**CS 499 Module Four Milestone Three Narrative**

Andrew Emilio DiStefano

SNHU

Professor Neil Kalinowski

May 31, 2025

1. **Briefly describe the artifact. What is it? When was it created?**

The MemoryBot artifact is a program meant to help users to remember anything from measurements to scientific formulas to the names of newborn nieces and nephews. I created this in 2023 during my completion of the *Sophia Learning Introduction to Python* course which I took for three credits toward my degree. The system was simple but effective in its functionality of approximating simple conversation considering the low level of complexity. The program was originally written in Python, but I have rewritten it in C++ in order to reach optimal efficiency.

1. **Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?**

I chose this artifact for my *Algorithms and Data Structures* enhancement due to the fact that the original program had been developed with a data structure which is vastly inferior to those which are commonly used for similar purposes in the professional world. Enhancing this artifact in the ways which I will discuss in this document will be an excellent way to showcase my ability to implement complex algorithms and data structures. The original program used an iterative algorithm over a vector data structure which yields barely satisfactory results with an unimpressive time complexity of *O(n)*. Since this application involves comparing strings and substrings in order to find the appropriate question-answer (key-value) pair for a given user input string (question), using a data structure which complicates the process of substring comparison is not ideal. I had originally planned on enhancing this artifact by implementing a binary search tree data structure. I have recently come to the conclusion that while this would dramatically improve time complexity, the comparison of string and substring values over this type of data structure would greatly complicate the search process. Search trees are excellent for situations where the key for a key-value pair will be entered identically, but not necessarily for situations where the most appropriate key is selected from user input which may differ from any stored key.

I solved this complex issue by deciding to implement an inverted index data structure rather than a search tree. The inverted index is used by mainstream search engines such as *Google* (Slawski, 2021) to produce the best results based on the words (or tokens) entered by a user in a search query. By comparing each word in a user input string to each word in each question key of a question-answer (key-value) pair, the system is now able to effectively approximate the answer that a user is looking for based on their input question. Due to the fact that only question-answer (key-value) pairs which share tokens (or words) in common with the input string will be compared to determine the appropriate output, the search algorithm implemented over this data structure achieves a much better time complexity than *O(n)* under normal circumstances. This is true unless the input string shares at least one token in common with every single question-answer pair stored in our inverted index, which becomes increasingly unlikely as new and unique question-answer pairs are added to the data structure. It goes without saying that these changes to our data structure and the CRUD algorithms used for manipulating our data store greatly improves the effectiveness and the efficiency of our program in terms of data lookup.

1. **Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?**

The changes made this week have actually met more of the course outcomes than originally intended in Module One. My use and citation of open-source libraries shows my commitment to building collaborative environments that enable diverse audiences to support development. My comprehensive documentation and easily interpreted ReadMe instructions show my ability to design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts. My decision to use an inverted index rather than a search tree as my data structure, due to the increased functionality and despite the slightly less efficient time complexity, shows my proficiency in designing and evaluating computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices. My implementation of data structures and search algorithms similar to those used by prominent search engines like Google (Slawski, 2021) demonstrates my ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals. My use of a *JSON* file rather than a *TXT* file as my long-term data storage option also does a great deal to assure that I have met this course outcome.

My security mindset will become much more apparent when I implement the required security functions into the MemoryBot system. That being said, my commitment to the goal of system security is clear from the *FIXME* comments added throughout the program to remind me of specific *C++* security functionalities which must be added to the finished product.

1. **Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?**

This week has been a great learning experience for me, and I am quite satisfied with the choices that I have made over the past few days in the interest of enhancing the effectiveness and efficiency of my data structure for this project. When I was about halfway through developing a *BinarySearchTree* class, I realized that any comparison of substrings within input strings and questions (keys) belonging to question-answer (key-value) pairs would greatly decrease the efficiency of the search algorithm. I realized that while a balanced search tree offers incredible time complexity when the exact value of a key is known, this data structure simply does not make sense for this application of comparing user input strings to saved strings word-for-word. After some research into different potential solutions, I decided to use an inverted index data structure to enable the querying of the most relevant data (answer) based on comparison of the corresponding question (key) with the question string input by the user. This will result in a much more effective and precise program which will be able to infer the most appropriate answers to user questions based on data stored in the inverted index.

The fact that I had never heard of (let alone implemented) this type of data structure before researching for this project certainly posed significant challenges. I spent almost a full work day researching and comprehending the actual nature of this type of data structure, and of the algorithms which can be used to perform CRUD operations on the data stored within. I decided that querying the data structure, adding data, and deleting data would be sufficient here as the implementation of an algorithm for editing and updating existing question-answer (key-value) pairs would not greatly enhance the functionality of the system. It is also worth mentioning that the implementation of update and edit functionality in a conversational interface would be incredibly clunky and unnatural.

While planning for my implementation of the inverted index into the MemoryBot project, I also realized that the use of a simple *TXT* file would not suffice for this application. I decided to use a *JSON* file to store question-answer (key-value) pairs for ease of interpretation by the program, and for potential integration with databases in the future. I used an [open-source *JSON* parsing library for *C++*](https://github.com/nlohmann/json) (Lohmann, 2025), which I have cited below and in the project itself via a file called *citations.md* in the */external/* folder of my project. The library is open-source under the MIT License, and “permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files” (Neumann, 2025). The library works seamlessly with the inverted index data structure which I have put together for the MemoryBot project. *JSON*-to-inverted-index and inverted-index-to-*JSON* functions have been placed before and after the conversation loop respectively. This of course ensures that the *JSON* file is updated upon exiting the conversation loop, and that the inverted index is loaded with the key-value pairs from the updated *JSON* file before entering the conversation loop. Measures have been taken to ensure the efficiency of these functions, such as the ignoring of duplicate key-value pairs when the *JSON* file is updated from the inverted index after a conversation with the user.

The system is up and running, but I still have to iron out a few logical bugs which print duplicate output in certain situations. This will be taken care of before the submission of my project.

**Sources:**

Lohmann, J. (2025). *JSON*. GitHub. <https://github.com/nlohmann/json>

Slawski, B. (2021, July 9). *Google’s Inverted Index of the Web*. Seo by the Sea.

<https://www.seobythesea.com/2021/07/inverted-index-of-the-web/>