Emen, AST-based programming language

Bexultan Tokan, Bekzat Kabdrashev, Marat Serikbayev, Nurlybek Surpkelov
Adviser(s): Hashim Ali

1. Executive Summary (10%)

The goal of our senior project, "Emen," is to overcome the traditional text-based programming languages' difficult learning curve and error-prone character, which deter many newcomers. Emen is a programming language that takes a new approach by using an Abstract Syntax Tree (AST) and combining a customized editor and compiler to reduce common syntactic and semantic problems. The constraints found in current programming environments and the advantages of visual languages like Scratch for education served as the inspiration for this project. By making programming more approachable and less daunting for novices while maintaining its power and flexibility for more experienced programmers, Emen seeks to transform the programming industry.

Our team used the Agile process to create Emen, a functional programming language with a feature set, by utilizing development technologies including Git, Go, and Raylib. Type and variable declarations, function calls, support for logical and arithmetic operations, static arrays, and control structures like while loops and if-else statements are a few examples of these. By enabling direct interaction and editing in the user interface (UI) and providing a visual representation of the code in AST format, Emen's editor guarantees a syntax-error-free writing experience.

Significant improvements were made in the spring after the programming language was successfully implemented with an operating editor and compiler by the end of the fall semester. A revamped user interface (UI) influenced by the color scheme of VS Code, a reorganized code base for easier navigation and maintenance, and the transition from an array to a linked-list structure for effective code node management were among the modifications. Notably, these updates improved the backend operations by fixing reference problems and drastically reducing time complexity.

As it stands now, Emen set the stage for future advancements. Future work ideas include investigating automatic mistake correction, making sure the language is cross-platform compatible, and extending its capabilities to accommodate object-oriented programming concepts. Emen aims to become a standard instrument in programming education and a driving force behind innovation in software development methodologies.

2. Introduction (10%)

Modern programming languages are quite difficult for novices to learn, primarily due to their intricate prerequisites. These challenges are frequently made worse by small mistakes that can lead to tedious syntactic and semantic problems and poor coding techniques. Finding and fixing these problems is a significant challenge for developers, made more difficult by the requirement for understandable code and the limitations of compiler error reporting features. Programming languages do not currently require code to be formatted for readability; programmers are responsible for this and also have to take care of syntactic and semantic problems. These elements require a significant amount of a programmer's attention and time, even though they have no direct impact on the logic of the program.

Compilers do not provide comprehensive static error reporting, which is another major issue. Compilers are often good at finding type mistakes, but only in the absence of syntactic issues. For instance, the compiler may refuse to process the remaining code in a C++ program if the semicolon at the beginning of the program is absent.

As part of our project, we created a system named "Emen" that handles the first problem by controlling code formatting and syntax. Emen is a programming language that is built on AST, and it comes with an editor and compiler designed specifically for it.

3. Background and Related Work (15%)

The team project idea was inspired by The Dion Systems[1] research project which served as a key source of inspiration. The Dion project highlighted the inherent limitations and challenges of the text-file based languages. Developers often grapple with cumbersome syntax rules, ambiguous semantics, and time-consuming debugging processes. These challenges not only create barriers for beginner programmers but also hinder the productivity of experienced developers. Moreover, the visual based programming language called Scratch[2] also accomplishes making coding more accessible and intuitive. Its original purpose is to help children to code by using the visual nature of the program to make it engaging. A key feature of Scratch is that the code is represented visually as interconnected blocks. Children can observe the logic and structure of their code clearly without paying attention to textual syntax. This visual abstract syntax tree (AST) format helps beginner programmers to focus on computational thinking. The coding environment in Scratch gives immediate visual feedback making it easier to debug the program. Expanding on these concepts from Dion and Scratch, our project develops a new format for source code representation to address the issues with syntax, semantics and debugging.This alternative programming format could supersede traditional systems by making it more intuitive and efficient.

4. Project Approach (20%)

Agile methodology was used by our team. The main objective of our project implementation was programming in order to reach our initial ideas. We used development tools such as Go, Raylib and Git. As a result we implemented the programming language with a developed editor and compiler in it. The programming language had following functionalities:

- type definitions
- variable declarations
- function declarations, function call
- arithmetic operators (+, *, -, /, %), logical operators (&&, ||, !)
- static arrays
- printing for strings and integers
- if, else, while, return
- suggestions of variable and function names

We can see all the functionalities in the editor display and edit directly on the display. The editing tool makes certain that there are no syntax errors.

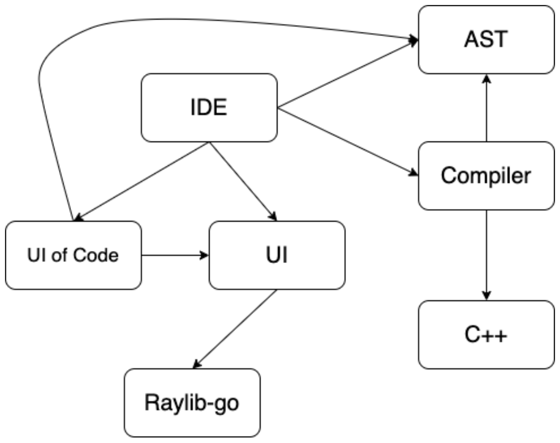

Fig 1. Architecture of a project

This diagram outlines the structure of a software project for an Integrated Development Environment (IDE). The IDE interacts with several components: it shapes the User Interface (UI), which includes a specialized section for code (UI of Code), and connects to an Abstract Syntax Tree, which is essential for understanding and managing the structure of the code. The IDE also interfaces with a Compiler that translates the AST into C++ code, and utilizes Raylib-go, a graphics library for Go language, to enhance the UI.

```
fun factorial(n int32) int32
    if n < 2
        return 1
    return n * factorial(n - 1)
```

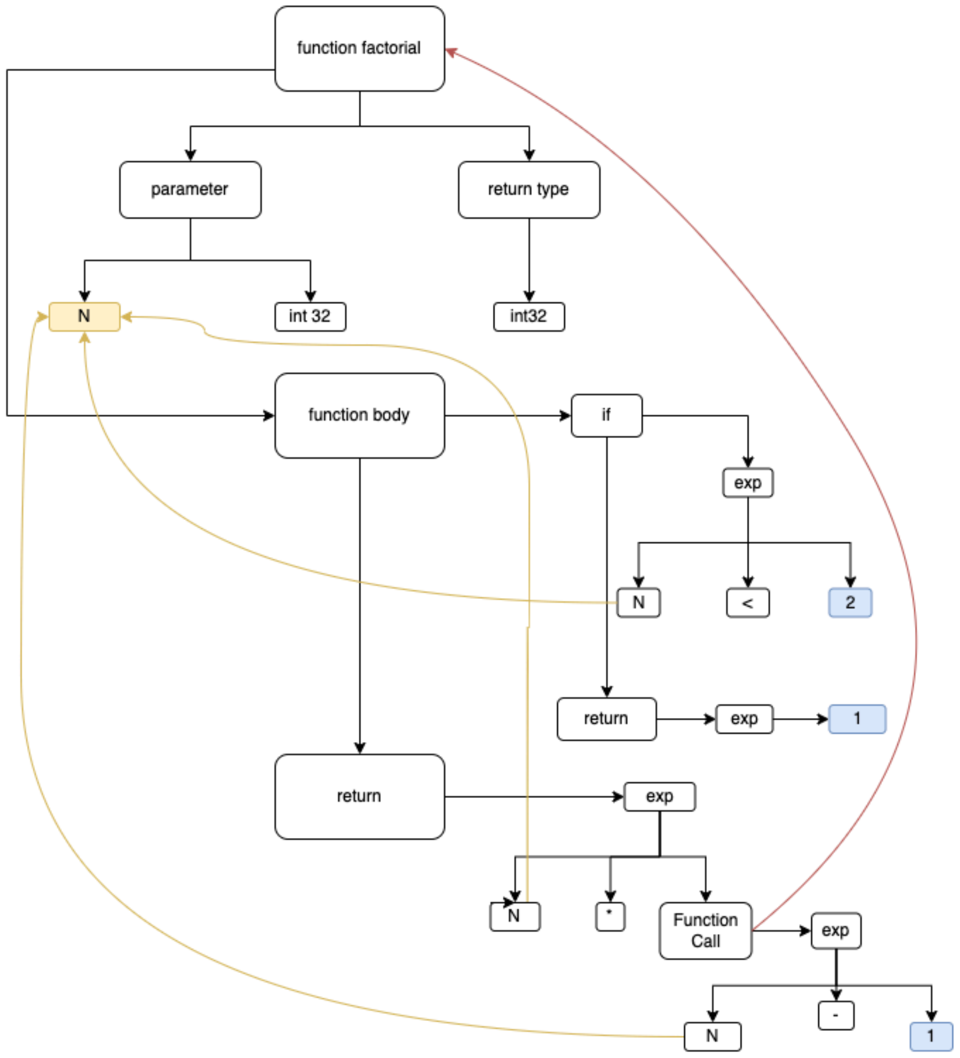

Fig 2. The code in the editor and in the Abstract Syntax Tree format

This diagram shows the structure of the simple recursive function to calculate the factorial of N. As it may be seen, each command and variable has their own nodes, and the same variables refer to the same node across the code (yellow, N). Constants are highlighted in blue. The "exp" node accounts to an expression node which contains one or more other nodes.

5. Project Execution (15%)

At the beginning, we stepped into the unknown and tested various ideas to build something that worked. We learned about how compilers work, explored AST-based trees, and looked into the details of IDEs while working on this project. By the end of the fall semester, we managed to create a basic version of our product that worked well enough to allow for writing, compiling, and executing code.

Originally, our aim for the fall semester was just to get the project up and running. But in the spring semester, we decided to improve the quality of our project. To make it easier to use, we updated the look of the IDE, taking inspiration from the popular VS Code's color scheme to make our project look more up-to-date.

We also made it easier to move through the code. Before, users had to navigate through code nodes, which worked but was not very user-friendly. We enhanced the user experience to allow for smoother navigation, making it more natural for users to move the cursor through the code.

Moreover, we needed to overhaul the code. We reorganized a large part of it to make it faster and easier to maintain. Before, all code nodes were kept in an array, and adding a new node took a long time - a time complexity of $O(N)$, to be precise. Now, by changing to a linked list, adding a new node requires only $O(1)$ time on average. This change also fixed a bug where adding a new node in the array would mess up the references for the nodes that came after it. Now that we use pointers, the references stay correct. Also, we implemented a terminal window where the output of the code is presented to the user.

These big changes made a significant difference both behind the scenes and in what users see and experience in the project.

6. Evaluation (20%)

To conduct a more thorough evaluation of Emen, a programming language designed to be accessible for beginners, we devised a targeted study involving a diverse group of participants. Our subject pool included six students: three from a beginner programming course, CSCI 115, where Python is introduced as their first programming language, and three from non-technical backgrounds, specifically majors in history and world linguistics and languages. This mix was chosen to assess Emen's usability across users with varying degrees of prior programming exposure.

The evaluation began with a brief introductory session where participants were given 15 minutes to familiarize themselves with Emen's basic concepts and interface. Following this, they were tasked with solving entry-level programming problems sourced from acmp.ru, such as “A + B” and other straightforward mathematical challenges. These problems were selected to minimize complexity and focus purely on the interaction with the language and its development environment.

After the test, we conducted a survey to measure the participants' experiences. The survey included questions on the ease of understanding Emen’s syntax, the intuitiveness of the IDE (Integrated Development Environment), and overall satisfaction with programming in Emen.

Results:

- Five of the six participants successfully completed all the given problems within the allotted time.
- Four students highlighted the minimal learning curve required to start programming in Emen. They appreciated not having to memorize or understand complex syntax rules, which often pose a significant barrier to beginners.
- All participants favored the visual design of the IDE, which they found to be clear and helpful in structuring their code visually. This aspect was particularly praised for reducing the intimidation factor associated with traditional text-based coding environments.
- The feedback was generally positive, with students expressing high levels of satisfaction with their ability to engage with programming tasks directly and efficiently.

The study provided strong evidence that Emen successfully addresses key barriers for beginners in programming. Its intuitive design and user-friendly interface make it an excellent choice for educational environments where students are introduced to programming concepts. However, for advancing programming skills beyond beginner level, it might be beneficial to gradually introduce elements of traditional programming to prepare students for more complex tasks. Future iterations of Emen could also benefit from incorporating features that ease this transition, maintaining its beginner-friendly nature while expanding its capability to support more advanced programming needs.

We believe that this evaluation not only confirms Emen's effectiveness as an introductory tool but also provides insights into areas for further enhancement to cater to a broader learning curve.

7. Conclusion and possible future work (5%)

In conclusion, our project, Emen, has successfully tackled the challenges faced by novice programmers by creating a more intuitive and error-resilient programming environment. Through the use of an AST-based programming language and a custom editor and compiler, we reduced the frequency and severity of syntax and semantics errors. The enhancements in the UI coupled with improvements in code navigation and maintenance, have made Emen a user-friendly platform conducive to learning and development.

Emen represents a step forward in programming language development, reflecting a potential shift from traditional text-file based languages to more visual and logical ones. Looking forward, there are several ways for future work to enhance Emen's capabilities.

- **Language Feature Expansion**: Incorporating classes, objects, and inheritance could extend the utility of Emen to more advanced programming.
- **Cross-Platform Compatibility**: Ensuring Emen runs smoothly on various operating systems will broaden its accessibility.

- **Educational Resources and Documentation**: Developing comprehensive tutorials, documentation, and interactive learning modules could position Emen as an educational tool for institutions.
- **Automated Error Correction**: Leveraging machine learning to predict and automatically correct common coding errors could further simplify the coding process.

The progress made thus far on Emen is just the beginning. With continuous improvement and adaptation, Emen has the potential to make a difference in the software development world.

8. References

[1] R. Fleury, Webster, and Lechón, “Dion Systems,” Dion Systems. https://dion.systems
[2] Scratch Team, "Scratch - Imagine, Program, Share," Scratch. https://scratch.mit.edu/