Enhancement Two: Algorithms and Data Structures

Christopher Sharp

Southern New Hampshire University

Dr. Greg Stefanelli

May 26, 2025

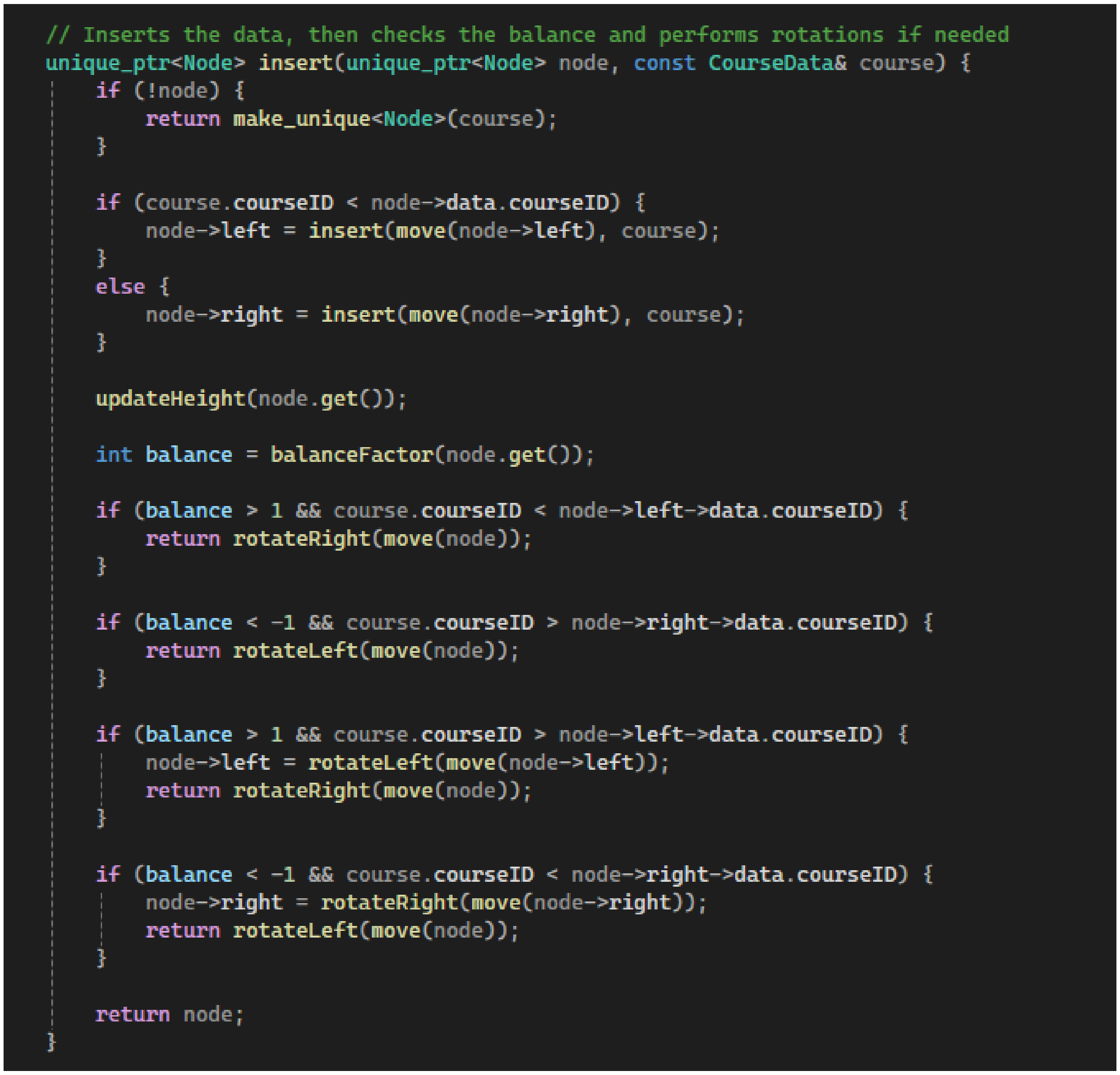
Enhancement Two: Algorithms and Data Structures

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

The artifact selected for Category Two, Algorithms and Data Structures, is coursework that was completed in CS-300: Data Structures and Algorithms, Analysis and Design in the Spring of 2023. This software was written in the C++ programming language and was an introduction to data structures and algorithms, while introducing the concepts of time and space complexity. For this specific artifact, we were tasked with creating an application that assists academic advisors at a fictional university, ABC University, with displaying and searching the course catalog. This application is a console-based application with a menu system for user interaction. The application imports data from a CSV file into a data structure. It then allows the user to list all of the courses in alpha-numeric order or search for a specific course, returning its data, including name and any prerequisites.

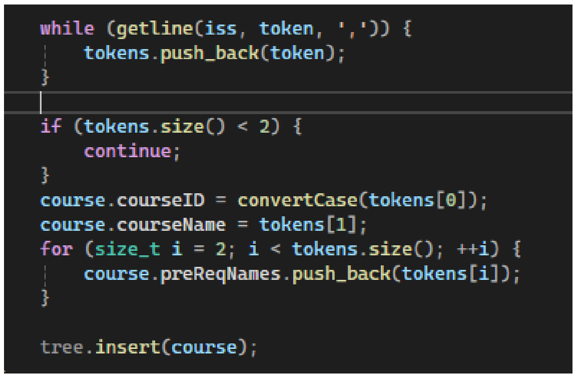
Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structures? How was the artifact improved?

The artifact was chosen because it offered the opportunity to demonstrate my improved understanding of algorithms and data structures. The original artifact utilized a vector to store the data, with each element in the vector stored as a struct, and it used quicksort to sort the data. This meant that the average sorting time was O(n log n) and the linear search time of O(n). While this approach is functional, it is neither the most efficient nor scalable for larger operations. So, for this assignment, I refactored the code to use a self-balancing AVL tree (Figure 1). By implementing this change, we allowed for quicker insertions and lookups, O(log n). This enhancement starts to shine as the dataset grows larger or is in a constant state of change. Also, other changes have been made to the code base to modernize it, and some defensive programming has been implemented. For instance, the introduction of unique pointers allows for the elimination of manual memory management, thus allowing the program to be memory safe. Another change that was implemented was the use of null checks, input sanitization (Figure 2), and structured exception handling (Figure 3).

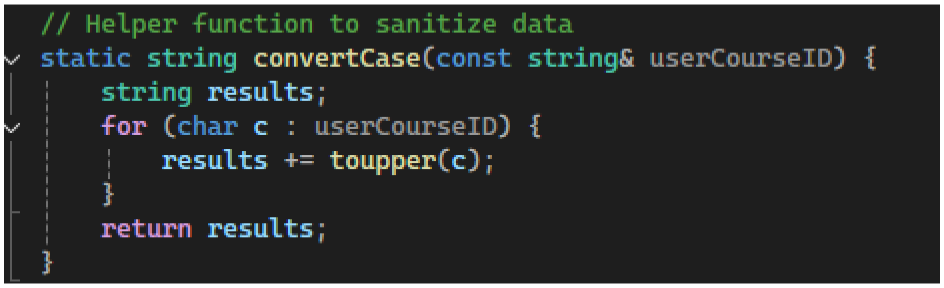


```cpp
// Inserts the data, then checks the balance and performs rotations if needed
unique_ptr<Node> insert(unique_ptr<Node> node, const CourseData& course) {
    if (!node) {
        return make_unique<Node>(course);
    }

    if (course.courseID < node->data.courseID) {
        node->left = insert(move(node->left), course);
    }
    else {
        node->right = insert(move(node->right), course);
    }

    updateHeight(node.get());

    int balance = balanceFactor(node.get());

    if (balance > 1 && course.courseID < node->left->data.courseID) {
        return rotateRight(move(node));
    }

    if (balance < -1 && course.courseID > node->right->data.courseID) {
        return rotateLeft(move(node));
    }

    if (balance > 1 && course.courseID > node->left->data.courseID) {
        node->left = rotateLeft(move(node->left));
        return rotateRight(move(node));
    }

    if (balance < -1 && course.courseID < node->right->data.courseID) {
        node->right = rotateRight(move(node->right));
        return rotateLeft(move(node));
    }

    return node;
}
```

Figure 1 – Insertion into AVL

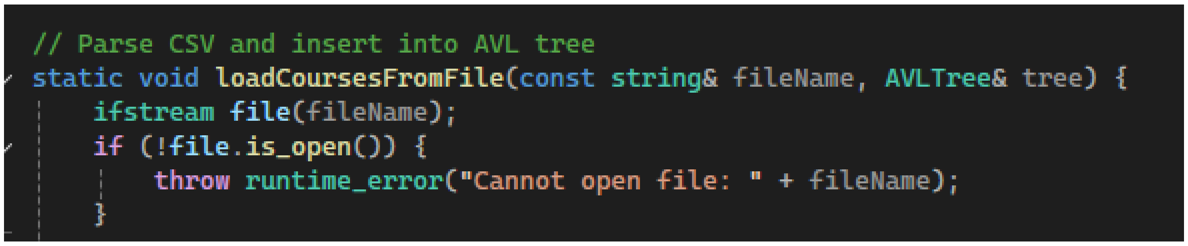


```
while (getline(iss, token, ',')) {
    tokens.push_back(token);
}

if (tokens.size() < 2) {
    continue;
}
course.courseID = convertCase(tokens[0]);
course.courseName = tokens[1];
for (size_t i = 2; i < tokens.size(); ++i) {
    course.preReqNames.push_back(tokens[i]);
}

tree.insert(course);
```



```
// Helper function to sanitize data
static string convertCase(const string& userCourseID) {
    string results;
    for (char c : userCourseID) {
        results += toupper(c);
    }
    return results;
}
```

Figure 2 - File Sanitization



```
// Parse CSV and insert into AVL tree
static void loadCoursesFromFile(const string& fileName, AVLTree& tree) {
    ifstream file(fileName);
    if (!file.is_open()) {
        throw runtime_error("Cannot open file: " + fileName);
    }
```

Figure 3 - Exception Handling

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?

Yes, I have met all of the course outcomes that I had planned to meet in this enhancement, as well as new outcomes due to my further understanding of them.

[CS499-01] Employ strategies for building collaborative environments that enable diverse audiences to support organizational decision making in the field of computer science.

By including the readme file in the repository, I employ strategies that allow for clear instructions on how to build and run the application. Also included are screenshots of the application and basic information. Finally, with GitHub Actions being implemented for building and static testing the application, this ensures that everyone on the team and any external contributors use the same process to reduce any complications that may occur.

[CS499-02] Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts.

By providing code comments located throughout the code base, as well as a brief description of the code and its functionality at the top of the code, I am delivering on this course outcome. Also, by including the accompanying readme that highlights different sections, including features, performance, installation instructions, and usage instructions, this provides technically sound documentation that helps both developers and users.

[CS499-03] Design and evaluate 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.

By evaluating the performance of the application in its former form, I was able to take the concepts of time and space complexity and improve upon the program. For instance, by implementing the self-balancing AVL tree, I was able to improve from quicksort O(n log n) + search O(n) versus O(log n) for the same functionality. Also, by implementing the AVL tree, with its rotations, balance logic, and in-order traversal, I demonstrated core algorithmic principles in action. Also, during the entire development process for the enhancement, GitHub Actions were implemented to build the application, MSBuild, and CodeQL to scan for any vulnerabilities.

[CS499-04] Demonstrate an 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

By implementing the use of modern C++ techniques like smart pointers for automated memory management show my skill in using innovative techniques. Another example of this is the use of structured exception handling and null-pointer guards. Also, with the use of a self-balancing AVL tree, this points to forward thinking in preparation for the growing and dynamic amount of data that could be used throughout the program. Throughout the development process, continuous integration/continuous development tools were deployed through GitHub Actions. These included MSBuild to perform automated build actions and CodeQL to run automated static analysis.

[CS499-05] Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources.

The implementation of input validation and checks during the CSV parsing demonstrates a mindset to anticipate exploits within the software. Other examples that demonstrate that a security mindset was present include the use of structured exception handling, the null-pointer guards, and the use of smart pointers for memory management. During the development of the application, CodeQL scans were implemented through GitHub Actions to perform scans on the code to check for many different vulnerabilities.

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?

The process of enhancing and modifying this artifact was a little tougher than I had envisioned, due to a lack of knowledge with C++ and the time since I had written the original piece of software. The implementation of smart pointers was an eye-opener, as I had no prior knowledge of them until I did some research on memory management, including implementing manual memory management. Also, I possessed the knowledge from the class about binary search trees, but knew that in some cases they could become unbalanced, which significantly impacts performance. This is where I learned about AVL trees and Red-Black trees through reading information on the web that can solve this issue. There were some issues that I ran into, especially with the implementation of the smart pointers. This is because of how they take exclusive ownership of the data. Once I figured this out and how to manipulate the data, everything started to run smoothly. Also, I had to look up a bunch of information on the AVL tree so that I could learn how it functions and how I could adapt it to work in this program.