International Journal of Science Academic Research
Vol. 07, Issue 04, pp.698-703, April 2026
Available online at http://www.scienceijsar.com
ISSN: 2582-6425

Research Article

DRUVA: A PROGRAMMING LANGUAGE WITH SOPHISTICATED IDE AND USER FRIENDLY SYNTAX FOR NOVICE LEARNERS

***Pothukuchi Ram Naresh**

IIIT-Hyderabad, Visakhapatnam, India

Received 13th January 2026; **Accepted** 16th February 2026; **Published online** 20th April 2026

Abstract

The idea is to build a new programming language and its IDE with name *Druva*. This programming language should be more user friendly than existing programming languages like PHP, Python or Java. It should have both compiler and interpreter. A programming language is a formal, engineered language composed of a set of instructions used to write computer programs and applications. This new programming language should be able to solve complex programs in a simpler way. *Druva* should be suitable to novice, intermediate and expert learners. In this version of *Druva,* we are not proposing OOPs concepts. The programming languages act as a bridge between human logic and machine execution, allowing developers to translate ideas into instructions that computers can understand and follow. This proposed programming language will be simple, secure, portable, robust, architectural neutral, distributed and dynamic. This programming language will have data types, variables, arrays, operators, control statements and library. This programming language will be systematic, efficient and will provide a logical approach to problem solving. It will be a step-by-step approach.

Keywords: Programming language, Simple, IDE.

INTRODUCTION

A programming language is a formal set of rules, syntax, and commands used by developers to write code, allowing them to instruct computers to create software, websites, and applications. These languages bridge human logic with machine execution, converting human-readable instructions into binary code that computers understand.

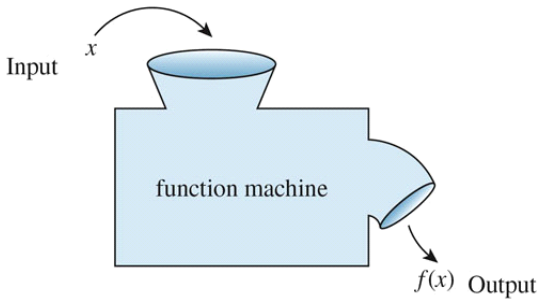

Fig. 1. Machine input and output

Programming languages are crucial because they serve as the foundation for modern technology, enabling human-computer communication, fostering innovation, enhancing problem-solving skills, and opening up numerous career opportunities. Computers cannot understand human languages directly; they require precise, structured instructions in the form of code (which is eventually translated into binary 1s and 0s). Programming languages act as the essential bridge, allowing humans to turn complex ideas into commands that machines can understand and execute. Nearly every modern technological advancement from smartphones and social media to artificial intelligence (AI), blockchain, and advanced medical equipment is built on programming.

Corresponding Author:** ***Pothukuchi Ram Naresh,
IIIT-Hyderabad, Visakhapatnam, India.

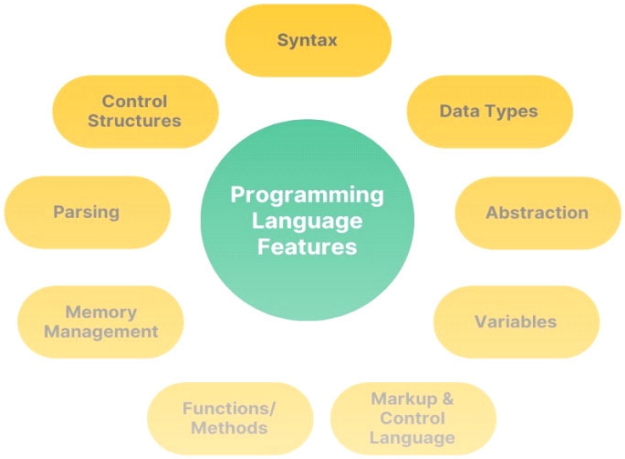

Fig. 2. Programming Language Features

As new languages and improvements to existing ones emerge, they open the door to previously unimaginable possibilities, such as AI algorithms for data analysis or new technologies like Mojo for AI development. Learning to program involves breaking down large, complex problems into smaller, more manageable steps to find systematic and efficient solutions. This process sharpens critical thinking, logical reasoning, and analytical skills, which are valuable in many aspects of professional and personal life. Programming enables the automation of repetitive and time-consuming tasks, which not only frees up human time for more important work but also reduces the chance of human error in processes like data entry or software testing. We need programming languages primarily to communicate instructions to computers in a structured, human-readable form to automate tasks, solve problems, and create software. Computers only understand low-level machine code (binary), which is extremely difficult for humans to write and understand directly. Programming languages act as a necessary abstraction layer. They allow humans to express logic and ideas using understandable syntax

(like words and mathematical symbols), which is then translated by a compiler or interpreter into the binary instructions a computer's CPU can execute. Writing directly in machine code would be incredibly time-consuming and error-prone. High-level languages offer features like abstraction, libraries (collections of pre-written code), and debugging tools that significantly speed up development, make the code easier to read, and simpler to maintain over time. They allow people to write instructions for computers to follow and create the technology we use every day. Programming languages are essential in technology because they enable us to automate tasks, make calculations, and process large amounts of data quickly and efficiently. A compiler translates the entire source code into machine code at once before execution, creating an executable file, while an interpreter translates and executes code line-by-line during runtime. Compilers offer faster execution speeds and better optimization, whereas interpreters provide faster debugging and easier error detection. Interpreters are better for debugging because they stop execution immediately when an error is found, whereas compilers report errors only after scanning the whole program. Programming languages are defined by their syntax, data types, and control structures, serving as formal rules for instructing computers. Key characteristics include simplicity, readability, efficiency, writability, and portability, allowing developers to create maintainable and reliable applications while managing complex computations, memory, and hardware interfaces.

Syntax, Sample problems and Proposed Druva IDE

In this programming language we will see some sample examples. An Integrated Development Environment (IDE) is a software suite that consolidates essential developer tools code editor, debugger, and build automation into a single graphical user interface (GUI) to enhance productivity. IDEs streamline development, offering syntax highlighting, code completion, and version control, departing from using separate tools like compilers and editors.

Fig. 3. Printing Simple Statement

IDEs offer a unified experience that saves time by reducing the need for manual tool integration, allowing developers to focus on building applications rather than managing the environment. They are crucial for improving, organizing, and speeding up the software development process. In fig 3.0 we can visualize outputs of sample print statements. In this program the execution is done successfully. When the user clicks "RUN" button then IDE executes the program and it is displayed inside "OUTPUT" box. To execute any program the user has to click the "RUN" button in order to execute the written program. Similarly in Fig 4.0 there is a program where addition takes place. There are two variables x and y in the program and values are assigned to those variables respectively. This program also executes successfully in Druva IDE.

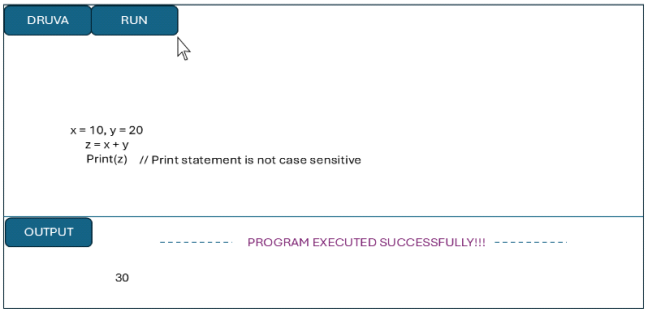

Fig. 4. Addition Program in Druva IDE

To continue further we have another program in Fig 5.0 which does subtraction. When user clicks the "RUN" button the output is successfully displayed.

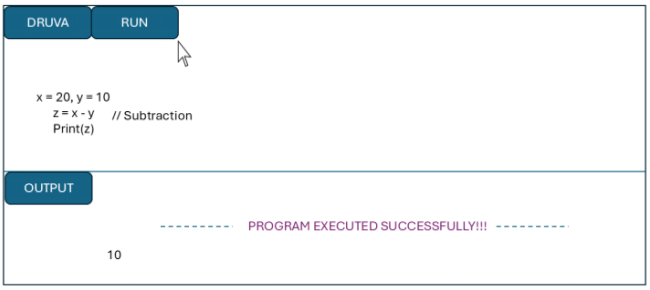

Fig 5.0: Subtraction Program in Druva IDE

We have another program in Fig 6.0 where multiplication is performed. It also executes successfully.

Fig. 6. Multiplication Program in Druva IDE

In Fig 7.0 we can visualize division between two numbers stored in x and y variables.

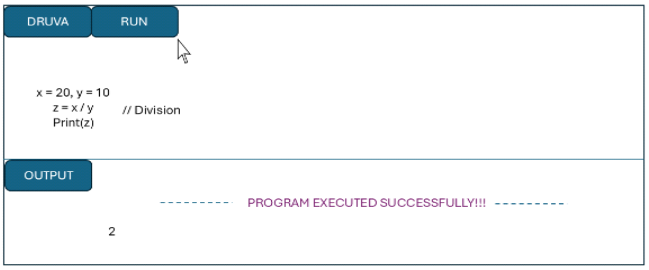

Fig 7.0: Division Program in Druva IDE

There are incidents when errors also occur in the IDE. We have one such sample example shown in Fig 8.0. The sample example shows that PRINT statement is not written completely and "RUN" button is clicked to execute it.

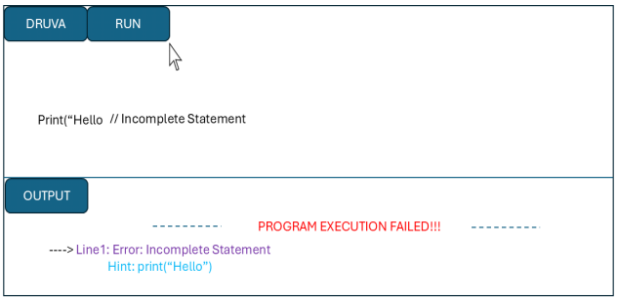

Fig. 8. Error Program in Druva IDE

The above program is failed. The errors are shown in particular line numbers. At the same time this IDE has a quality of providing "Hint" to programmers. The programmer can very well make use of the "Hint" provided by IDE to successfully execute the code.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Fig. 9. Arithmetic Operators in Druva

In the above fig 9.0 we can see the arithmetic operators proposed for Druva language with example.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Fig. 10. Comparison Operators in Druva

In the above fig 10.0 we can visualize the comparison operators described for Druva programming language.

Compiler and Interpreter

Druva require both compiler and interpreter to execute its programs. A compiler translates the entire high-level program into machine code at once before execution, producing an independent executable file that runs quickly. Conversely, an interpreter translates and executes code line-by-line at runtime, which is slower but makes debugging easier. A compiler translates the entire source code into intermediate machine code (object code) all at once. An interpreter translates the code line-by-line, translating and executing one instruction at a time. Compiled programs generally run faster because they are already translated. Interpreted programs are slower because translation happens during execution. A compiler identifies errors in all lines, reporting them only after scanning the entire code. Compilers translate the entire source code into native machine code once, allowing it to run quickly, while interpreters translate code line-by-line during runtime, creating overhead. An interpreter stops execution when it encounters an error, showing errors on a line-by-line basis. Some key characteristics of compiler are below**:-**

- **Execution Speed**: Since the code is fully compiled before execution, compiled programs generally run faster.
- **Error Checking**: The compiler checks for syntax errors in the entire program, often highlighting all issues at once before any code runs.
- **Executable Output**: Compilers produce an executable file (like .exe on Windows) that can be run without further translation.
- **Storage**: The compiled code is stored as machine language and is often larger than the source code.

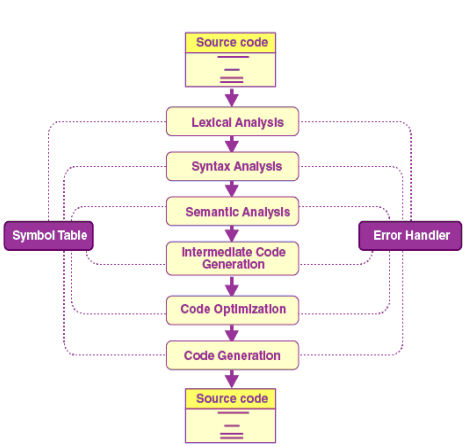

Fig. 11. Compiler Block diagram

Compilers are significantly faster than interpreters at executing programs. However, interpreters provide faster initial development and debugging because they skip the separate, long compilation step. Interpreters start faster and are better for debugging because they stop at the first error, whereas compilers require full recompilation after changes. A compiler translates the entire code into machine language in a single operation, creating an executable file that can be run on a target system. Compilers are typically used for languages where execution speed is crucial, such as C and C++.

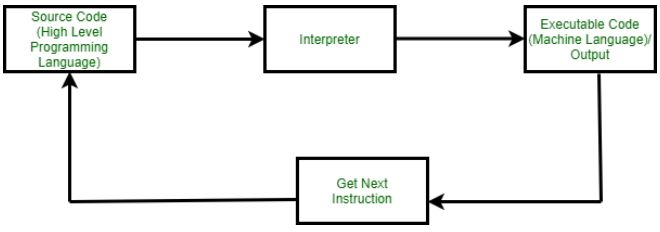

Fig. 12. Interpreter Block diagram

The interpreter takes each line of code and translates it into machine code on the fly. This means there is no separate compilation step, making interpreters ideal for testing and debugging. However, this line-by-line execution often leads to slower program performance compared to compiled languages. An interpreter is a tool that reads and executes code line-by-line, translating each line into machine code as the program

runs. Interpreters are commonly used in scripting languages and other high-level programming languages where dynamic execution is crucial. An interpreter, on the other hand, translates code line-by-line or statement-by-statement as it is executed, without creating a separate executable file. This real-time translation approach allows programs to start running immediately after coding, even if the entire program hasn't been translated. Some key Characteristics of an Interpreter are below:-

- **Execution Speed**: Interpreted code tends to run slower since each line is translated on-the-fly during execution.
- **Error Checking**: Errors are caught as each line is translated, meaning you may only encounter errors one at a time.
- **No Executable Output**: Unlike a compiler, an interpreter doesn't produce a separate executable file, so the code must be interpreted each time it's run.
- **Flexibility**: Interpreters are often used in environments where real-time feedback is essential, like scripting, testing, or interactive programming.

Control statements and functions

The control statements in programming dictate the execution flow sequential, conditional, or iterative—rather than just running top-to-bottom. They enable decision-making (if-else, switch) and repetition (for, while loops) to handle dynamic input and complex logic, essential for building flexible programs.

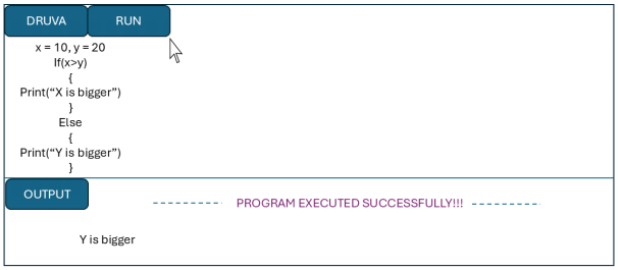

Fig. 13. A Sample if-else Program in Druva

They make programs interactive and capable of making decisions. They eliminate the need to write the same code multiple times. They allow for organized branching in complex applications. Control flow statements are fundamental components of programming languages that allow developers to control the order in which instructions are executed in a program. They enable execution of a block of code multiple times, execute a block of code based on conditions, terminate or skip the execution of certain lines of code, etc. The if-else statement Executes a block of code if a specified condition is true, and another block if the condition is false. The for block Executes a block of code a specified number of times, typically iterating over a range of values.

Fig. 14. Another Sample Program in if-else

The if-else statement is used to execute one block of code if a specified condition is true, and another block of code if the condition is false**.**

Fig 15.0: Printing an Array with numbers

The Looping statements, also known as iteration or repetition statements, are used in programming to repeatedly execute a block of code.

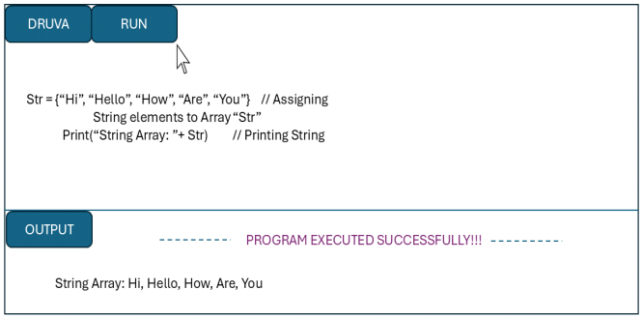

Fig. 16. Printing a String Array

They are essential for performing tasks such as iterating over elements in a list, reading data from a file, or executing a set of instructions a specific number of times.

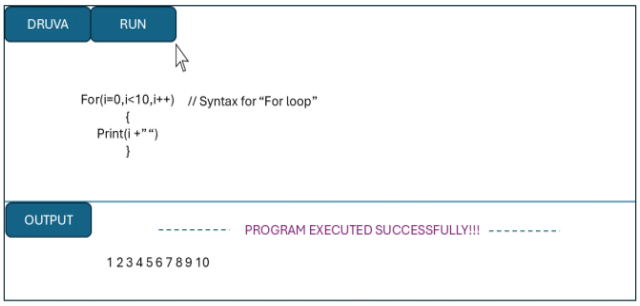

Fig. 17. Printing a For Loop

Below in Fig 18.0 we can visualize an example of inbuilt functions. These inbuilt functions are printed using "Print" statement. The out is successfully printed.

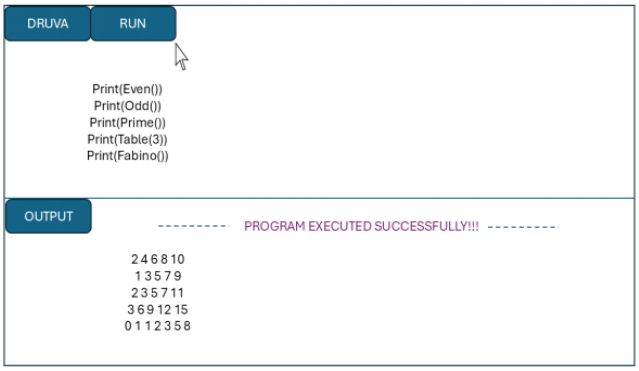

Fig. 18. Printing Inbuilt Functions

Similarly in Fig 19.0 there is another example of function. In this example function calling is done and function definition is mentioned.

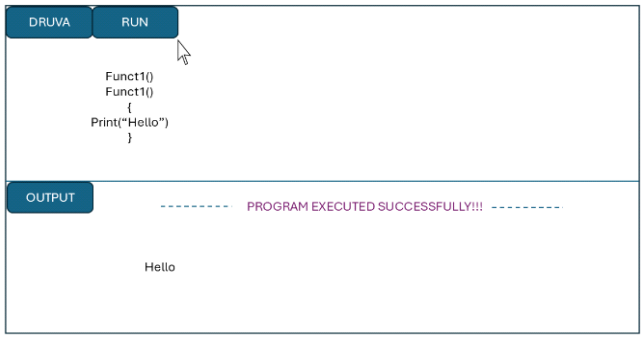

Fig. 19. An example of Function call

In computer programming, a function is a self-contained block of organized, reusable code designed to perform a single, related action. Functions provide modularity, making programs easier to read, test, and maintain by breaking complex problems into smaller, manageable pieces.

Conclusion

Druva is newly proposed programming language designed especially for novice learners. In this we use sophisticated *Druva* IDE and, in the background, we have both compiler and interpreter which run parallel. *Druva* is simple, robust, portable and distributed language. *Druva* syntax is made simpler for novice learners. We can also make use of libraries in *Druva* IDE. We believe that in next version we will bring numerous changes with more features. The researchers, programmers and novice learners can make use of *Druva* programming language for complex problems. Above we mentioned simple programs, if-else, loops, arrays and functions. *Druva* has capability to have more library functions. In future the developers should build the *Druva* IDE in a sophisticated manner to have better algorithms and to ease developer's issues.

REFERENCES

1. Parambil A. J., A. S. Mishra, S. Chakravarty and Y. Pingle, "Saarthi: A Programming Language Designed to Introduce Coding to High Schoolers," *2024 11th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2024, pp. 384-389, doi: 10.23919/INDIACom61295.2024.10498555.
2. Yalagi P. S., T. S. Indi and M. A. Nirgude, "Enhancing the Cognitive Level of Novice Learners Using Effective Program Writing Skills," *2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, Mumbai, India, 2016, pp. 167-171, doi: 10.1109/LaTiCE.2016.14.
3. Dorelo A. A. and S. da Rosa Zipitría, "Studying novice learners knowledge about program execution," *2017 XLIII Latin American Computer Conference (CLEI)*, Cordoba, Argentina, 2017, pp. 1-8, doi: 10.1109/CLEI.2017. 8226370.
4. Uppsäll C., A. Nylén and G. Dodig-Crnkovic, "Novice Students Explain: What is Computer Science?," *2024 IEEE Frontiers in Education Conference (FIE)*, Washington, DC, USA, 2024, pp.1-9, doi:10.1109/FIE61694.2024. 10893370.
5. Sim T. Y. and S. L. Lau, "Online Tools to Support Novice Programming: A Systematic Review," *2018 IEEE Conference on e-Learning, e-Management and e-Services (IC3e)*, Langkawi, Malaysia, 2018, pp. 91-96, doi: 10.1109/IC3e.2018.8632649.
6. Pongpakatien P. and A. Sipitakiat, "An Iterative Learning Approach for Novice Engineering Students to Grasp Important Concepts in Autonomous Systems," *TENCON 2023 - 2023 IEEE Region 10 Conference (TENCON)*, Chiang Mai, Thailand, 2023, pp. 1-6, doi: 10.1109/TENCON58879.2023.10322547
7. Rababaah, A. R. "A New Simple Programming Language for Education," *2020 15th International Conference on Computer Science & Education (ICCSE)*, Delft, Netherlands, 2020, pp. 145-149, doi:
8. Ibne Amin M. F., M. M. Rahman, Y. Watanobe and M. M. Daniel, "Impact of Programming Language Skills in Programming Learning," *2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Penang, Malaysia, 2022, pp. 271-277, doi: 10.1109/MCSoC57363.2022.00050.
9. Mekouar, L. "The Art of Teaching Programming Languages: Challenges and Accomplishments," *2022 IEEE World Engineering Education Conference (EDUNINE)*, Santos, Brazil, 2022, pp. 1-6, doi: 10.1109/EDUNINE53672.2022.9782372.
10. Grindei L., C. Constantinescu, A. Bojita, R. Holonec and L. Rapolti, "Project-oriented approach in teaching programming to the first year undergraduate students in Electrical Engineering," *2023 10th International Conference on Modern Power Systems (MPS)*, Cluj-Napoca, Romania, 2023, pp. 1-4, doi: 10.1109/MPS58874.2023.10187591.
11. Wagner, E. G. "On the structure of programming languages, or, six languages for turing machines," *8th Annual Symposium on Switching and Automata Theory (SWAT 1967)*, Austin, TX, USA, 1967, pp. 45-54, doi: 10.1109/FOCS.1967.26.
12. Nothen E. and L. M. de Fátima Mastroianni, "Data structures, from theory to bits: Using theory of formal languages to analyze structured data," *IEEE CACIDI 2016 - IEEE Conference on Computer Sciences*, Buenos Aires, Argentina, 2016, pp. 1-6, doi: 10.1109/CACIDI.2016. 7785981.
13. Lawan A. A., A. S. Abdi, A. A. Abuhassan and M. S. Khalid, "What is Difficult in Learning Programming Language Based on Problem-Solving Skills?," *2019 International Conference on Advanced Science and Engineering (ICOASE)*, Zakho - Duhok, Iraq, 2019, pp. 18-22, doi: 10.1109/ICOASE.2019.8723740.
14. Chafle G. *et al*., "An Integrated Development Environment for Web Service Composition," *IEEE International Conference on Web Services (ICWS 2007)*, Salt Lake City, UT, USA, 2007, pp. 839-847, doi: 10.1109/ICWS.2007.38.
15. Ertl D. and H. Krapfenbauer, "A Case Study of Developing an IDE for Embedded Software Using Open Source," *2009 Fourth International Conference on Software Engineering Advances*, Porto, Portugal, 2009, pp. 191-196, doi: 10.1109/ICSEA.2009.38.
16. Tran H. T., H. H. Dang, K. N. Do, T. D. Tran and Vu Nguyen, "An interactive Web-based IDE towards teaching and learning in programming courses," *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, Bali, Indonesia, 2013, pp. 439-444, doi: 10.1109/TALE.2013. 6654478.

17. Iyer G. N., A. G. Yisheng, C. H. Er Metilda, W. X. Choong and S. W. Koh, "A Web-Based IDE for DevOps Learning in Software Engineering Higher Education," *2024 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, Bengaluru, India, 2024, pp. 1-8, doi: 10.1109/TALE62452.2024.10834361.
18. Cavazos, J. "Intelligent compilers," *2008 IEEE International Conference on Cluster Computing*, Tsukuba, Japan, 2008, pp. 360-368, doi: 10.1109/CLUSTR.2008. 4663796.
19. Ahmed H. *et al*., "Selecting the Best Compiler Optimization by Adopting Natural Language Processing," in *IEEE Access*, vol. 12, pp. 121700-121711, 2024, doi: 10.1109/ACCESS.2024.3451516. keywords:
20. Datta A. and A. K. Paul, "Online compiler as a cloud service," *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, Ramanathapuram, India, 2014, pp. 1783-1786, doi: 10.1109/ICACCCT.2014.7019416
21. Martins do Rosario V., M. Mikio Hato, R. Azevedo and E. Borin, "A Methodology for Optimization of Interpreters," *2018 Symposium on High Performance Computing Systems (WSCAD)*, Sao Paulo, Brazil, 2018, pp. 205-212, doi: 10.1109/WSCAD.2018.00040.
22. Xiao X. and Y. Xu, "The Design and Implementation of C-like Language Interpreter," *2011 2nd International Symposium on Intelligence Information Processing and Trusted Computing*, Wuhan, China, 2011, pp. 104-107, doi: 10.1109/IPTC.2011.33.
23. Keerthi Kumar M., B. D. Parameshachari, A. Vizzarri, M. S. Nagesh and H. A. Deepak, "Implementation of Machine Learning Based Interpreter for Real Time Sign Language Detection and Action Recognition," *2024 Second International Conference on Networks, Multimedia and Information Technology (NMITCON)*, Bengaluru, India, 2024, pp. 1-6, doi: 10.1109/NMITCON62075.2024. 10699131
24. Inturi S. and M. Swamydas, "Programming Assignment Grading Through Control Statement and Program Features," *2023 International Conference on Emerging Techniques in Computational Intelligence (ICETCI)*, Hyderabad, India, 2023, pp. 122-129, doi: 10.1109/ICETCI58599.2023.10331134.
25. Al-Hashimi M., M. Saleh, O. Abulnaja and N. Aljabri, "Evaluation of control loop statements power efficiency: An experimental study," *2014 9th International Conference on Informatics and Systems*, Cairo, Egypt, 2014, pp. PDC-45-PDC-48, doi: 10.1109/INFOS.2014.7036676.
26. www.w3schools.com
27. www.codefinity.com
28. www.medium.com/@ashishkuma rjena1437/
29. https://byjus.com/gate/phases-of-complier-notes/
30. https://www.geeksforgeeks.org/c/c-arrays/

****