in ABES Engineering. College, Ghaziabad (U.P.-India), affiliated to Uttar Pradesh Tech. University, Luck now.

His research interests include Data ware Housing and Data Mining, DAA, TAFL and DBMS.

At present, He is engaged in Clustering of Mixed Variety of Data and Attributes with real life application applied by Genetic Algorithm, Pattern Recognition and Artificial Intelligence. Also, He is preparing some interesting algorithms on Swarm Intelligence approaches like PSO, ACO, and BCO etc.

Ms. PINKI MONDAL has done her High School from Leelawati Public School and Intermediate from Shree Thakur Dwara Balika Vidhalaya (CBSE Board), Ghaziabad. Presently she is pursuing B.Tech. in Computer Science and Engineering form ABES Engineering College, Ghaziabad affiliated to Uttar Pradesh Technical University. Her field of interest includes coding in languages like C and Java and subjects like DBMS, DLD and DAA.

Ms. KRITIKA AGARWAL has done her schooling from Holy Child School (ICSE Board) and Intermediate from GSM Senior Sec. School (CBSE Board), Ghaziabad. Presently she is pursuing B.Tech. in Computer Science and Engineering form ABES Engineering College, Ghaziabad affiliated to Uttar Pradesh Technical University. Her field of interest includes coding in languages like C, C++ and Java and subjects like DBMS, Data Structures and DAA.