# Pseudocode Interpreter (Pseudocode Integrated Development Environment with Lexical Analyzer and Syntax Analyzer using Recursive Descent Parsing Algorithm)

**Asia Pacific Journal of Multidisciplinary Research**
Vol. 5 No.4, 31-38
November 2017
P-ISSN 2350-7756
E-ISSN 2350-8442
www.apjmr.com

**Christian Lester D. Gimeno**
Computer Department, College of Arts and Sciences, Iloilo Science and Technology University, Iloilo City, Philippines
*christianlester.gimeno@isatu.edu.ph*

*Date Received: August 2, 2017; Date Revised: October 27, 2017*

***Abstract*** *–This research study focused on the development of a software that helps students design, write, validate and run their pseudocode in a semi Integrated Development Environment (IDE) instead of manually writing it on a piece of paper.Specifically, the study aimed to develop lexical analyzer or lexer, syntax analyzer or parser using recursive descent parsing algorithm and an interpreter. The lexical analyzer reads pseudocodesource in a sequence of symbols or characters as lexemes.The lexemes are then analyzed by the lexer that matches a pattern for valid tokens and passes to the syntax analyzer or parser. The syntax analyzer or parser takes those valid tokens and builds meaningful commands using recursive descent parsing algorithm in a form of an abstract syntax tree. The generation of an abstract syntax tree is based on the specified grammar rule created by the researcher expressed in Extended Backus-Naur Form. The Interpreter takes the generated abstract syntax tree and starts the evaluation or interpretation to produce pseudocode output.*

*The software was evaluated using white-box testing by several ICT professionals and black-box testing by several computer science students based on the International Organization for Standardization (ISO) 9126 software quality standards. The overall results of the evaluation both for white-box and black-box were described as "Excellent in terms of functionality, reliability, usability, efficiency, maintainability and portability".*

***Keywords*** *–Interpreted Programming Language, Lexical Analysis, Pseudocode Interpreter, Recursive Descent Parsing,Syntax Analysis*

## INTRODUCTION

Pseudocode is an outline that simplifies and represents programming language, used in designing programs [1].Pseudocode is very important in the field of Computer Science because of its simplicity in representing algorithms and is considered as one of the best notation used in teaching introductory computer programming. An algorithm by definition is an ordered set of well detailed and clear steps provided to a computer in solving a particular computing problem.

Teaching Computer Science students how to design, analyze and write pseudocode in solving computer programming problems is very difficultand challenging to the Computer Science educators. At present, there is no standard pseudocode syntax adopted in teaching computer program logic formulation. Syntax refers to the correct structure and grammar of the pseudocode. Book authors and educators implement their own syntax, special techniques, and sets of grammar rules in writing pseudocode.

Educators spend a lot of time checking and evaluating pseudocodes manually written by students on a piece of paper. As a result, educators cannot give immediate feedback and corrections to each student.
In the same way, students who are taking Programming Fundamentals subject cannot comprehend the concept of creating an algorithm well since it is written on a piece of paper. As a result,students are not given the chance to actually see the output of their written pseudocode and they cannot test their pseudocode designs whether it is correct or wrong as the designs are not executable. Executable

means that it can be run on the computer to see the output. Furthermore, students are not sure whether their pseudocode meets an educator's expected syntax and logical designs.

To solve these problems, the researcher developed a software that served as a tool in teaching Programming Fundamentals. The tool served as an Integrated Development Environment (IDE) where students can design, write, validate and run their pseudocode to see its output. Moreover, educators can save time checking the correctness of the pseudocode written by each student and can provide immediate feedback to the students.

The researcher developed a lexical analysis or lexer that splits pseudocode file or source into tokens; a syntax analysis or parser using recursive descent parsing algorithm that validates pseudocode syntactically and generates an abstract syntax tree along with the commands. The generation of an abstract syntax tree is based on the specified grammar rule created by the researcher expressed in Extended Backus-Naur Form.In computer science definition, Extended Backus-Naur Form is a notation used to expressa formal description of a programming language. In addition, the researcher developed an interpreter that interprets the generated abstract syntax tree and block of commandsto produce the output.In this study, abstract syntax tree refers to an abstract data plan or data structure used as global storage of all valid tokens and commands generated by the parser or syntax analyzer. The keywords used in this study are common words from the English language and can be easily understood by students.

The result of this research will help students to quickly write, debug and run pseudocode with ease instead of writing it on a piece of paper. The students can grasp and understand the concept of programming well when they see the output of their written pseudocode quickly using the software. The software will provide accompanying keyword highlighter to help students familiarize with all the keywords used in pseudocode.

The research is also beneficial to the Computer Science educators who are teaching introductory computer programming subjects, for it can be used as an instructional tool in teaching pseudocode and problem-solving. This research will help educators explain the concept of algorithm expressed in pseudocode with minimal effort by using the proposed software. They can allow students achieve a deeper understanding of pseudocoding techniques using the software rather than by simply reading a textbook and writing pseudocode on a piece of paper.

## RELATED LITERATURE

The syntheses of ideas and information that support the development of Pseudocode Interpreter conducted through an active and profound research are presented. An overview of the Lexical Analysis (Lexer or Tokenizer), Syntax Analysis (Parsing or Parser), Recursive Descent Parsing, Abstract Syntax Tree, and Interpreter are discussed. Moreover, the review of previous researches, applications, requirements and other considerations that were found in the journals, books, websites, encyclopedias, thesis/dissertations and other published studies were included to further support the study.

A lexical analyzer, or lexer for short, takes a string of individual letters and divide this string into tokens. Additionally, it will filter out whatever separates the tokens (the so-called white-space), i.e., layout characters (spaces, newlines etc.) and comments [2].
In this study, the researcher used Lexical Analysis to scan the pseudocode source as a stream of characters and converts it into meaningful lexemes and stores it on a data structure. Lexical analyzer represents these lexemes in the form of the token as: <token-name, attribute-value>. The lexical analyzer splits the pseudocode source into lexemes, by skipping any whitespaces, single line comments and multiple line comments.

Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions [3].

In this study, a token of a language is a category of its lexemes. For example, an identifier is a token that can have lexemes, or instances, such as sum and total. Consider the following pseudocode statement:
*magic = 2 * counter + 10;*

The lexemes and tokens of this statement are:

| **Lexemes** | **Tokens** |
|---|---|
| *magic* | *identifier* |
| *=* | *equal-sign* |
| *2* | *integer-literal* |
| * | *multiplication-operator* |

| | |
|---|---|
| *counteridentifier* | |
| *+* | *addition-operator* |
| *10* | *integer-literal* |
| *;* | *semi-colon* |

Syntax analyzer is the act of checking whether a grammar "accepts" an input text as valid (according to the grammar rules).As a side effect of the parsing process, the entire syntactic structure of the input text is uncovered. Since the grammar rules are hierarchical, the result is a tree-oriented data structure, called parse tree or derivation tree [4].

In this study, the syntax analyzer or parser is the part of the program that takes tokens from a data structure passed by the lexer, checks them, and takes some action according to the token's value. The parser takes those actions based on the rules of the programming language expressed in Extended Backus-Naur Form.

Another literature is the Abstract Syntax Tree.

In computer science, an abstract syntax tree (AST), or just a parse tree or syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is "abstract" in not representing every detail appearing in the real syntax. For instance, grouping parentheses are implicit in the tree structure, and a syntactic construct like an if-condition-then expression may be denoted by means of a single node with three branches. This distinguishes abstract syntax trees from concrete syntax trees, traditionally designated parse trees, which are often built by a parser during the source code translation and compiling process. Once built, additional information is added to the AST by means of subsequent processing, e.g., contextual analysis [5].

In this study, the Abstract Syntax Tree is a data structure that acts as a global storage of programming language commands. The researcher used linked list to implement this abstract syntax tree to organize commands (statements) in a data structure.

The researcher also used the Extended Backus-Naur Form (EBNF) in describing pseudocode grammars and structure theorem in implementing the interpreter. The Interpreter, on the other hand, takes the instructions handled by the parser and does the items in the source code in a certain order. Moreover, the interpreter reads the source code, gets tokens and parses it to have an abstract plan in a data structure. When the execution time comes, the interpreter reads the abstract instructions from the data structure and executes things one by one.

Structure Theorem states that it is possible to write any computer program by using only three basic control structures that are easily represented in pseudo code: sequence, selection, and repetition.

*Sequence*.The sequence control structure is the straightforward execution of one processing step after another.

*Selection*.The selection control structures are the presentation of a condition and the choice between two actions, the choice depending on whether the condition is true or false. This construct represents the decision-making abilities of the computer and is used to illustrate the fifth basic computer operation, namely to compare two variables and select one of two alternative actions.

*Repetition.*The repetition control structure can be defined as the set of instructions to be performed repeatedly, as long as a condition is true. The basic idea of repetitive code is that a block of statements is executed again and again, until a terminating condition occurs. This construct represents the sixth basic computer operation to repeat a group of actions[6].

The researcher used Structure Theorem as the proper structure on how the user will input the pseudocode. The user must follow the Structure Theorem to prevent error/s in validating the inputtedpseudocode. However, there will be a difference with the syntax in the structure theorem. The researcher created and adopted the closely related language like C language to the syntax found on the Structure Theorem. The uses of curly braces, brackets, semicolon at the end of every statement are examples to be adopted in this study.

PseudoCode Compiler is a software that implements a basic version of the pseudocode used for teaching algorithms and problem-solving concepts. The software was developed by Chris Henderson [7].

The Pseudocode Compiler is similar to this study because this software used Lexer to tokenize written pseudocode and a Parser that parse tokens to produce the output. The software also supports looping constructs such as while, repeat until and counted loop. However, the software does not support nested construct such as nested loop and nested condition. It does not also support combined condition using logical AND and OR. In this study, the researcher's

software is similar to the Pseucode Compiler but it can support nested loops, nested condition, combined condition and modulo operator that enables students to solve complex programming problems.

Another software called Dynamic Parser can perform syntactic analysis or parsing of input data consisting of a set of tokens based upon a provided grammar including conditional tokens. It was developed by Evgueni *Zabokritski*. While the parser grammar can be fixed, the dynamic parser can utilize an independent transform function at parse time to translate or replace particular tokens effectively performing dynamic parsing. The transform function can be utilized in conjunction with conditional tokens to selectively activate and deactivate particular grammar rules [8].

A dynamic Parser is a software that can parse the input of the user based upon a provided grammar including conditional tokens similar the Pseudocode Interpreter that uses a parser that validates the pseudo code source inputted by the user according to the specified grammar expressed in Extended Backus-Naur Form.PSeInt is a pseudocode interpreter for Spanish-speaking programming students developed by Pablo Novara. Its main purpose is to be a tool for learning and understanding the basic concepts about programming and applying them with an easy understanding Spanish pseudocode[9].PSeInt is a tool to assist students in their first step in programming. Through a simple and intuitive pseudo-language in Spanish (supplemented with a text editor flowcharts), it allows you to focus on the fundamental concepts of computational algorithms, minimizing the difficulties of language and providing a work environment with numerous grants and didactic resources. PSeInt is related to this study because it uses Lexical and Syntax Analysis in processing pseudocode source in the Spanish language. The tool also supports keywords and syntax coloring similar to the tool to be developed in this study which will support variety set of keywords and functions library.

## OBJECTIVES OF THE STUDY

The general objective of this study was to develop a Pseudocode Interpreter.

Specifically, this research study aimed to develop lexical analyzer that performs lexical analysis on pseudocode to produce a valid token; develop syntax analyzer using recursive descent parsing algorithm that validates pseudocode source syntactically and generates an abstract syntax tree and commands; develop an Interpreter that interprets the generated abstract syntax tree and commands to produce the necessary interpretation of pseudocode source; and evaluate the system based on International Organization for Standardization (ISO) 9126 in terms of functionality, reliability, usability, efficiency, maintainability and portability.

## MATERIALS AND METHODS

Pseudocode Interpreter is a software that acts as an Integrated Development Environment for the students who are taking programming introductory subject such as program logic formulation. Its main purpose is to be a tool for learning and understanding the basic concepts of programming and applying an easy understanding of the programming language structure. The software was designed to help students learn writing correct pseudocodes and to solve programming problems. This software was also intended to help educators teach the basic of programming notion using pseudocode. The researcher designed a software to establish standard pseudocode for teaching introductory programming subject (CS 2- Program Logic Formulation) in Computer Science, Information Systems, and Information Technology curriculums.

The skeletal structure of the software was depicted and shown in Figure 1.

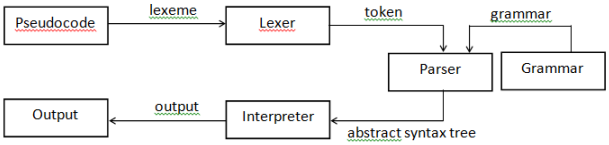


*Figure 1. Skeletal Structure of the Software*

This is how it works semantically:

• LEXER gets the *pseudocode source* and splits it into *tokens*

• PARSER gets those *tokens* and creates commands from them according to thelanguage grammar. It places those commands into the *abstract syntax tree.*

• INTERPRETER takes the *abstract syntax tree* and starts interpretation to produce output.

To fully understand the flow of the software, the context diagram had been discussed in this section.

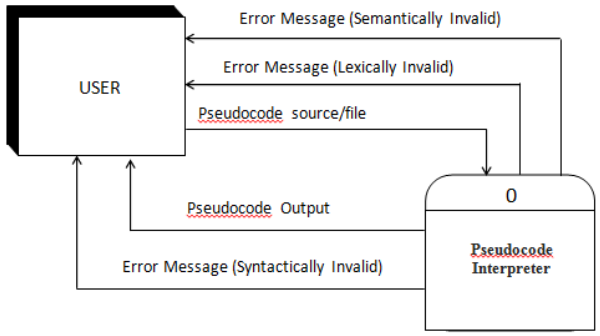


*Figure 2. Context Diagram of Pseudocode Interpreter*

The Context Diagram of the Pseudocode Interpreter is shown in Figure 2 depicting the actual flow of data coming from the user and its transformation. It contains the process symbol that represents the software to model. It also shows the external entity (user) who will interact with the software. In this diagram, the users who may interact include the students, teachers or anyone who wants to use the software and learn programming essentials. In between the process and the external entity, there are data flows (connectors) that indicate the existence of information exchange between the entity and the software.The software accepts a pseudocode source or a pseudocode file given by the user. Then, it performs lexical and syntax analysis to determine the validity of the tokens. The software will notify the user if there are some invalid tokens received, otherwise, it will generate an abstract syntax tree to be used by the interpreter in generating the output.

*Project Development Methodology*

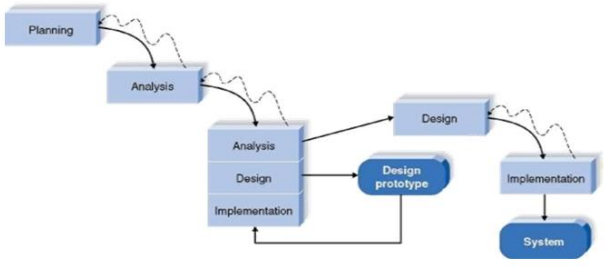


*Figure 3. Prototyping-based Methodology [11]*

Software prototyping methodology is the software engineering model used by the researcher in the development of Pseudocode Interpreter as shown in Figure 3.

The software prototyping refers to building software prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software [10].
In software prototyping, instead of freezing the requirements before proceeding to design or coding stage, a prototype is built to understand the requirements. The prototype is developed based on the currently known requirements. By using the prototype, the end user can get an actual feel of the software since the interactions with the prototype enable them to understand the requirements of the desired software better.

The software was implemented using Java programming language and Netbeans 8.2 on Windows environment. Java Developers Kit version 8 and Java Runtime Environment were also used.

## RESEARCH METHOD USED

The evaluation of the software was based on the evaluation criteria of the standards of ISO 9126. The ISO 9126 software quality model identifies six main quality characteristics namely: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability.

For the test case design method, the researcher used both the white box and black box testing methods. For the white box, the researcher took the basis path testing wherein it examines all the possible paths of execution for at least once, which includes the flow graph notation, cyclomatic complexity, independent path and the graph matrices. These methods focus on control structure and the internal/logical structure of the software. The respondents for the white box testing are mostly developers, IT personnel, programmers, and educators. The Black box testing was conducted by the researcher and was done based on the Graphical User Interface(GUI) of the system, where the researcher examined some fundamental aspects of the software with little regard for the internal logical structure of the software. The respondents of the black box testing are the students who have already taken up introductory programming subject. For the testing strategies, the researcher applied unit testing wherein each method or operation within the class of the software will be tested to uncover errors in the internal processing logic and data structure within the boundaries of the module. Next, the researcher applied an integration testing wherein the approach

used is the bottom-up approach. The researcher began the constructions and testing from the lowest level and moving upward. In addition, the researcher applied the regression testing ensuring that the integrated software does not produce unintended side effects while integrating each cluster. Then the researcher applied the validation testing which is based on the requirement specifications of the end-users wherein in all functional requirements were satisfied, and all behavioral characteristics were achieved. Under validation testing, the researcher conducted the Alpha and Beta testing wherein the Alpha testing was done on the developers side with the representative group of the end-user while the beta testing was done in an environment that cannot be controlled by the researcher; the end-users record all problems that had been encountered and these were modified. Lastly, the researcher used software testing to fully exercise the Pseudocode Interpreter through stress testing, and deployment testing.

**VALIDATION OF INSTRUMENT**

The instrument used for the study contains the Personal Information of the respondents, as well as their educational and employment information.
The list of ISO 9126 statements used in the instrument has been validated and published by Abra, Al-Qutaish, Desharnais and Habra [12].

**DATA COLLECTION PROCEDURE**

To gather initial data such as software requirements and in-depth knowledge of the study,the researcher conducted series of interviewswith different computer science students and with the different ICT professionals. The purpose of the interview was to get all the software requirements to start the software development. After the development of the software, the researcher conducted quantitative data gathering in the form of a survey using questionnaires. Two sets of survey questionnaires were prepared for two sets of respondents.

The researcher has chosen30 respondents including two teachers specializing in programming subjects, ten software developers from the industry and eighteen computer science students. The respondents were chosen through purposive sampling technique in which the researcher has chosen a specific group or a person willing to participate in this research. Implementing purposive sampling technique warrants that a cross-section of computer science students is included in the sample. This method allows each computer science student,already taken (CS 2) Program Logic Formulation subject (who have passed or failed the subject), male or female, 2nd year to 3rd year were invited to evaluate the software. Furthermore, ten software developers from the industry, who are 18 years old and above, male or female, specializing in programming (any programming language), junior or senior programmer were also invited to evaluate the software. Each respondent had the privilege to state their suggestions, comments, and feedback.Two important ethical issues adhered during the conductof the research, confidentiality and informed consent. The respondent's right to confidentiality is always respected in this research and any legal requirements on data protectionadhered. The respondents were fully informed about the aims of the research, and the respondent's consent to participate in the evaluation were obtained and recorded.

**DATA PROCESSING AND STATISTICAL TREATMENT**

The researcher used Microsoft Excel 2010 for the data processing and analysis in this study. To determine the validity and reliability of the software, the weighted Mean and Standard Deviation (SD) was used to ensure that the software conforms to its stated requirements.

Displayed in Table 1is the scoring method used by the respondents to evaluate the software based on ISO 9126.

**Table 1.Scoring Method**

| Range of Scale | Description |
|---|---|
| 4.21-5.00 | Excellent |
| 3.41-4.20 | Very Satisfactory |
| 2.61-3.40 | Satisfactory |
| 1.81-2.60 | Good |
| 1.00-1.80 | Poor |

**RESULTS AND DISCUSSION**

The researcher used the weighted mean and standard deviation in determining the functionality, and the user acceptability of the software. Based on the results of the evaluation collected from the ICT professionals, the overall results of the software's evaluation based on ISO 9126 software quality standards was rated with a mean of 4.31 and standard deviation of 0.63 and is described as "Excellent" as shown in Table 2.This means that software is

acceptable with no revisions as evaluated by the different ICT professionals including programmers, developers, and ICT teachers.

**Table 2.Evaluation of the System through White Box Testing of the Software (perceived by ICT professionals)**

| Statement | Mean | Description | SD |
|---|---|---|---|
| Objective 1: develop lexical analyzer that performs lexical analysis on pseudocode to produce a valid token. | 4.00 | Very Satisfactory | 0.85 |
| Objective 2: develop syntax analyzerusing recursive descent parsing algorithm that validates pseudocode source syntactically and generates an abstract syntax tree and commands. | 4.17 | Very Satisfactory | 0.58 |
| Objective 3: develop an Interpreter that interprets the generated abstract syntax tree and commands to produce the necessary interpretation of pseudocode source. | 4.75 | Excellent | 0.45 |
| **Overall** | **4.31** | **Excellent** | **0.63** |

For objective 1stated as develop lexical analyzer that performs lexical analysis on pseudocode to produce a valid token has a Mean value of 4.00 and SD of 0.85 which described as "Very Satisfactory". This means that the lexical analyzer functions accurately based on its stated purpose and set of functionalities to validate tokens on pseudocode source. The results denote that the software as perceived by the respondents, handles tokens and lexemes coming from lexer while maintaining its performance level. The software can accommodate multiple entries of pseudocode source. The software also exhibits lack of software errors and failure.

For objective 2 stated as develop syntax analyzerusing recursive descent parsing algorithm that validates pseudocode source syntactically and generates an abstract syntax tree and commands has a Mean value of 4.17 and SD of 0.58 which described as "Very Satisfactory". This means that the main function of the softwareto analyze pseudocode source syntactically shows that the software components interacted with other components of the software accurately and completely displayed the necessary results such as parsing arithmetic and boolean expressions; ladderized and nested conditional statements; switch case andarrays; and nested looping statements such as for loop, while loop and do-while loop.

For objective 3 stated as develop an Interpreter that interprets the generated abstract syntax tree and commands to produce the necessary interpretation of pseudocode source has a Mean value of 4.75 and SD of 0.45 which described as "Excellent". This means that the software accurately and completely displayed the necessary results. It is also implied that the software, as evaluated by the respondents' complied as an application which validates and interprets pseudocode correctly based on the syntax and makes it executable whenever the user writes and runs pseudocode source on the software.The majority of the respondents also agreed that the software functions accurately based on its stated functionality.

Table 3. Evaluation of the System through Black Box Testingof the Software (perceived by computer science students)

| Characteristics | Average Mean | Description | Standard Deviation |
|---|---|---|---|
| Functionality | 4.18 | Very Satisfactory | 0.56 |
| Usability | 4.22 | Excellent | 0.53 |
| **Average Mean** | **4.20** | **Very Satisfactory** | **0.6** |

The overall average mean of the Pseudocode Interpreter was rated as 4.20 with a standard deviation of 0.60 described as "Very Satisfactory" based on the perception of the computer science students. It signifies that the software is easy to use and the features are easily understood by the students.As shown in Table 3,the assessment of the users in terms of functionality was described as "Very Satisfactory" with a mean value of 4.18 and SD of 0.56 which described as "Very Satisfactory". This means that the software functions accurately based on its stated functionality. It also implies that the software interoperates cohesively or smoothly with other related libraries which include the abstract syntax tree classes, the runtime classes and the grammar rules.

Usability was described as "Excellent" by the respondents with a mean of 4.22 and SD of 0.53.This means that the functions and content of the software were designed for its intended user.The software is easy to use and can be operated by the user in a given environment and the software provides easy

navigations.It shows that it was readily accepted by the majority of the students because it shortens the learning curve of the students. Moreover, the results of the evaluation show that the software was designed according to the needs and specifications of the end-users.

## CONCLUSION AND RECOMMENDATION

After attaining all the objectives of the research entitled Pseudocode Interpreter, the researcher established the following conclusions: (1) the software can accept the user's input based on the Robertson's structure theorem; (2)the software performs lexical analysis on the pseudocode source, matching every element found and creates a valid token; (3)the software can interpret arithmetic expressions, boolean expressions, and combined arithmetic-boolean expressions; (4) the software can parse three control structures such as sequence, selection, and repetition; and (5) the software can interpret the pseudo code source and provides the user the interpreted results.

The following are the recommendations for the other functions and improvement of the software. These are the following: (1) the software does not have code completion to speed up the process of coding, the researcher recommends toenhance the software that has the intelligence or can give auto-suggestion; (2) that software does not have number lines for better readability, hence, the researcher recommends to enhance the software that can display number lines to help user finds specific line of code;and (3) the software does not have desk check table for variable monitoring purposes, thus, the researcher recommends to enhance the software that can produce desk-check table for step by step simulation of the output and better monitoring of the values displayed on the screen.

## REFERENCES

[1] Terms, P. I. (2017). SOFTWARE PROGRAMMING. Retrieved June 27, 2016, from https://prezi.com/umsi0yh9oey8/software-programming

[2] Compiler design Tutorial. (2017). Retrieved June 7, 2016, from https://www.tutorialspoint.com/compiler_design/

[3] Mogensen, T. ©. (2009). Basics of compiler design. [S.l: TorbenÆgidiusMogensen.

[4] Aho, A. V., Lam, M. S., &Sethi, R. (2006). Compilers: Principles, techniques, and tools (2nd ed.). Boston, MA: Pearson Addison-Wesley.

[5] Tutorial on abstract syntax trees (ASTs). Retrieved July 2, 2016, from http://www.jmodelica.org/api-docs/usersguide/1.2.0/ch04s01.html

[6] Robertson, L. A. (2006). Simple program design, A step-by-step approach, fifth edition (5th ed.). United States: Delmar Cengage Learning.

[7] Henderson, C. Retrieved June 27, 2016, from http://www.hendersontech.com/?cat=5

[8] Zabokritski, E. (2008). Dynamic Parser. United States Patent

[9] PSeInt. Retrieved June 3, 2016, from http://pseint.sourceforge.net

[10] SDLC - Software Prototype Model. Retrieved from Tutorialspoint, http://tutorialspoint.com

[11] Tegarden, D. P., Dennis, A., & Wixom, B. H. (2012). Systems analysis and design with UML (4th ed.). United States: John Wiley & Sons.

[12] A. Abran, R. A. Qutaish, J. Desharnais, and N. Habra (2008). Software QualityMeasurement: Concepts and Approaches. Institute of Chartered Financial Analysts of India.

## COPYRIGHTS