# A Novel Approach towards Very High Level Programming

Darshan Patel, Devashish Thakur and Kavi Mahesh

*Abstract*—A significant share of programming errors occurs in looping structures, especially in code written by beginners. In this paper, we present an approach to very high level programming to reduce the complexity of the syntax and semantics of looping constructs. Our solution provides the user with an interactive editor where he can program in high-level algorithmic language independent of the syntax of any particular programming language. Any errors in the algorithm are removed by interacting with the user through a series of dialog boxes. Errors are rectified automatically by the editor without the need of the user to find the particular line of code. The editor also provides the user with an option to view equivalent Java code of the algorithm being programmed. The algorithm can also be converted to a Java program free from any looping errors.

*Keywords*—very-high level programming, loop structures, interactive programming, editor.

## I. INTRODUCTION

A major difficulty in today's programming languages is the difficulty in learning and the need for remembering the syntax of that particular language. Programmers often get confused with the looping syntax and end up using wrong looping structure that leads to erroneous output that are very difficult to debug. In our proposed editor the user need not remember the syntax of any of the loops. The user just needs to select the group of lines that have to be looped. The enclosure of the statements in a loop is done automatically by the editor and the type of loop is decided by the editor depending on the inputs given from the user.

Similar problems are faced in conditional operations where inappropriate use of if-else statements may lead to 'dangling else' problem resulting in ambiguous results. Errors of these kinds will be removed in the proposed editors as the user can group statements effectively with a few mouse clicks and the dangling else problem will never happen.

Centre for Knowledge Analytics and Ontological Engineering
KAnOE, PES Institute of Technology, Bangalore, India
http://kanoe.org
darsh.patel27@gmail.com, devashish.thakur11@gmail.com, drkavimahesh@gmail.com

Our proposed editor also allows the user to perform complex variables like addition of an array and a number without the use of any kind of loops. The user can simply declare variables without assigning the datatype. The data type is later decided by the editor depending on the type of information stored and automatic typecasting is done if needed.
The editor provides some inbuilt functions like print, average, sort, max, swap, min and factorial to make the computation process simpler. Compilers for the languages in case of compilation errors just show the errors and the line numbers and the user has to go to that particular line and has to find the error and correct it. We change the approach towards error correction entirely. All errors done in the editor are removed by the editor itself by interacting with the user through dialog boxes. Our editor acts as an interpreter between the user and the compiler. Errors are corrected directly by the interpreter by interacting with the user. By interaction the editor knows exactly what the user intends to do and it modifies the line appropriately without the user to go to that particular line. After interaction is over the error free code is sent to the compiler.

In between the user can see actual Java code of the algorithm written in the editor by simply moving to the code-view section provided. He can then shifts back to the algorithm view and continue writing the algorithm in the editor.

The rest of the paper is organized as follows. We first describe the prior work in section 2. Section 3 discusses the our new approach. Section 4 shows the implementation of the editor.We indicate our future work and conclude in section 6

## II. RELATED WORK

Alba-Mutka [1] states that the more it takes to learn the programming language and the use of the programming environment, the harder it is for the students to assimilate the general aspects of programming and problem solving.

Very high level language programming has been an area of research for almost 30 years. In this topic we discuss some of the prior work done in this field and the flaws that lead to its failure.
In pseudo code programming [2, 3, 4] visualization has been suggested as a method for enhancing programming learning, especially with novice students. Animation of these objects is usually utilized to visualize the changes in execution states. The user gets the platform to write a pseudo code to get the

visualization of the program where the mapping of language to the pseudo code has to be given by the user [2]. The tool uses a compiler that involves normal error correction mechanism where the error in the code has to be corrected by the user. Moreover the mapping provided is static in nature even if it is provided by the user [4]. For example if the user declares a=2 then the tool cannot make out that a is an integer and make it int a=2, which proves that the tool is not intelligent. There was also no option of executing the pseudo code directly [3].

The optimized compiler/interpreter [5] for very high level program was introduced an attempt to compile very high level programs. It used low-level C and thereby high-level constructs of the source program is lost. Compiled code is efficient, but cannot be debugged using user-level concepts. The low level C used makes the scalability of the compiler/interpreter a major issue.

High level/scripting languages [6, 7, 8, 9, 14, 17] have been proposed aiming to develop an environment which is less stringent towards rules of syntax as compared to system level programming. They are strongly typed and the abstract data types have to be passed through procedures [6]. Scripting cannot be used by novice as it is not meant to building an application from scratch. Scripting also results in more code and less flexible programs [7]. Object oriented languages have been used as very high level languages which are developed using the framework and libraries of Java [8]. This makes coding possible only in Java.

Non-procedural languages [10] with an interactive compiler have been used as an approach towards automated programming for Cobol/PL/I. The level of interactions is very shallow and effective GUI wasn't available for interactions which made the interaction only possible through the terminal.

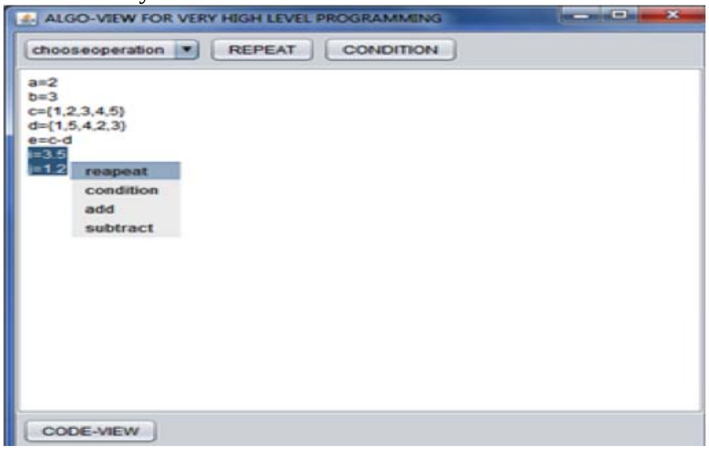
IDEs [11, 12, 13] provide the user an environment that makes the coding simpler. The compiler associated with the IDE intimates the user with the line number of error in case a logical or syntactical error occurs, but it doesn't provide a mechanism of correcting these errors automatically. Moreover the coding is made simple by features like drag-and-drop, spell checker, predictions etc., but ultimately the user needs to write the code in a particular programming language. Without the knowledge of the programming language the IDE is of no use to the user.

Very high-Level languages proposed in the paper intend to increase the programmer's productivity by easing the programmer's task in a way that enhances the user's reliability and understanding of code. It frees the programmer from details which are not relevant to the problem he is solving. It provides the user with high level of abstractions which hides irrelevant details thereby increasing the accuracy and the quality of the programmers. The user can now concentrate more on the logic involved in programming rather than the syntax of the language. The complete approach reduces the learning curve involved in learning a language. The editor can also help to teach the beginners the basic control structures and operational logic independent of any programming logic. Later it can be mapped onto Java code which will help in better understanding of the language.

## III. APPROACH

In normal programming language we need to define a variable before we use it along with the datatypes. This acts as a burden for the user as he doesn't know what kind of data the variable can hold in future. We propose an editor that allows the user to declare the variable directly without worrying about the definition. The definition is provided implicitly depending on the type of value stored. Array manipulation can also be done directly without explicitly writing any loops.

Normal programming languages offers three types of looping constructs. Many times the user is unsure of the looping construct he must use in a particular program and invalid use may lead to logical errors that are difficult to debug. Our approach tries to interact with the user and know the type of looping construct he intends to use in the program. The construction of the loop is done intuitively without the need of writing the actual code. The communication happens with the help of dialog boxes in non-programming terms so that the user doesn't get confused with the jargons involved with programming.



```
a=2
b=3
c={1,2,3,4,5}
d={1,5,4,2,3}
e=c-d
i=3.5
i=1.2
```

Fig 1 – Algo view of the editor

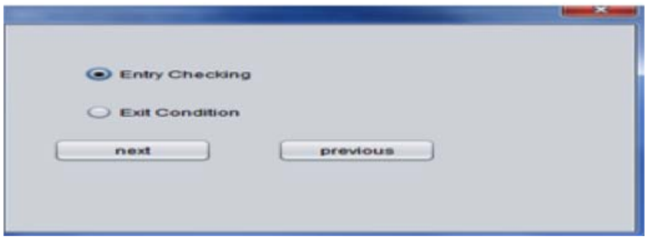


Fig 2 – Entry vs Exit condition

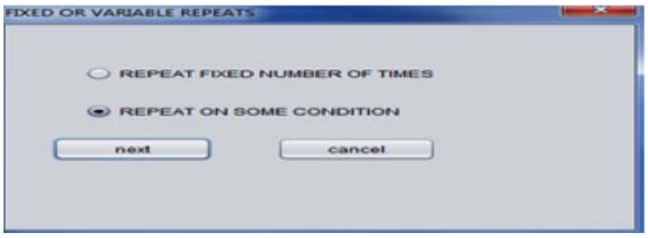
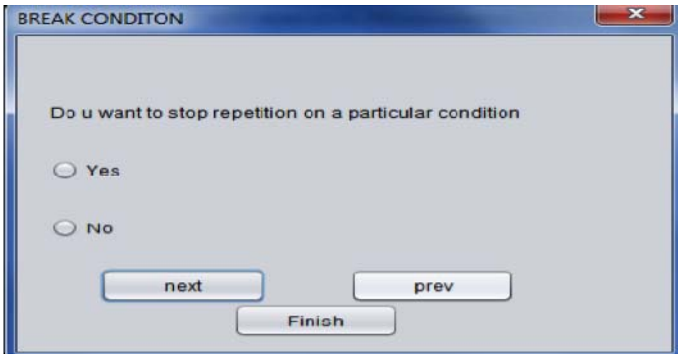
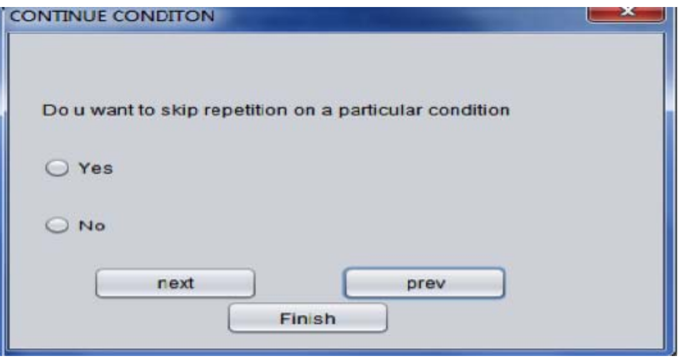


Fig 3 – Fixed vs Variable repeats



Do u want to stop repetition on a particular condition

Fig 4 – Break condition



Do u want to skip repetition on a particular condition

Fig 5 – Continue condition

## IV. OUR SOLUTION

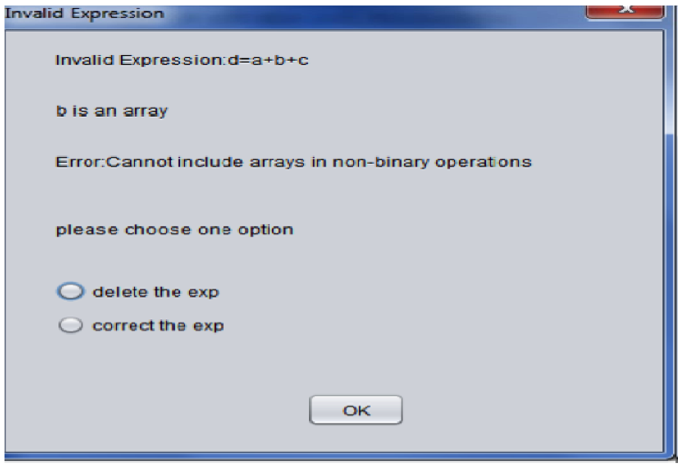
Fig.1 shows the overview of the algorithm. The user can bound statements in a loop by simply highlighting the required text. An interactive dialog box pops which tries to know the kind of loops the user wants. Fig. 2 shows the dialog boxes which checks if the user wants an entry condition or an exit condition. Similarly the dialog box shown in Fig. 3 inquires about the structure of the loop. This interactive approach not only reduces the chances of having manual errors in the loop but also eases out the construction of the loop.

The construction of loops also involves optional break and continue conditional that helps the user to jump out of the loop or skip the loop on a particular user defined condition. Fig. 4 shows the break condition whereas Fig. 4 shows the continue condition. The conditions entered are added to the loop by the editor itself and duplicate entries of the same break or continue condition is removed by the editor. The editor also makes sure that the break condition isn't a part of any while condition or any if condition within the same loop.

Similar to loops, the process of entering a condition involves interaction with the user. The Dangling else problem occurs in programming languages when there are more if's than else in a conditional statement. This results in ambiguous association of an else with an if which gives logical errors that are difficult to debug. Our approach tries to remove this problem by bounding statements automatically within an if or else clause depending on the input from the user. This approach not only prevents dangling else problem but also makes it easier and intuitive for the user to write conditional statements in a program.

Our approach also provides user with various inbuilt functions like sort, print, swap, min, max, search, average and factorial. The user can select any function and specify the arguments for the same.

One of the most important aspect of the editor is the process of error detection and correction. Compilation errors in any programming language is shown by simply specifying the line number of the erroneous line along with the cause of the error.In this editor we try a different approach towards error correction where the editor displays the error and it interacts with the user and tries to resolve the error. The user need not know the particular line or correct it himself. Errors like uninitialized variables(b=a with the value of 'a' not given) as shown in Fig. 7 lead to popping of a dialog box which tells the error and asks the user to initialize or change the statement.Invalid use of operators (a++b) or invalid assignments in the algorithm will lead to invalid expression dialog box and the user will have to delete or correct the expression explicitly.Once the user makes a valid change the change in the code is done automatically.



Invalid Expression

Invalid Expression:d=a+b+c

b is an array

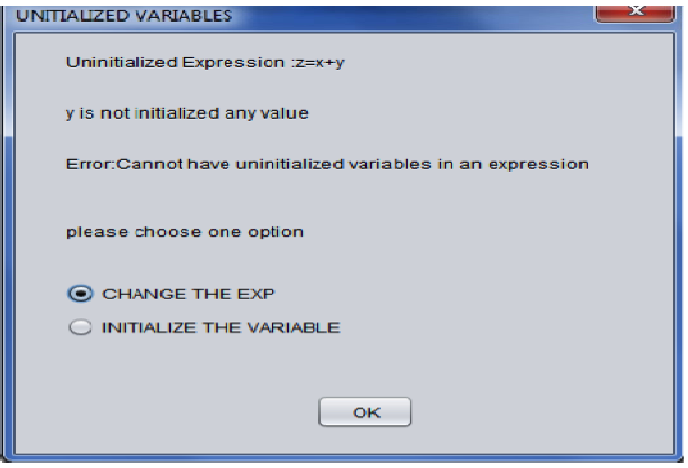
Error:Cannot include arrays in non-binary operations

please choose one option

○ delete the exp
○ correct the exp

OK

Fig 6 – Invalid Expression



UNITIALIZED VARIABLES

Uninitialized Expression :z=x+y

y is not initialized any value

Error:Cannot have uninitialized variables in an expression

please choose one option

[✓ CHANGE THE EXP]
○ INITIALIZE THE VARIABLE

OK

Fig 7 – Uninitialized expression

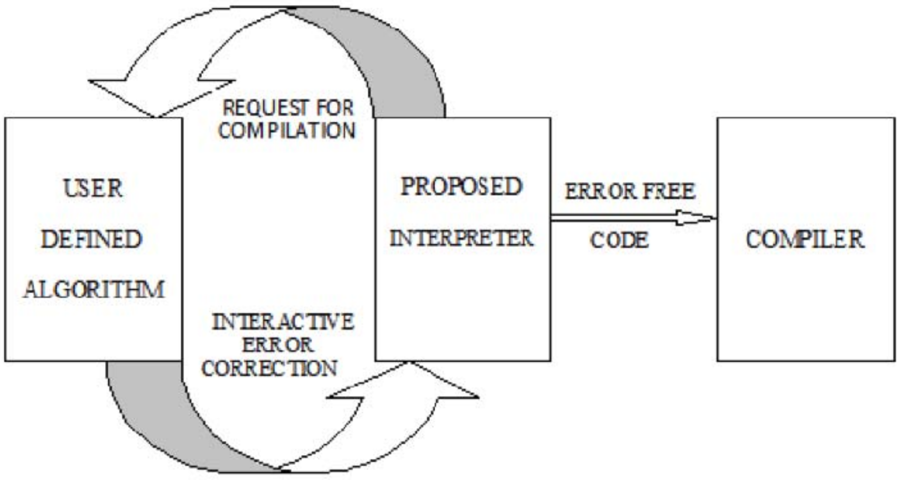


Fig 8 – Working of interpretor

As shown in Fig. 8 the interaction between the user and the interpreter for the purpose of error correction takes place in the form of a loop till all the errors are corrected. The advantage the interpreter has over compilers is that it even corrects the logical errors of the program as well as the syntactical errors. The error-free code is then saved in a .java file which can be sent for compilation. The interpreter makes sure that the final output is error-free.

## V. IMPLEMENTATION

The editor proposed in the paper is inplemented in open source Java programming language.The GUI for the editor is developed using javax.swing API and the code running in the background is written in Java.

## VI. CONCLUSION AND FUTURE WORK

In this paper we proposed a new approach to programming that may be very useful for novice programmers who are not used to the syntax of a programming language. The editor may also makes the programming simpler as the user can program with mouse clicks and writing of long codes for loops and conditional statements is minimized. The inbuilt functions may further simplify the programming. We also propose an interactive way of resolving errors in the program which may be more easily understood by the user.

In future the work done in the editor can be extended for the entire programming language so that the user need not know the syntax of any language. He can simply type algorithm independent of any syntax and execute it to get the result.

## ACKNOWLEDGMENT

This work is supported in part by the World Bank/Government of India research grant under the TEQIP programme (subcomponent 1.2.1) to the Centre for Knowledge Analytics and Ontological Engineering (KAnOE) at PES Institute of Technology, Bangalore, India.

## REFERENCES

[1] Ala-Mutka, K. 2005. Ohjelmoinnin opetuksen ongelmiaja ratkaisuja. Tekniikan opetuksen symposium 20.-21.10.2005. Helsinki University of Technology.http://www.dipoli.tkk.fi/ok/p/reflektori/verkkojulkaisu/index.php?p=verkkojulkaisu.

[2] Define and Visualize Your First Programming LanguageMikko-Jussi Laakso, Erkki Kaila, Teemu Rajala & Tapio Salakoski University of Turku, Turku Centre for Computer Science, Turku, Finland

[3] Olsen, A.L. (2005). Using Pseudocode to Teach Problem Solving. Journal of Computing Sciences in Colleges, 21(2):231-236.

[4] Effectiveness of Program Visualization: A Case Study with the ViLLE Tool Mikko-Jussi Laakso, Erkki Kaila, Teemu Rajala & Tapio Salakoski University of Turku, Turku Centre for Computer Science, Turku, Finland

**[5]** Debugging a High Level Language via a Unified Interpreter and Compiler Runtime Environment Jinlong Cai, Marc Moreno Maza, Stephen Watt Ontario Research Center of Computer Algebra University of Western Ontario {jcai,moreno,watt}@scl.csd.uwo.ca Martin Dunstan Department of Applied Computing University of Dundee, UK mdustan@computing.dundee.ac.uk

[6] Programming with abstract datatypes, Barbara Liskov and Stephen Zilles

[7] Scripting: Higher Level Programming for the 21st Century John K. Ousterhout Sun Microsystems Laboratories 901 San Antonio Rd., MS UMTV29-232 Palo Alto, CA 94303-4900 john.ousterhout@eng.sun.com

[8] Very High Level Programming with Collection Components Mark Evered, Gisela Menger University of Ulm

[9] The SIMPLE Language Robert M. AkscynComputer Science DepartmentThe University of Waikato, New Zealandra33@cs.waikato.ac.nz

[10] Use of a Nonprocedural SpecificationLanguage and Associated Program Generator in Software Development N. S. PRYWES University of PennsylvaniaA. PNUELI TeI-Aviv University and S. SHASTRYBell Telephone Laboratories

[11] NETBEANS 6.9 http://netbeans.org/

[12] Eclipse www.eclipse.org

[13] JCREATOR www.jcreator.com

[14] Define and Visualize Your First Programming Language Mikko-Jussi Laakso, Erkki Kaila, Teemu Rajala & Tapio Salakoski University of Turku, Turku Centre for Computer Science, Turku, Finland {milaak, ertaka, temira, sala}@utu.fi

[15] Software Prototyping using the SETL Programming Language Philippe Kruchten, Edmond Schonberg, and Jacob Schwart New York university.

[16] Supporting High Andrew Chien Level Programming with High Performance: The Illinois Concert System Julian Dolby Bishwaroop Ganguly Vijay Karamcheti Xingbin Zhang Department of Computer Science University of IllinoisUrbana , Illinois

[17] What programmers should know-By J T Schwartz

[18] Programming Languages: Fundamental Concepts for Expanding and Disciplining the Mind Mitchell Wand College of Computer and Information Science Northeastern University, Daniel P. Friedman Computer Science Department Indiana University.