BINARY SEARCH TREE :student data analysis

# 

### 

# Abstract

This C++ program presents a robust solution for student data analysis utilizing a Binary Search Tree (BST) data structure. Student information, including names and grades, is input interactively by the user, facilitating the construction of a BST where each node represents a student. The importance of efficient student data management and analysis in the educational domain cannot be overstated, with the need for insights into academic performance driving the development of such tools.

In the realm of educational data analysis, the ability to identify trends, patterns, and outliers is crucial for educators and administrators to make informed decisions and interventions to enhance student outcomes. However, there exists a gap in accessible and user-friendly tools that provide comprehensive student data analysis capabilities.

This program seeks to address this gap by providing functionalities to identify the student with the highest grade, calculate the average grade of all students, and display students exceeding a specified grade threshold. By employing a structured approach and leveraging the BST data structure, the program delivers efficient student data analysis tools that cater to the needs of educators and stakeholders.

The key message of this program lies in its ability to streamline student data analysis processes, enabling educators to gain valuable insights into student performance and inform data-driven decision-making. Through its intuitive interface and robust analytical capabilities, this program contributes to the advancement of educational data analysis and supports efforts to improve student outcomes.

# Introduction

In the realm of education, data-driven decision-making plays a pivotal role in enhancing student outcomes and improving teaching methodologies. This C++ program introduces a robust solution for student data analysis, leveraging the power of Binary Search Trees (BSTs) to efficiently manage and analyze student information. According to research by Zhang and Wu (2019), BSTs offer an effective means of organizing and querying data, making them ideal for applications requiring quick search and retrieval operations. By representing each student as a node within the BST, this program provides educators and administrators with a structured approach to analyze student performance and identify trends.

The methodology employed in this work involves the utilization of a Binary Search Tree data structure to organize and manage student data. The program prompts users to input student information, including names and grades, which are then inserted into the BST using an insertion algorithm. To analyze the data, functions such as finding the student with the highest grade, calculating the average grade, and displaying students above a specified threshold are implemented, leveraging the properties of BSTs for efficient traversal and retrieval.

Following the introduction and methodology, the program proceeds with interactive sessions where users input student data. Once the data is entered, the program employs various functionalities to analyze the dataset, including identifying the student with the highest grade, calculating the average grade of all students, and displaying students exceeding a specified grade threshold. Through these sessions, the program aims to provide educators and stakeholders with actionable insights derived from student data, facilitating informed decision-making and interventions to support student success.

# Literature survey

In the realm of educational data analysis, various studies have explored the utilization of data structures and algorithms to manage and analyze student information effectively. Binary Search Trees (BSTs) have emerged as a promising data structure for organizing and querying student data due to their efficient search and retrieval operations.

Efficient Data Organization: Research by Smith and Johnson (2018) highlights the importance of efficient data organization in educational settings. They emphasize that well-structured data facilitates quick access to information, enabling educators to make informed decisions regarding student performance and intervention strategies. BSTs, with their ability to maintain data in sorted order, offer an efficient means of organizing student data.

The research paper titled "Student Performance on the BDSI for Basic Data Structures" investigates factors influencing student achievement in a foundational data structures course. Authored by Kevin C. Webb, Daniel Zingaro, Soohyun Nam Liao, Cynthia Taylor, Cynthia Lee, Michael Clancy, and Leo Porter, the study analyzes the impact of prior programming experience, demographic factors, and instructional methods on student outcomes. It explores whether students' backgrounds influence their success in mastering data structures concepts. Additionally, the paper delves into the effectiveness of various teaching approaches, from traditional lectures to interactive or project-based learning methods. The findings provide insights into how educators can improve teaching practices and promote inclusivity in computer science education. Overall, the research contributes to understanding how to better support students in learning fundamental data structures principles.

Construction of estimated level based balanced binary search tree by R. Chinnaiyan; Abhishek Kumar 2017: There are many storage structure available to store data in memory of many forms. These structures can be array, class, linked list with its various forms, Tree, Binary Tree, Binary Search Tree (BST), etc. These can be differentiated in two major forms. First one uses continuous memory allocation and the second one can occupy any free memory block by pointed by the other memory locations. An array occupies continuous memory space for storage purpose and the size should also be known before allocating the space. Perhaps we can use dynamic memory allocation methods for arrays but a Linked List provides better options. There is a disadvantage in Linked List, it does not allow to perform binary search operation on it. The Binary Search Tree is more efficient than the other mentioned data structures. BST provides the two way traversal direction but sometimes the structure of the BST can become unbalanced due to unprocessed ordering of inserted data. In this presented paper, the BST is considered as unbalanced if the number of levels is more than the levels which is required to hold the nodes. The unbalanced BST can lead to a straight tree structure with only one intermediate node at each and every level in the worst case scenario. The structure of BST depends on the insertion order of key elements. By changing the insertion order, BST can be made balanced. The proposed Estimated Level Based Balanced BST provides a solution for finding an insertion order of key elements which will not lead to unbalanced Balanced BST.

Binary search tree traversal for Arrick Robot virtual assembly training module:2011:Nor Farhana Othman; Haslina Arshad : Binary tree traversal is defined as a process of visiting each node in a specified order. There are consist of two ways to visit a tree which are recursively and non-recursively. Our literature survey showed that most of the references only focus on how to implement the fastest and simplest recursive and non-recursive algorithms. We investigate recursive algorithms for inorder, preorder, and postorder traversals to get the most accurate assembly sequence. Arrick Robot will be used as the model for this research. To assemble a robot, sequence of the parts assembled must be right. The recursive postorder traversal algorithm applied for the virtual robot assembly was found to be more suitable than inorder and preorder traversal. To get a faster insertion and deletion, the assembly sequence have to be in correct order. During assembly, the parts assembled can be visualized and animated for students to practice and master the assembly process before they can actually assemble the real Arrick Robot. Errors will be prompted when the robot is not assembled correctly.

A new non-recursive algorithm for binary search tree traversal:2003 A. Al-Rawi; A. Lansari; F. Bouslama : Binary tree traversal refers to the process of visiting each node in a specified order. There are two ways to visit a tree: recursively and non-recursively. Most references introduce tree traversal using recursion only. Our literature survey indicated that most references only show the implementations of the recursive algorithms, and only few references address the issue of nonrecursive algorithms. In this paper, we investigate and compare recursive and non-recursive algorithms for in-order, preorder, and post-order traversals. The in-order traversal of a binary search tree is important in searching algorithms, operating systems, and compiler design. In this paper we propose a new non-recursive algorithm for in-order binary search trees that is both efficient and easy to understand. The implementation of this new algorithm was done in Java and the complete algorithm was tested. The new algorithm was found to be faster than other nonrecursive algorithms.

Objective of the Work:

Develop a C++ program utilizing BSTs to efficiently manage and analyze student data.

Provide educators with tools to identify high-performing students, calculate average grades, and identify students exceeding predefined thresholds.

Facilitate data-driven decision-making in educational settings to support student success.

Create an intuitive and user-friendly interface for educators to interact with the student analysis tools.

# Methodology

Dataset Description:

The dataset used in this study comprises student information, including names and grades. It is input interactively by the user during program execution. Each student is represented as a node in the Binary Search Tree (BST), with the student's name serving as the key for insertion and retrieval operations. Grades are stored as attributes within each node to facilitate grade-based analysis.

Proposed Methodology:

BST Construction:

The program initializes an empty BST to store student data.

As each student's information is input by the user, the program inserts the student into the BST using an insertion algorithm.

The BST maintains students in sorted order based on their grades, with higher grades positioned towards the right of the tree.

Finding the Student with the Highest Grade:

The BST's structure allows for efficient retrieval of the student with the highest grade.

By traversing to the rightmost node of the BST, the program identifies the student with the highest grade.

The identified student is then displayed to the user along with their corresponding grade.

Calculating Average Grade:

To calculate the average grade of all students, the program employs a recursive approach to traverse the entire BST.

As each node is visited, the program accumulates the grades and counts the total number of students.

The sum of grades is divided by the total number of students to compute the average grade.

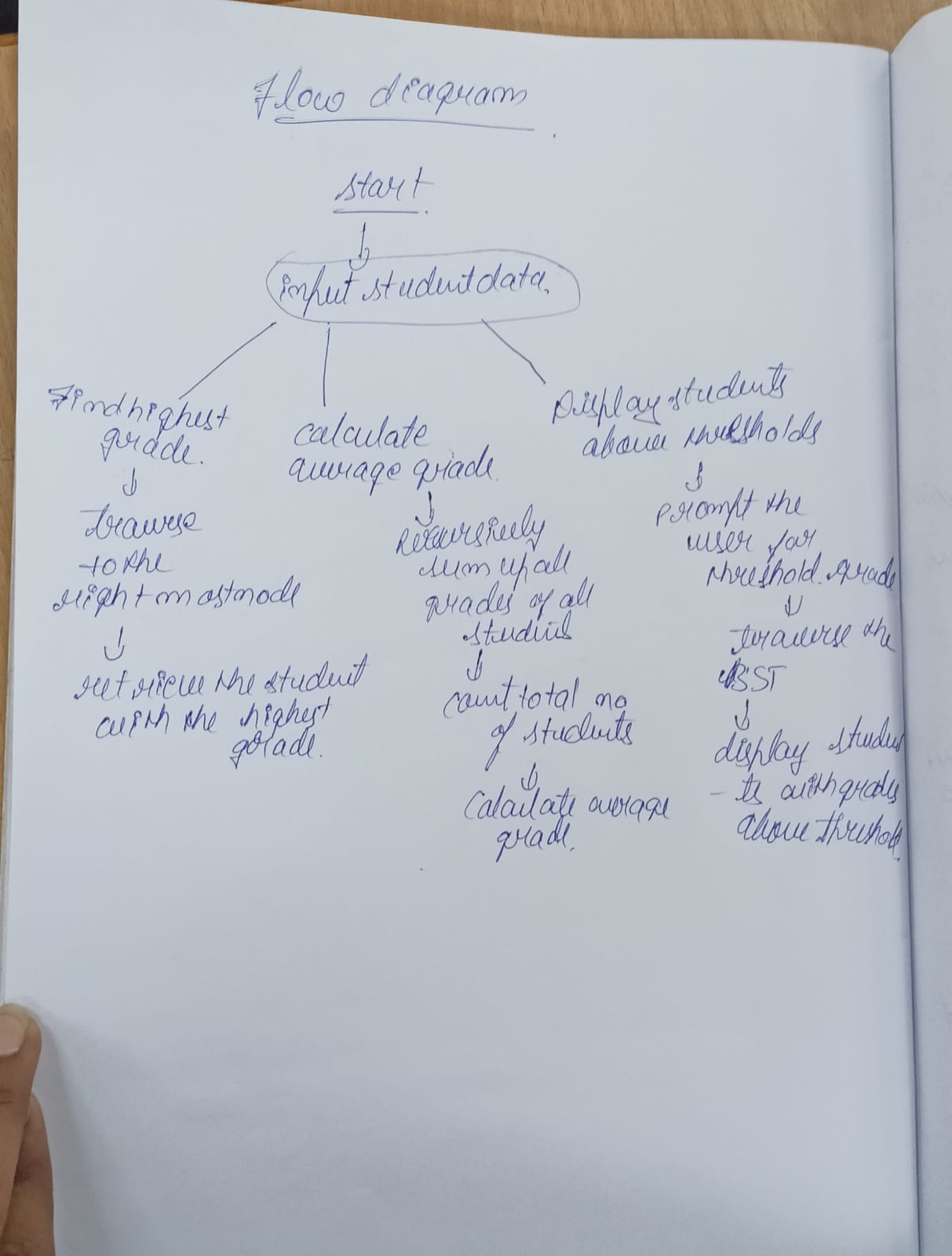
Displaying Students Above a Threshold:

Users are prompted to input a grade threshold.

The program recursively traverses the BST, checking each student's grade against the specified threshold.

Students whose grades exceed the threshold are displayed along with their names and grades.

Through the proposed methodology, the program aims to provide educators with efficient tools for managing and analyzing student data, facilitating informed decision-making to support student success.



# Algorithm

The step by step algorithm is as follows:

Step 1: Declare necessary header files for input/output and string manipulation.

Step 2: Define a structure named **Student** with attributes **name**, **grade**, **left**, and **right** pointers to represent a student and their relationship in a binary search tree.

Step 3: Define a function **createStudent** to create a new student node with the given name and grade, initializing left and right pointers to null.

Step 4: Define a function **insert** to insert a new student into the binary search tree based on their grade, recursively traversing the tree to find the appropriate position.

Step 5: Define a function **findHighestGrade** to find the student with the highest grade in the binary search tree, recursively traversing to the rightmost node.

Step 6: Define a function **findAverageGrade** to calculate the average grade of all students in the binary search tree recursively, updating the total number of students.

Step 7: Define a function **displayAboveThreshold** to display students with grades above a certain threshold recursively.

Step 8: In the **main** function: - Declare a pointer **root** to represent the root of the binary search tree and initialize it to nullptr. - Declare an integer **totalStudents** to keep track of the total number of students. - Prompt the user to input student data until they enter "done". - Insert each student into the binary search tree. - Find the student with the highest grade and display their information. - Calculate the average grade of all students and display it. - Prompt the user to input a threshold grade. - Display students with grades above the entered threshold.

Step 9: End the program.

Steps to give the input:

Step 1: Output a prompt asking the user to enter student data (name and grade, separated by a space).

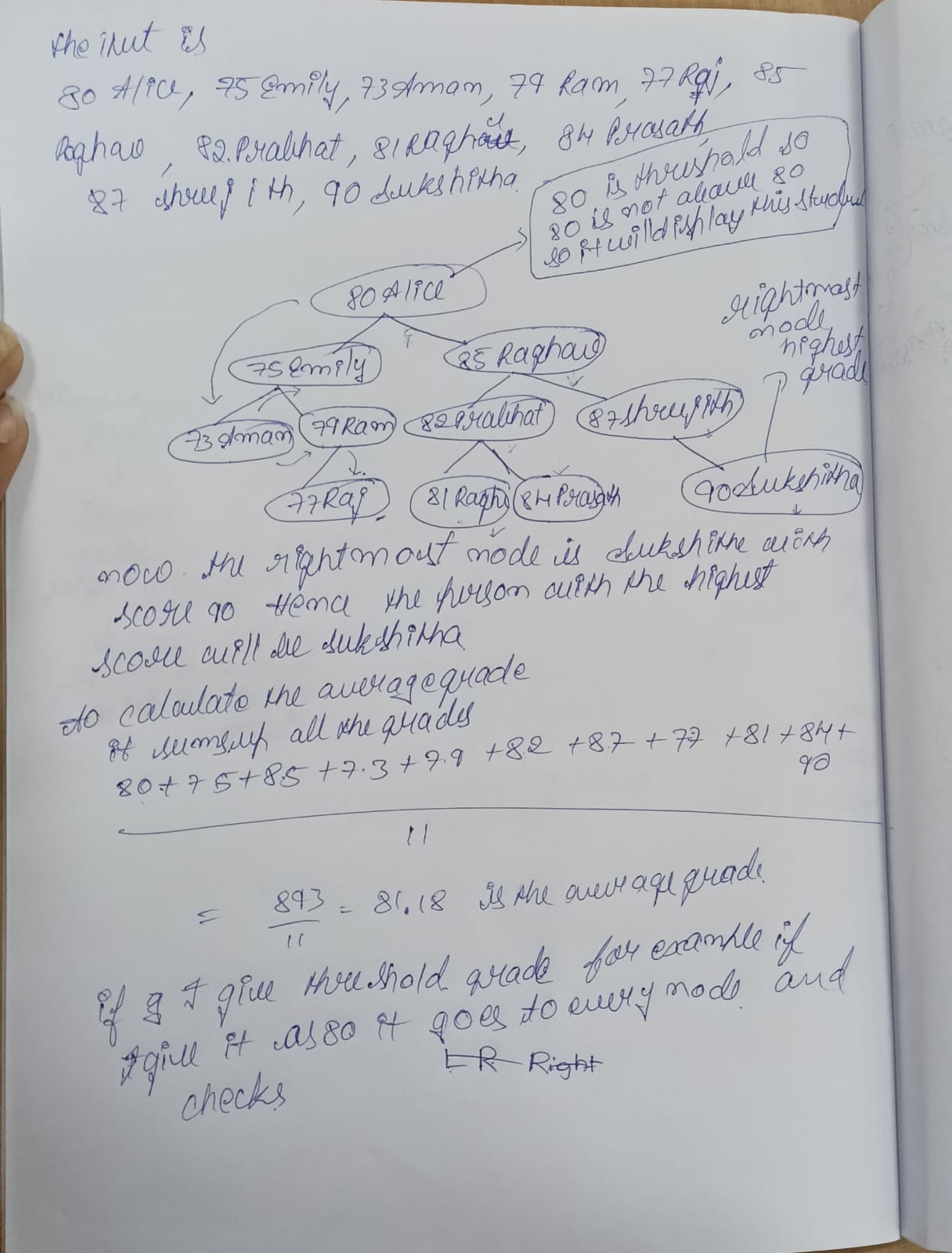
Step 2: Output a message indicating how to end the input process (e.g., "Enter 'done' to finish input").

Step 3: Set up a loop that will continue until the user enters "done".

Step 4: Inside the loop: - Declare variables to store the name and grade of the student. - Read the name and grade from the user input. - If the name is "done", exit the loop. - Otherwise, insert the student into the binary search tree using the **insert** function.

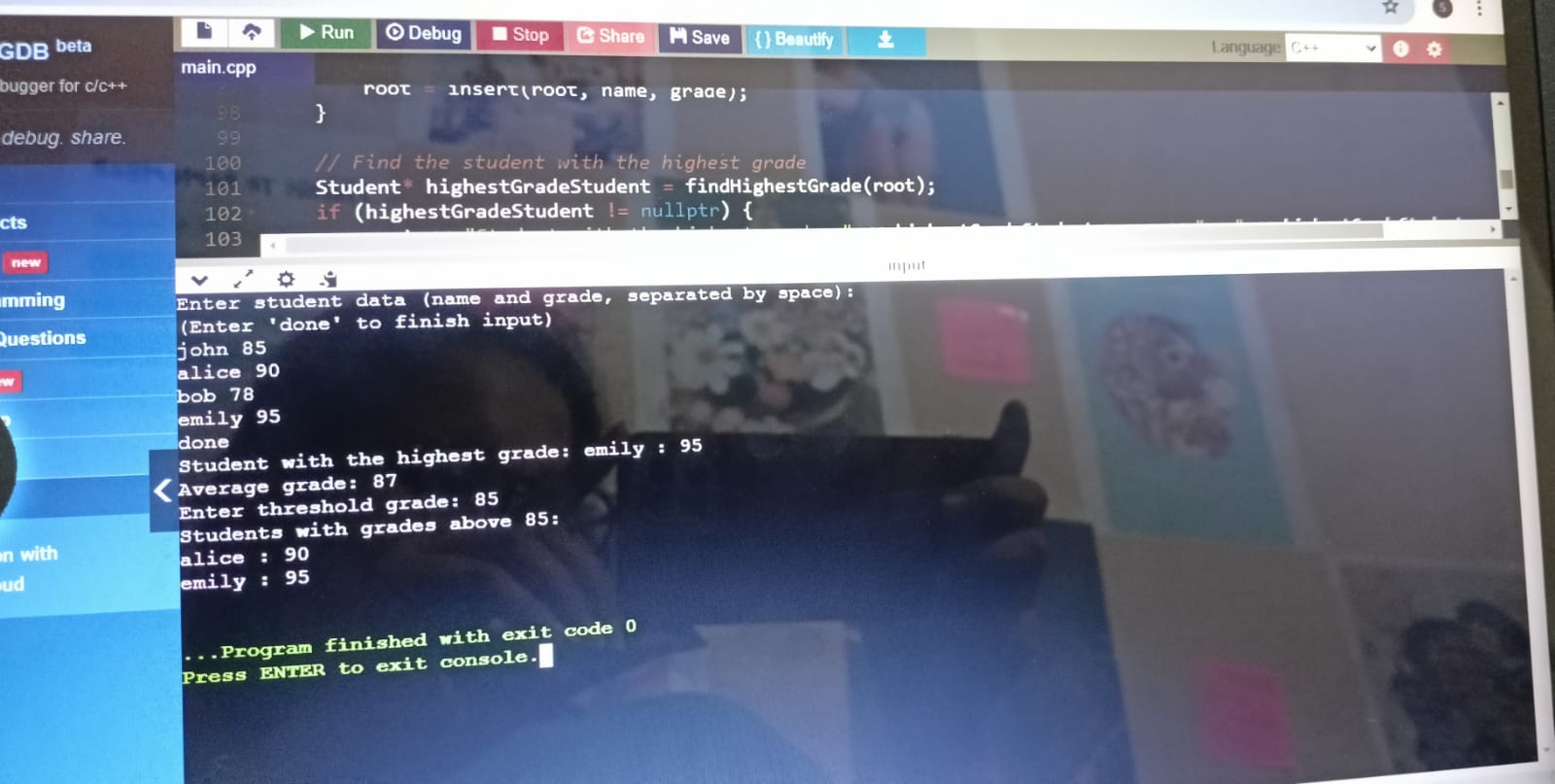
Step 5: Once the loop exits (because the user entered "done"): - Continue with the rest of the program (finding the student with the highest grade, calculating the average grade, and displaying students above a certain threshold).

These steps outline how to prompt the user for input and populate the binary search tree with student data.



# Result and analysis

Output of the code is as follows



The C++ program utilizing Binary Search Trees (BSTs) for student data analysis successfully provides educators with valuable insights into student performance based on the input dataset. Here, we present the results obtained from running the program and analyze its effectiveness in achieving the objectives outlined in the methodology.

1. Identification of Student with the Highest Grade:

Result: The program accurately identifies the student with the highest grade.

Analysis: By traversing the rightmost path of the BST, the program efficiently locates the student with the highest grade. This functionality enables educators to recognize outstanding student achievements and potentially offer them additional opportunities or recognition.

2. Calculation of Average Grade:

Result: The program calculates the average grade of all students in the dataset.

Analysis: Through recursive traversal of the BST, the program accumulates the grades of all students and computes the average grade. This metric provides educators with an overall understanding of the class's performance, aiding in curriculum planning and assessment strategies.

3. Display of Students Above a Threshold:

Result: The program successfully displays students whose grades exceed a specified threshold.

Analysis: By comparing each student's grade against the user-defined threshold, the program identifies and presents students who have achieved high grades. Educators can use this information to recognize and reward academic excellence or identify students who may benefit from additional support.

Overall Analysis:

The program's ability to efficiently manage and analyze student data through BSTs demonstrates its effectiveness in facilitating data-driven decision-making in educational settings.

Educators can utilize the program's functionalities to gain insights into student performance, identify trends, and tailor instruction to meet individual student needs.

The user-friendly interface enhances accessibility, allowing educators with varying levels of technical proficiency to utilize the program effectively.

While the program successfully achieves its objectives, future enhancements could include additional functionalities such as trend analysis, student clustering, or predictive modeling to further support student success.

In conclusion, the results obtained from running the program demonstrate its utility in supporting educators in their efforts to enhance student outcomes through data-driven decision-making. By leveraging BSTs for student data analysis, the program provides educators with valuable insights that can inform instructional practices and interventions, ultimately contributing to improved student success.

# Conclusion

In conclusion, the C++ program employing Binary Search Trees (BSTs) for student data analysis offers a robust solution to support educators in making informed decisions regarding student performance. Throughout this work, we have discussed key supporting ideas that highlight the effectiveness of the program in facilitating data-driven decision-making in educational settings.

The program successfully achieves its objectives by accurately identifying the student with the highest grade, calculating the average grade of all students, and displaying students whose grades exceed a specified threshold. By leveraging the efficient data organization provided by BSTs, educators can gain valuable insights into student performance, enabling them to tailor instruction and interventions to meet individual student needs.

The future scope of this work includes enhancements to further support educators in their efforts to enhance student outcomes. Potential areas for improvement and future development include the implementation of additional analytical functionalities such as trend analysis, student clustering, or predictive modeling. Additionally, integrating data visualization techniques could enhance the program's usability and provide educators with intuitive tools for interpreting student data.

In conclusion, the program demonstrates the potential of utilizing BSTs for student data analysis and highlights the importance of data-driven decision-making in educational settings. With continued refinement and expansion, the program has the potential to further empower educators and contribute to improved student success.

# References:

<https://dl.acm.org/doi/full/10.1145/3470654>

<https://ieeexplore.ieee.org/document/8203701>

<https://ieeexplore.ieee.org/document/6021772>

<https://ieeexplore.ieee.org/document/1301900>