Transfer yard Algorithm: Novel mathematical infix to postfix expression evaluator with minimum stack operations

Omar H Abu El Haijaa[1], Ahmad Al-Jarrah[2] Mohammmad A. Al-Jarrah[1,3]

[1]Yarmouk Univeisty, Irbid, Jordan
[2]Applied Science Department, AlBalqa Applied University, Jordan
[3]Arab Open Univeristy, Dammam, KSA

Abstract

Reverse Polish Notation (RPN) is vital in evaluating machines' mathematical expressions. Further, it has many important applications. For more than 50 years, RPN Shunting Yard was utilized to evaluate mathematical expressions written infix notation. Most resources recommend using the Shunting yard algorithm to convert infix to RPN notation. In this paper, we proposed a variant shunting yard algorithm named Transfer Yard algorithm (TY) that transfers infix expression to RPN. The proposed algorithm has the advantage of using an array structure with minimum stack operations. The array structure is proved to have better performance in compassion of utilizing stack memory. Actually, utilizing array structure benefits of random access. In the proposed algorithm, we utilized the array to arrange operators' precedency so we can perform transformation in an efficient way. We implemented the proposed algorithm and compared its performance with the Shunting Yard algorithm. To achieve a highly accurate comparison, we designed experiments that minimize any artifact that would affect the results. For that purpose, we repeated the experiment in the same environment more than thousand time and selected the lowest time to represent the execution time for those inputs. Furthermore, both algorithms are tested with variable inputs. Results show that the proposed transfer yard algorithm's performance is significantly better than the performance of the Shunting Yard algorithm.

Keywords: Transfer Yard Algorithm; Shunting Yard Algorithm; Reverse Polish notation (RPN); infix notation; postfix notation.

1. Introduction

The mathematical expression is standard in programming language. Each programming language contains an expression or string. The mathematical expression has three formats: infix expression, postfix expression, and prefix expression. For example, the (+ a b) is a prefix; A B + is a postfix; A + B is an infix expression. Infix expression notation is a human-readable format but postfix and prefix is machine-readable format. The machine can easily evaluate the result according to this prefix and postfix format. The vision of this research is to evaluate the infix expression without parenthesis in mathematical postfix format with the minimum number of stack operations. Consider the following expression as an example with a polynomial expression of degree 2.

The problem of evaluating mathematical expressions is a fundamental concern of computer science. There lies a great importance of efficiently evaluating the mathematical expressions in a computer program. As computer modeling and simulation became prominent, there is a huge

extent of dealing with mathematical equations in the computer program. Inmost of the scientific computation algorithms, initial estimates of the variables are iteratively refined to reach a solution to the formal mathematical problem. Mathematically complicated initial guess estimates are very common during the simulation, and there is usually a number of repeated guesses. This is also visible in most of the data analysis procedures using least square methods. Finally, a computer needs to evaluate complicated expressions to generate such variables. To evaluate the expression contained big mathematical equations, a simple and efficient evaluator is desired.

Mathematical expressions evaluation, simplification, and transferring between different notations are fundamental in many applied and theoretical sciences. With many wonderful applications and wide popularity in the scientific community, Jan Lukasiewicz [1] invented Polish notation (PN) in 1924, which is commonly known as a prefix notation (Also called Polish prefix notation, Normal Polish Notation, Lukasiewicz notation, and Warsaw notation). Reverse polish notation (RPN) which is another name for polish postfix notation or postfix notation, becomes a complementary part of many computer science textbooks [2]–[6]. Moreover, it becomes the core of many technological applied techniques, such as scientific calculators, compilers, numerical methods, geometry, bioinformatics, genetic programming rules, etc [37-40]. Reverse Polish Notation (RPN) is a bracket-free notation that deduces the operator's presidency problem, makes expressions easier to be evaluated by machine, and proves excellent advantages over most other notations.

Postfix solves the precedency problem. Therefore, comparing to infix notation, postfix notation is the middle way to final computation. Moreover, we can develop a straightforward computer algorithm to evaluate postfix expressions. On the other hand, infix notation was traditionally used in mathematics and all other scientific fields. Actually, Infix notation is closer to human logical thinking. Therefore, it leads to the necessity of having an excellent algorithm to convert infix notations to postfix notation and vice versa. The conversion between notations is vital to fill the gap between human logic and machine simplified computation. Table 1 shows examples of expressions written using infix notation and their equivalence expression in postfix notation.

Table 1: Mathematical expressions written in infix and the equivalent postfix.

Infix	Postfix (RPN)
3 + 5 * 2 – (6/3)	3 5 2 * + 6 3 / -
5 + 4 * 3 / (1 – 5) ^ 2 ^ 3	5 4 3 * + 1 5 – 2 3 ^ ^ / +
4 * (5 + 4/2) + 3^2	4 5 4 2 / + * 3 2 ^ +

Shunting yard algorithm [7, 8] is one of the most populor solutions that convert infix expressions to postfix ones. It converts mathematical expression utilizing one stack and one queue. Shunting yard algorithm relies on stack operations (push and pop) heavily. Indeed, it performs many stack operations to accomplish infix to postfix conversion, where stack operation consumes many machine cycles and extra memory space. Thus, micro machines with limited memory can convert a limited size of mathematical expression.

The mathematical shunting yard algorithm parses numbers and operators in the form of infix expression to postfix expression with operator precedence. This grammar has extra stack operations to handle parentheses in this unoptimized form. The existing approach evaluates the mathematical expression which handles polynomial and matrix without operator precedence. The existing postficaptor contains a small gap of stack operations. The investigator will design a new mathematical infix to the postfix expression evaluator with minimum stack operations which can expand the mathematical expression in postfix easily. The mathematical grammar of this proposed approach can handle the polynomial and matrix format expression without operator precedence. The mathematical postficaptor contains two innovative ideas such as transfer yard and empty parentheses. Empty parentheses is employed to eliminate excess stack-push operation in shunting-yard algorithm and transfer yard is utilized to change shunting-yard grammar without parentheses. The designed approach is post-fix equivalent of the existing shunting yard which can ensure correctness [41].

In programming, a set of rules (grammar) is used to define a language. There are many programming languages, text based scripting languages (PHP, Perl, JavaScript) or markup languages (HTML, XML). Some languages are tree based visual languages (Scratch, IBasic, AppInventor) where programming is done by creating and assembling blocks together. All these languages consist of a set of valid tokens. Instructions can be composed using these tokens. However, the composition must comply with the rules defined by the grammar. When a code is written in a programming language, it is usually first checked in the grammar according to a given set of rules. If the code complies with the grammar rules, it is translated into machine code necessary to be executed. Otherwise, a compilation error is produced, listing unexpected tokens and expected ones.

Input expressions are often given in infix format where operators are situated between operands: A + B. The Operator Precedence is determined by a precedence hierarchy defined in the set of grammar rules. Output expressions may be given in postfix format where operators are situated after their operands: I: A B +. It is common to use operator stack stack for parsing infix expressions. On the parsing process, the stack saves operators and parenthesis. Output stack saves postfix form operands and their operators. There are different implementations of the algorithm with and without operator stack and with or without output stack. Some implementations are more efficient than others according to a defined usage scenario. The one with the output stack is the most memory demanding approach [41].

2. Related works

Since Shunting yard algorithm was appeared [8], many enhancements and extensions were suggested [9, 10]. Further, the algorithm and its enhancements were utilized in many applications and fields such as PKR by Rastogi et al. [9]. The PKR algorithm applies two additional rules for operators' comparison, where accordingly, the algorithm decides to keep the operator inside the stack or pop it out to the output. For example, assume that we have the following operation as part of an expression (… A * B - C ….), after pushing the subtract operator to the stack, the algorithm scans the last two items in the stack where they are '-' and '*',. Then, according to the first rule, the algorithm pops and implements multiplication. Krtolica et al. extend shunting yard algorithm

usability by few steps to make shunting yard algorithm accepting transferring comma-separated multi parameters functions [10]. Tiwari et al. introduce a multi-threaded algebra system, where they use RPN and shunting yard in the expression evaluation part [11].

Many applications, across various industries, heavily rely on the Reverse Polish Notation (RPN) and the Shunting Yard algorithm as fundamental components of their computational processes[12]–[17]. These algorithms have proven to be incredibly efficient and reliable for carrying out complex mathematical operations and logical evaluations.

One of the earliest instances of the RPN being utilized was in the first calculator ever designed by Hewlett-Packard in 1968[18]. This groundbreaking calculator ingeniously employed the RPN methodology to effortlessly perform all the necessary calculations. Shortly after, in 1972, Hewlett-Packard released the remarkable HP-35, a handheld scientific calculator that revolutionized the field[4]. The HP-35 is widely regarded as the first portable scientific calculator.

The significance of RPN extends far beyond calculators. It plays a crucial role in the field of computer science, specifically in compilers. Compilers, which are essential tools for converting high-level programming languages into machine code, often adopt the RPN as an initial step to transform infix notations into a more manageable format. This conversion allows for smoother processing and optimization of the code.

RPN's versatility is further exemplified in its application within the Postscript language. Postscript, a programming language primarily used for describing and rendering text documents, incorporates RPN as a key component. This integration enables the representation of complex document structures and enhances the overall readability and understandability of the syntax.

Beyond computer science and software engineering, RPN has also found its place in business domains. Avarm[19], a notable authority in the field, proposes a comprehensive approach to formalizing business rules using computational formulas. In this context, both RPN and XML are employed as foundational principles, ensuring that these formulas are intelligible to both humans and machines. This exemplifies how RPN can bridge the gap between human-readable and machine-readable representations, offering a practical solution for various business applications.

In conclusion, the extensive utilization of the Reverse Polish Notation (RPN) and the shunting yard algorithm spans across a wide range of applications and industries. From calculators and compilers to programming languages and business domains, RPN consistently serves as a robust and efficient methodology for performing complex operations and facilitating clear communication between humans and machines. Its time-tested reliability and versatility continue to make it an indispensable tool in the ever-evolving world of computing and technology.

Furthermore, Avarm is conducting research to explore the neural underpinnings of how hierarchical structures are constructed at different levels of hierarchy within the cognitive domains of language and mathematics. RPN (Reverse Polish Notation) is also being used in cognitive brain sciences. For example, Makuuchi et al. are employing RPN to elucidate the precise nature of mental arithmetic [20].

Furthermore, biology simulation and bio-inspired technologies are other examples of reverse polish notation applications. Reverse polish notation conditional expressions have been used in each production role, and Lindenmayer systems inspired a new method to develop genetics

generated by neural networks; Neural networks are used to control robots of spiders to follow compass [21]. Dhenakaran studied RPN utilization in Cryptography using RPN [22]. Dhenakaran's work suggested scramble cryptography by extending reverse polish notation to use characters for both operations and operands. This algorithm reserves few characters as operations and the remaining as operands. RPN was used to test paralleled Genetic Programming (GP) operations. A reverse polish notation interpreter and randomized sub-selection form were used to test 22 peta GP operations/day on parallelized GPU using 14 workstations. The measured performance was marvelous; it is twenty times the best speed of the fastest previously published GP and more than sixty times that of the best-reported performance of the next fastest genetic programming on a single GPU system [23]. Zeng et al. [24] proposed auto programming for numerical data, "Remnant-standard-Deviation guided Gene Expression Programming (RD-GEP)." They presented and studied K-expression to reverse polish notation code generation without expression tree construction algorithm (K2RPN) and remnant-standard-deviation based fitness evaluation method in RD-GEP. Furthermore, RPN has applications in Datamine and data reverse engineering applications. Vanderbeek [25] suggested using RPN instead of infix notation in education. Vanderbeek recommended RPN as a teaching tool to help build a solid computational foundation. He suggests that RPN forces a shift in responsibility for operations order from the person performing the calculation to the person notating the calculation. Since each operation given in RPN creates implied parentheses around the two previous values, the skill of translating infix to RPN becomes essential. Thus, once an expression is correctly written in RPN, it can be quickly calculated with a low probability for error [26].

An extension of the learning classifier system, XCS is used to recombine and mutate classifiers [27], [28]. RPN expressions and stack-based Genetic Programming represent XCS classifier conditions. In contrast with other extensions of XCS involving tree-based Genetic Programming, the applied representation produced conditions that are linear programs, interpreted by a virtual stack machine (similar to a pushdown automaton), and recombined through standard genetic operators [27]. One solution to generate Catalan numbers is the reverse polish method, in which the sum would be presented as a string [29]. In fact, the reverse polish string can be formed for any product simply by deleting all left parentheses and substituting X for all right parentheses. Therefore, the string (a((bc)d)) has reverse Polish string abcXdXX [29].

The Bioinformatics field utilizes RPN to extract discriminant rules from oligopeptides for protease proteolytic cleavage activity prediction. The extract discriminant algorithm is developed using genetic programming. The algorithm has three essential components: min-max scoring function, RPN, and minimum description length [36] The min-max scoring function is developed using amino acid similarity matrices to measure the similarity between an oligopeptide and a rule, which is a complex algebraic equation of amino acids rather than a simple pattern sequence. The Fisher ratio is then calculated on the scoring values using the class label associated with the oligopeptides. The discriminant ability of each rule can therefore be evaluated. The use of RPN makes the evolutionary operations more straightforward and thus reduces the computational cost [30]. In [31], RPN was utilized to predict congestive heart failure timing errors asynchrony. In this study, the robustness is measured by an iterative program based on RPN to test the degree of asynchrony derived from an individual segment analysis.

In geometry, RPN is utilized to solve the convex polygon triangulation problem. Stanimirović et al. Presented an algorithm for convex polygon triangulation based on the reverse Polish notation [32]. They successfully showed how to apply the reverse Polish method in this area of symbolic computation.

Terris et al. used RPN to create a functional calculator, where python programming language and object-oriented paradigm were used [33]. Krtolica et al. also suggested a symbolic derivation method without a need for expression trees [34]. The expressions in RPN have unary pair functions and their compositions. These types of manipulation can be applied to symbolic computation areas, such as symbolic differentiation, polygon triangulation, elimination of redundant parenthesis [34].

Gruber et al. propose a new standard form for regular expressions. Reverse polish notation length was used with the alphabetic width of a regular expression as a measurement [35]. They showed that every regular alphabetic expression with n could be converted into a non-deterministic finite automaton with ϙ-transitions of size at most $42/5n + 1$, which results in an optimal bound.

3. Shunting Yard Algorithm

The Shunting Yard is a classic CS algorithm developed by Dr. E. W. Dijkstra in the early 1960s. It interprets a mathematical expression (usually in infix notation) and creates a reverse polish notation equivalent (postfix) while preserving operator precedence. The Shunting Yard algorithm, which is also known as Reverse Polish Notation (RPN), utilizes one stack and one output queue to transfer infix notation to postfix notation. Algorithm (1) shows the pseudocode for the Shunting Yard algorithm. If we consider the machine level when using push and pop operations, we can see that when we use push, we first need to load data from the stack and then compare it. In contrast, when we use pop, we compare first and then retrieve the data from the stack. This means that using stacks requires two steps, while using arrays allows for a more straightforward process, as arrays allow for direct comparison of their members at the machine level without the need for additional steps. Furthermore, the pop and push disturb the cpu pipeline. Thus, the cost of array access instructions is less than the cost of pop and push instruction.

4. The Proposed Algorithm: Transfer Yard

The proposed Transfer Yard algorithm utilizes an array to accomplish infix to postfix Transfer. Unlike the Shunting Yard algorithm, which is mainly designed to utilize the stack, the proposed algorithm utilizes only the stack for pre-brackets operation. Indeed, the proposed algorithm benefits from using an array through elements' direct access. For example, accessing an element in location n from the top of the stack requires n pop and push operations. While utilizing an array, accessing an element in position n in the array requires only one operation.

Moreover, the proposed algorithm uses operations position in the utilized array to indicate its presidency. Therefore, the proposed algorithm increases the performance by benefiting from the direct access of arrays. Algorithm 2 shows the pseudocode for the proposed algorithm where a Boolean array and in-place stack are utilized.

In-place stacks use infix expressions with tokens and minimum space for stack operations. This approach helps to reuse lost and unused memory for stack operations.

4. Implementation and Evaluation

To evaluate the proposed algorithm, we implemented it in two different environments utilizing c# and python. For a comparison purposes, we also implemented Shunting Yard as shown in Algorithm 1. We utilized Microsoft Visual Studio 2022 Professional. Microsoft Visual studio has been installed on personal computer with Intel Core™ i7-12700, 2.11GHz processors, and 16GB RAM. The input for the proposed Transfer Yard and Shunting Yard algorithm are infix expressions. The infix expressions are generated randomly with variable lengths. The first row of Table 1 shows the generated infix sizes (x); the size of expressions length ranges from 16 to 2024 character. The same expression is converted to postfix utilizing both algorithm and the running time was measured. To eliminate and artifact that affect the measured value, we conducted the following steps:

1- We stopped and disabled all unnecessary tasks and services
2- Disconnect the computer from the Internet and stopped all communication services.
3- To ensure minimum overheads, we developed a Console application for implement Shunting Yard and the proposed Transfer Yard algorithms.
4- The algorithms is executed 1500 times for each expression and we selected the minimum measured execution time to represent execution time for this expression.
5- Step 4 is repeated 500 times, and then the minimum execution time from all results is considered the execution time. Moreover, the standard deviation for all measured results were calculated.

The measured execution time for each expression size is shown in Table 1. The standard deviations for all experiments were less than 0.1. Figure 1 depicts the measured time for both algorithm with respect to expressing size.

To parametrize the comparisons between the two algorithm, we defined two performance measures Time% and performance. Which defined as

$$Performance = \frac{\text{shunting yard time}}{\text{Transfer Yard time}} \times 100\% \quad (1)$$

and

$$Time\ \% = \frac{\text{Transfer Yard time}}{\text{Shunting yard time}} \times 100\% \quad (2)$$

The two performance measures have been calculated for each expression size. Figure 2 shows the *Performance* and *Time%* and proves the superiority of the proposed algorithm.

Algorithm 1: The pseudo code for Shunting Yard algorithm.

```
Algorithm 1: Shunting Yard
-------------------------------------------------
Input:  IE: is an expression written in infix order
Output: PE: is an expression written in Postfix order
-------------------------------------------------
1 begin {main}
2     t: is a token in the IE expression
3     St: is temporary stack
4     while IE has more token then
5           t = read token from IE
6           if isOperand(t) then
7                 PE.enqueue(t)
8           else if t is left parenthesis then
9                 St.push(t)
10          else if isOperator(t) then
11                if precedence(t) > precedence (St.top()) then
12                      St.push(t)
13                Else
14                      repeat
15                            PE.enqueue(St.pop())
16                      until (St.top() is left parenthesis) Or
                              isEmpty(St)
17          else if t is right parenthesis then
18                repeat
19                      PE.enqueue(St.pop())
20                until (St.top() is left parenthesis)
21    end {while}
22    while (!isEmpty(st))
23          PE.enqueue(St.pop()
24    end {while}
25 end {main}
```

Algorithm 2: The pseudo code for proposed Transfer Yard algorithm.

```
Algorithm 2: Transfer Yard Algorithm (TYA)
-------------------------------------------------
Input:     IE: is an expression written in infix order
           x : the index to start parsing from
Output:    PE: is an expression written in Postfix order
-------------------------------------------------
1 begin {main}
2     tokens     : is the list of tokens in the IE start from x
                   to the end of expression
3     list_ops   : An array of operations ordered by precedency
4     op         : The index of current operation
5     prop       : The highest precedency appeard till now
6     output     : the subexpression in Postix order
7     prop =0
      For (i = x to tokens.length)
8           t = tokens(i)
            op = 0
9           if t is open paranthesis then
10                i++
11                output.append(TYA(i, IE) )
12          else if t is close parenthesis then
13                for (k = 4 to 2 step -1)
14                      if list_ops[prop] != ' ' then
15                            PE.append(list_ops[prop])
16                            list_ops[prop] = ' '
17                x = i
18                return output
19          else if isdigit(t) then
20                output.append(t)
21          else //t is an operation
22                if t == '/' or t == '*' then op = 3
23                else if t == '+' or t == '-' then op = 2
24                else if t = '^' then op = 4
25                if prop = 0 then list_ops[op] = t
26                else if list_ops[op] <= list_ops[prop] then
27                      while (op <= prop)
28                            if list_ops[prop] != ' ' then
29                                  output.append(list_ops[prop])
30                                  list_ops[prop] = ' '
31                            prop--
32                list_ops[prop] = t;
33                prop = op
34    end {For}
35    for (k = 4 to 2 step -1)
36          if list_ops[prop] != ' ' then
37                output.append(list_ops[prop])
38          prop--
39    End {for}
40    PE = output
41    return PE
42 end {main}
```

Table 2: Experimental results for Shunting Yard and the proposed Transfer Yard algorithm.

Expr size (x)	16	32	64	128	256	512	1024	2048
Expr size (2^x)	4	5	6	7	8	9	10	11
Shunting yard Alg.	5.55	10.16	20.27	40.08	80.19	161.19	321.28	695.66
Transfer Yard Alg.	4.64	6.59	14.48	27.67	54.31	107.55	213.29	460.32
Time %	83.51%	74.66%	71.45%	69.05%	67.33%	66.72%	66.39%	66.17%
Performance	119.7%	133.9%	140.0%	144.8%	148.5%	149.9%	150.6%	151.1%

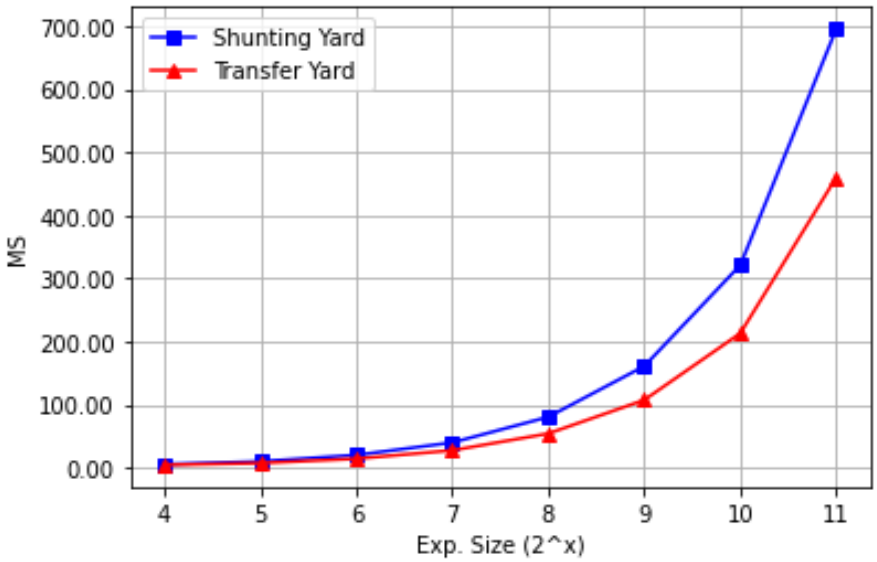

Figure 1: Measured time (MS) for Shunting Yard and Transfer Yard with respect to expression size.

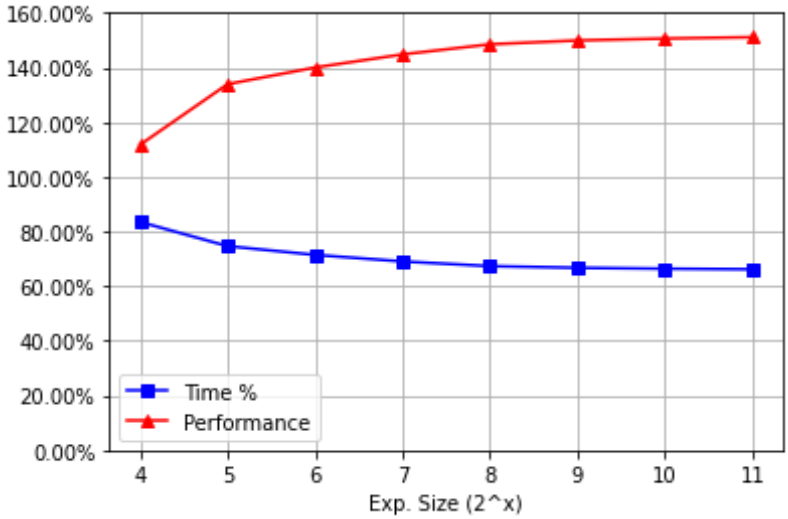

Figure 2: Time % and Performance for the propose Transfer Yard performance with respect to shunting Yard algorithm.

5. Conclusion

This paper has proposed a novel algorithm that is an alternative to Shunting yard algorithm. The proposed algorithm reduces the utilization. It actually utilizes an array structure in storing the operations and compare operator precedency. The proposed use random access memory instead of stack memory. Actually, the proposed algorithm has boosted performance by replacing heavy stack and most stack operations with Boolean array. The Boolean array is faster to test if an operator is already passed or not, and to drift an operator to its presidency index directly without the overhead of pushing and popping operations. The proposed algorithm in implemented utilizing two programming languages to measure its performance. In addition, the known shunting yard algorithm was implemented in the same manner. The results of the proposed algorithm outperform the result of shunting yard algorithm. Finally, we emphasize that a paralyzed and a hardware implementation of the proposed algorithm will be more efficient and boost results.

References:

[1] Wikipedia - The free encyclopedia, "Reverse Polish notation." 2021, [Online]. Available: http://en.wikipedia.org/wiki/Reverse_Polish_notation.

[2] J. Dixon, *A Brief History of the Computer Sequencer and DAW*, Quick Star. 2015.

[3] J. Levine, *Flex & Bison: Text Processing Tools*. "O'Reilly Media, Inc.," 2009.

[4] S. Lempp, "Kunen Kenneth, The Foundations of Mathematics, Studies in Logic, Mathematical Logic and Foundations, vol. 19. College Publications, London, 2009, vii+ 251 pp.," *Bull. Symb. Log.*, vol. 22, no. 2, pp. 287–288, 2016.

[5] A. S. Tanenbaum, *Structured computer organization*. Pearson, 2006.

[6] T. K. Enterprises, *Everything you've always wanted to know about RPN but were afraid to pursue, comprehensive manual for scientific calculators*. T.K. Enterprises, 1976.

[7] L. Null and J. Lobur, *The essentials of computer organization and architecture*. Jones & Bartlett Publishers, 2014.

[8] E. W. Dijkstra, "An ALGOL 60 translator for the x1," *Annu. Rev. Autom. Program.*, vol. 3, pp. 329–345, 1963.

[9] R. Rastogi, P. Mondal, and K. Agarwal, "An exhaustive review for infix to postfix conversion with applications and benefits," in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, 2015, pp. 95–100.

[10] P. V Krtolica and P. S. Stanimirović, "On some properties of reverse Polish notation," *Filomat*, pp. 157–172, 1999.

[11] A. Tiwari, A. Gupta, and V. K. Singh, "Developing a Multi-threaded Algebraic Application using Mathematical Pseudo Language for Efficient Computing," 2013.

[12] M. Semenov, Y. S. Colen, J. Colen, and A. Pardala, "An Introduction to the Edumatrix Set and Its Didactic Capabilities," *Res. Math. Educ.*, vol. 23, no. 1, pp. 47–62, 2020.

[13] D. AROTARITEI, M. TURNEA, C. IONITE, and M. ROTARIU, "Artificial intelligence applied to model the sulphur absorption process-a possible application in cure with sulphurous mineral water," 2020.

[14] S. Gocht and J. Nordström, "Certifying Parity Reasoning Efficiently Using Pseudo-Boolean Proofs," 2021.

[15] G. I. Shivacheva and K. B. Yankov, "Graphical simulation of functions," in *IOP Conference Series: Materials Science and Engineering*, 2021, vol. 1031, no. 1, p. 12052.

[16] B. Zhang *et al.*, "Power User Taging System Based on Micro-service," in *IOP Conference*

Series: Earth and Environmental Science, 2021, vol. 632, no. 4, p. 42088.
[17] M. Chodacki, "Feedback Shift Registers Evolutionary Design Using Reverse Polish Notation," in *Asian Conference on Intelligent Information and Database Systems*, 2019, pp. 475–485.
[18] D. M. Kasprzyk, C. G. Drury, and W. F. Bialas, "Human behaviour and performance in calculator use with Algebraic and Reverse Polish Notation," *Ergonomics*, vol. 22, no. 9, pp. 1011–1019, 1979, doi: 10.1080/00140137908924675.
[19] V. AVRAM, "A Formalization Way For Computational Formulas Within Business Rules Defined In Sik Repositories," 2014.
[20] M. Makuuchi, J. Bahlmann, and A. D. Friederici, "An approach to separating the levels of hierarchical structure building in language and mathematics," *Phil. Trans. R. Soc. B*, vol. 367, no. 1598, pp. 2033–2045, 2012.
[21] M. E. Palmer, "Evolved neurogenesis and synaptogenesis for robotic control: the L-brain model," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 1515–1522.
[22] S. S. Dhenakaran, "Employing Reverse Polish Notation in Encryption," *Int. J. Comput. Sci. Inf. Secur. No. 0975-3826)*, 2011.
[23] W. B. Langdon, "A Many Threaded CUDA Interpreter for Genetic Programming.," in *EuroGP*, 2010, pp. 146–158.
[24] T. Zeng, Y. Liu, X. Ma, X. Bao, J. Qiu, and L. Zhan, "Auto-programming for Numerical Data based on Remnant-standard-deviation-guided Gene Expression Programming," in *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, 2009, vol. 3, pp. 124–128.
[25] G. Vanderbeek, "Order of Operations and RPN," *MAT Exam Expo. Pap.*, p. 46, 2007.
[26] S. Okamura, I. Matushima, and Y. Yano, "The effective learning support strategy for self learning with problem-based learning," in *Creating, Connecting and Collaborating through Computing, 2005. C5 2005. Third International Conference on*, 2005, pp. 150–157.
[27] P. L. Lanzi, "XCS with stack-based genetic programming," in *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, 2003, vol. 2, pp. 1186–1191.
[28] P. L. Lanzi and M. Colombetti, "An extension to the XCS classifier system for stochastic environments," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, 1999, pp. 353–360.
[29] F. JARVIS, "Catalan Numbers," *Math. Spectrum", number={36}, Vol. pages={9-11}, year={2003}*.
[30] S. Porwal, Y. Chaudhary, J. Joshi, and M. Jain, "Data compression methodologies for lossless data and comparison between algorithms," *Int. J. Eng. Sci. Innov. Technol. Vol.*, vol. 2, pp. 142–147, 2013.
[31] C. Van Eyll, M. F. Rousseau, J. Etienne, L. Stoleru, A. Charlier, and H. Pouleur, "High performance regional wall synchrony analysis in severe systolic dysfunction: a new program based on reverse Polish notation," in *Computers in Cardiology 1995*, 1995, pp. 509–512.
[32] P. S. Stanimirović, P. V Krtolica, and R. Stanojević, "A non-recursive algorithm for polygon triangulation," *Yugosl. J. Oper. Res.*, vol. 13, no. 1, pp. 61–67, 2003.
[33] B. Terris, P. Toussaint, S. Varga, and D. MacQuigg, "A calculator program using Object Oriented Data Structures," *Instructor*, 2007.
[34] P. V Krtolica and P. S. Stanimirović, "Reverse polish notation method," *Int. J. Comput.*

Math., vol. 81, no. 3, pp. 273–284, 2004.
[35] H. Gruber and S. Gulan, *Simplifying regular expressions: A quantitative perspective*. Universitätsbibliothek, 2012.
[36] Yang, Zheng Rong, et al. "Searching for discrimination rules in protease proteolytic cleavage activity using genetic programming with a min-max scoring function." Biosystems 72.1-2 (2003): 159-176.
[37] Peretiaha, M., Poltavets, M., Smelyakov, K., & Chupryna, A. (2023). SYNTACTIC ANALYSIS OF ARITHMETIC EXPRESSIONS FOR OPTIMIZING THE OPERATION OF PROGRAMS. Grail of Science, (26), 215-229.
[38] Bochenek, B., & Tajs-Zielińska, K. (2023). TABASCO—Topology Algorithm that Benefits from Adaptation of Sorted Compliances Optimization. Applied Sciences, 13(19), 10595.
[39] Albazar, H. (2020). A new automated forms generation algorithm for online assessment. Journal of Information & Knowledge Management, 19(01), 2040008.
[40] Al Bazar, H., & Abdel-Jaber, H. (2020). A Developed Uncapacitated Scheduling Algorithm of Building Timetables for Different Exam Kinds.
[41] Wei, G. (2023). Metaprogramming Program Analyzers (Doctoral dissertation, Purdue University).