WIP: Memory Visualizer (MV) - A Memory Visualization Tool for Teaching and Understanding Pointers

***Abstract*—This innovative practice WIP paper describes an effective computer-based instruction tool that can be used in Undergraduate Education for teaching the fundamentals of memory management. C++ has been a popular teaching language in the past few decades and continues to do so because of its wide use. One of the main features that provides it with exceptional strength is its direct access to memory through pointers. Unfortunately, students who are familiarizing themselves with programming basics often struggle to use pointers effectively, as they find it challenging to create a mind-map of values and addresses while programming. We have developed a tool called Memory Visualizer (MV) that visualizes the data and memory allocation across stack and heap in real-time while writing C++ code. This tool parses the written code and visually creates relationships between variables and pointers so that students can see how their code affects the memory. We tested this tool on new students, those who had already worked with pointers, and even teachers who have been teaching for a while and took a survey about the usefulness of the tool. The general response from the students has been positive as they have found it useful to visualize memory. Instructors have praised the tool for real-time memory representation, which helps in teaching memory-related concepts.**

***Index Terms*—memory, pointers, programming, learning, teaching**

I. Introduction

Lately, there has been a push towards using memory safe languages, most notable recommendation coming from the United States' National Security Agency (NSA). Their Cybersecurity Information Sheet [1] suggested using memory safe programming languages like, Rust while doing away with C and C++, which rely heavily on the programmer to perform the needed checks on memory references. While this may be the right move to make to ensure security in future software, we still have two important challenges to contend with. Firstly, a lot of legacy software is written in C and C++ and will continue to be used for a very long time. Secondly, understanding the memory issues is critical in resolving them, and we cannot deny the importance of being aware of these problems as a programmer.

As a result, C and C++ are continuously being taught in a lot of institutes, and they are not going to be replaced by other languages anytime soon. In C and C++, it has been found that the students struggle while understanding the concepts of pointers and memory management [2]. We realized that one of the issues in understanding has been the inability of students to make a mind-map of variables and pointers and thus end up making mistakes. Hence, we developed a tool, Memory Visualizer (MV), which demonstrates memory allocation to stack and heap visually. With this tool, we can show the relationships between the pointers and other variables in real-time as the user writes the code. This makes it a very powerful tool for teaching and learning concepts like memory leaks, dereferencing, dangling pointers, etc. Anyone can write any code as per the supported parser and the tool will highlight how the memory is assigned and the various relationships between pointers and variables.

It is also a very effective tool during teaching because it uses arrows to make connections between variables and pointers, display the allocated memory, and show the leaks created when writing problematic code. All of this is done in real-time and with proper visualizations to hold the interest of the users. We believe that such a tool will be extremely beneficial for teachers and learners and will see wide use in institutes where C++ is a language of instruction.

II. Related Work

Due to the importance of memory concepts in understanding the basics of programming, several researchers have explored how students view the storage of values in memory. Postner and Turns [3] conducted surveys and interviews to explore students' conceptions of variables, focusing on their understanding of memory allocation, variable declaration and initialization, data storage, and pointer operations. Their findings revealed, for instance, that 24% of survey participants were unaware that declaring a variable allocates memory to it. Their questionnaire also indicated that a significant number of students could not differentiate between the values stored in pointers and normal variables.

We have also seen that students who have had some programming experience in other languages also struggled with understanding pointers. While teaching C, Craig and Petersen [2] spent two lecture hours introducing pointers by explaining the theory and live-coding a worked example. Then, the students were given programming problems to evaluate mastery of the pointer concepts. The students found pointers challenging and also had trouble understanding how arrays and pointers are related.

Most of the work has been towards conducting lectures, interviews, and various assessments to gauge how well the students perform when the effort has been made to specifically target the problematic concepts. However, the work towards making a robust teaching tool has been a bit limited. In our

quest to search for similar software, we normally found tools that showed the state of memory rather than how the code itself influences memory allocation. For example, the aptly named "Memory Visualizer" [4] is a program implemented in C++ using the OpenGL library. It allows you to see the contents of every address in the process address space. It also color-maps every byte in memory so one can visually get an idea about the most common byte values. It however, does not show the effects of code in memory itself. Another notable tool was CppMem [5] which displays some visualization, but we found it to be too complicated for teaching purposes. We also found SeePlusPlus [6] but its online link was not working and its code has not been updated for the past 5 years.

We conclude our discussion by highlighting the Online C++ Compiler at Python Tutor [7] as a particularly valuable tool for visually tracing the step-by-step compilation of C++ code. This tool enables users to debug their C++ code visually, one step at a time. However, its lack of real-time feedback makes it less suitable for teaching basic programming concepts. Its true value lies in a laboratory setting where students can execute their code and observe the results directly.

III. Problem statement

Students often struggle to develop a clear mental model of how their programs interact with memory, which makes concepts involving pointers and memory management particularly challenging [2]. Boustedt et al [8] categorized pointers as one of the threshold concepts in computer science. A threshold concept is a core, transformative idea in computer science whose understanding is important; otherwise, it ends up blocking students' progress. Iain and Glenn [9] analyzed the difficulties in learning and teaching Programming by gathering views of students and tutors. They found that pointers as a 'concept' are usually easy to explain and get the basic idea of, but it is their actual implementation and subsequent behavior in the running program that causes problems for most students. The lack of effective visualization tools further compounds this difficulty, especially as students progress to advanced topics in computer science. To address this issue, we propose a sophisticated learning tool that generates real-time visualizations of how programs manipulate memory. By bridging the gap between code and execution, this tool aims to provide students with a clearer understanding of memory interactions, and help them build stronger mental models for memory management. We had to make some design considerations when developing this tool to keep the focus on learning and visualization than developing a complete parser to handle all types of code. This is further explained in the next section.

IV. Methodology

This section will provide a detailed account of the processes and techniques employed in the development of the Memory Visualizer (MV) tool and also outline the strategies implemented for data collection and evaluation to assess its effectiveness.

A. Design and Architecture

MV is built using the Tauri framework [10], which allows us to create lightweight desktop applications by combining Rust for the back-end with web technologies such as React and NextJS for the front-end. This integration allows us to leverage Rust's performance and memory safety features while utilizing modern web development tools for a dynamic user interface. By using Tauri, we streamline the development process, enabling us to concentrate on enhancing the application's core functionalities and user experience. We have developed a desktop application and have also made it available online, by using Web Assembly for easy access [11].

Rust has recently become very popular due to its memory safety related features and is one of the preferred languages as per NSA's Cybersecurity Information Sheet [1] and thus aligns perfectly with our development goals for performance, safety and overall efficiency.

At the core of MV's architecture is a custom-built system, comprising a lexer, a parser, and an analyzer; each designed to meet the application's unique requirements. The process begins when the user inputs code into the application's editor. The entered code is first tokenized by the lexer, which translates the raw input into meaningful tokens. The tokens are then passed to the parser, which interprets them, constructing logical constructs such as pointer declarations, variable assignments, and basic programming statements.

Following parsing, the analyzer validates the logical constructs, ensuring correctness by detecting issues such as references to non-existent pointers or type mismatches. After error checking, and validation, the analyzer organizes the data into two distinct arrays representing the stack and heap memory allocations. These memory representations are then sent back to the front-end for real-time visualization, offering users immediate insight into how their code interacts with memory. Designing this way provides us with the following benefits:

- We can focus directly on variable and pointer declarations, and can ignore the extra code one requires to run a program in C++.
- We can see the visualizations in real-time as we type, thus giving us instantaneous feedback.
- We can mimic memory behavior of Stack and Heap.
- We do not need to compile code, and students can follow the instructor in real-time as well.
- We can show bugs like, memory leaks and dangling pointers visually.

B. User Interface

The user interface (UI) is distributed into two sections, which are split in the middle; the editor section on the left-hand side and the visualization section on the right-hand side. The size of these sections can be changed as per user requirements. The user writes code in the editor, and the appropriate visualization is displayed in the visualization section. In this way, we can instantaneously compare the written code with the memory allocation in stack and heap without the need

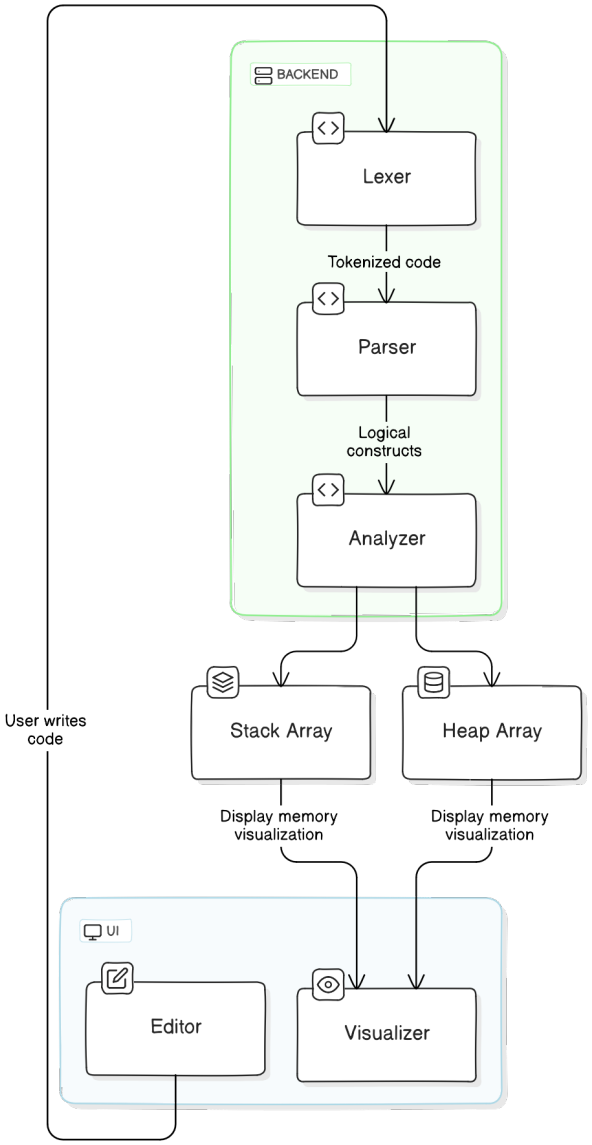

Fig. 1. Relationship between the components of the software

for compiling the code. We have created a tutorial video that demonstrates how to use this tool, which is present on the following link [12].

C. Concepts Taught

When we started the development of MV, our main goal was to have a tool that can be used in the classroom in real-time to teach memory related issues. Our goal was not to make a complete parser or a tool that handles every type of code. Our focus was to make it teaching oriented, which can be quickly brought up in class to teach the different concepts and for reviewing the written code. Therefore, it does away with the normal C++ conventions you have to go through to run the code like defining the main function or including header files, etc. Hence, we chose to focus on a couple of

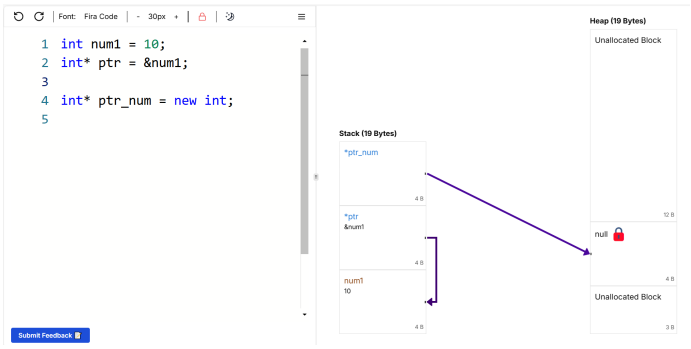

Fig. 2. User Interface of Memory Visualizer

concepts initially which we want the students to learn before they advance towards more complicated programming tasks. Support for advance programming concepts will be added in due time, but for now it supports the following concepts:

- Variables.
- Pointers.
- De-referencing pointers.
- Understanding of Stack and Heap.
- Memory allocation and de-allocation.
- Memory leaks.
- Dangling pointers.
- Null values.

These appear to be very basic concepts, but we believe that the actual problem starts from misunderstanding them. Students should be able to visualize how memory is allocated for different variable types and then move onto pointers as variables. One of the approaches for teaching can be that once the concept of pointers has been explained, the next step is to demonstrate how de-referencing is achieved through them. MV does a good job in doing this as we can see the relationship between the pointers and variables through arrows drawn in the visualization section and can also see the changes made in the memory through code. This is followed by introduction to dynamic memory allocation and then de-referencing that memory location. We then introduce how we can free memory location and introduce concepts of memory leaks and dangling pointers by making mistakes in our code intentionally just to show that there can be bugs in your code even if it appears to work correctly. This step-wise teaching progression is important in explaining the concept of pointers and memory management and is made very easy while using MV. The clear understanding of these concepts is also necessary when explaining the workings of arrays or argument passing in functions as their basis lies in memory declaration and manipulation.

D. Data Collection & Results

Our study was conducted in various academic institutions where we invited students with computer science and other backgrounds along with teachers to test the tool and provide feedback through a survey. Some industrial specialists were

also contacted to try it out and provide feedback. They received a consent form prior to filling out the survey to confirm the use of their submission for this study. Due to time constraints, we were unable to test this tool in an actual class environment for first-year students, as they had not yet encountered this topic in their programming fundamentals course.

This tool was tested by students who were sophomores, juniors, and seniors. In the coming semesters, we hope to use this tool in actual introduction to programming course and that will give us the real data about the utility of this tool when teaching to students who had not studied pointers before.

We were able to take a survey from 33 participants where 88% were from Computer Science or related background. Out of these, only 36.4% were undergraduate students while the rest had graduated and were either working in the industry or teaching at an institute. Out of these, 57.6% had previously worked in C++ while 30.3% had C experience.

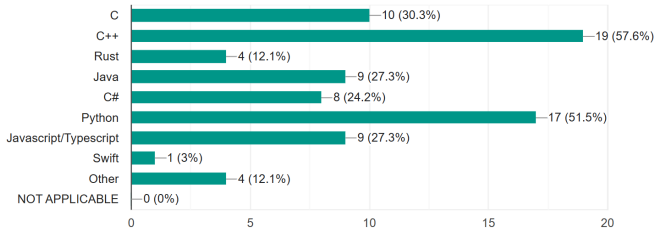

Fig. 3. Prior programming experience

Most of the participants had programming experience with 21.2% claiming that they were not confident in their programming skills. When asked how intuitive they found MV, 81.8% participants gave 4 or higher score on a scale of 1 to 5 where 5 being best rating. 87.9% participants gave a score of 4 or more when asked if the animations and visualizations were helpful in understanding memory management. When asked what concepts they learned after using MV, 40.6% selected dangling pointers while 37.5% chose heap and pointer each with these choices not being mutually exclusive.

Fig. 4. Utility of MV in understanding concepts

An observation is that most of the participants were not undergraduate students, but rather those who had already graduated. As a result, the feedback comments were mostly about features they wanted to be implemented, or the bugs that they encountered. Surprisingly, we did not get any suggestions about the User Interface which suggests that it was widely acceptable.

All participants however, agreed that MV should be used in class for teaching purposes, while 97% said that they would recommend MV if someone is having trouble with C++. The single person who chose not to recommend MV did not give a reason why. These results suggest that our experiment with MV was quite successful, but we agree that 33 participants is a small number to determine the success of the tool. This tool was not tested in a true learning environment and it still remains to be seen how effective it was for the teacher to convey ideas and for the students to accept them. We will get a much better picture once this tool is taught in a class environment and tested over multiple sessions.

V. Future Work

The road for MV is a bit long as we have to provide support for a lot of concepts that have been asked by various instructors who tried the tool. Some of the most in demand requests have been:

- Support for arrays
- References
- Double and Triple pointers
- Functions and arguments
- Pointers to objects

Most of these requests were already part of our future milestones and will be implemented, but we are currently struggling with another issue of finding the sweet spot where the tool remains useful and does not become an unintelligible network of arrows. As explained before, this tool is specifically being made to be a teaching and learning tool and satisfying all the language requirements is not the aim here. For example, currently we do not see a use-case for providing support for conditionals and loops. Their utility only seems relevant if we want to display how poorly written code may create bugs and memory leaks. We are also trying for greater adoption of this tool for the introduction to programming course while teaching, so by the time we incorporate new features, the tool would have been more widely tested thus providing us with more targeted data.

Another requested feature is AI integration so that the users are told about the mistakes they make and guide them towards writing better code. We see the potential of this functionality, and it will be added once we achieve other critical milestones regarding teaching the basic programming.

Similarly, using generative AI to create problems that specifically talk about bugs in the system can be useful in this tool, as it will build the debugging skills of the students and they may gain skills to spot issues with problematic code.

Finally, we plan to keep the code for this tool to be open source once it is in a presentable state to encourage development of tools on top of its basic architecture. We believe that this will help in future proofing the software and might end up creating support for many more features requested by the users and developers.

VI. Conclusion

The aim of MV was to have a teaching tool that visualizes changes in memory in real time. Currently available tools help with this visualization and can also show the step-by-step debugging of the code, but the user-written code needs to be parsed before any visual feedback can be viewed. This 2-step process makes it difficult to show the changes in code quickly and introduces undesirable downtime while teaching. We also realize that while teaching programming, a perfect reproduction of the coding constructs is not necessary, thus they are ignored to keep focus on the relevant code.

The feature that has been appreciated the most is the real-time visualization of the code in memory as we make changes without compiling. Finally, putting MV online through Web Assembly has greatly improved the usability of the tool, as it is now platform independent and anyone can use it safely without worrying about installation details. Hence, a successful teaching tool should be easy to access and must prioritize learning over unnecessary complications that come with advance programming. It should also offer enough visual fidelity to keep the user's focus and provide the necessary optimization to keep the user interface clean and uncluttered. Having a tool that provides immediate feedback is extremely important when teaching coding in early stages of learning, and it can be of great benefit in the classroom for teaching memory management in C++.

References

[1] S. Dept. of Defense [Online]. CSI Software Memory Safety. Nov 10, 2022. [Accessed Sep 3, 2024]. Available: https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

[2] Michelle Craig and Andrew Petersen. 2016. Student difficulties with pointer concepts in C. In Proceedings of the Australasian Computer Science Week Multiconference (ACSW '16). Association for Computing Machinery, New York, NY, USA, Article 8, 1–10. https://doi.org/10.1145/2843043.2843348

[3] Postner, Lori & Turns, Jennifer. (2002). Using facet-based assessment to understand introductory programming students knowledge. Proceedings - Frontiers in Education Conference. 1. T4G-7. 10.1109/FIE.2002.1158012.

[4] Ali BaderEddin. Memory Visualizer, a memory visualization tool. https://www.codeproject.com/Articles/71644/Memory-Visualizer. Accessed 11th June, 2025.

[5] CppMem: Interactive C/C++ memory model. [Accessed 11th June, 2025]. Available: http://svr-pes20-cppmem.cl.cam.ac.uk/cppmem/.

[6] SeePlusPlus. [Accessed 11th June, 2025] Available: https://github.com/knazir/SeePlusPlus.

[7] Online C++ compiler, visual debugger, and AI tutor. [Accessed 11th June, 2025]. Available: https://pythontutor.com/cpp.html#mode=edit.

[8] Boustedt, Jonas & Eckerdal, Anna & Mccartney, Robert & Moström, Jan & Ratcliffe, Mark & Sanders, Kate & Zander, Carol. (2007). Threshold concepts in computer science: Do they exist and are they useful?. ACM SIGCSE Bulletin. 39. 504-508. 10.1145/1227504.1227482

[9] Milne, I., and Rowe, G. 2002. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. Education and Information Technologies, 7 (March 2002), 55-66.

[10] Tauri Contributors. Tauri Framework. [Online]. [Accessed 11th June, 2025] Available: https://tauri.app/

[11] Web based Memory Visualizer. [Online]. [Accessed 11th June, 2025] Available: https://humblepenguinn.github.io/memv/

[12] Tutorial Video. [Online]. [Accessed 11th June, 2025] Available: https://youtu.be/lzccceC31s0