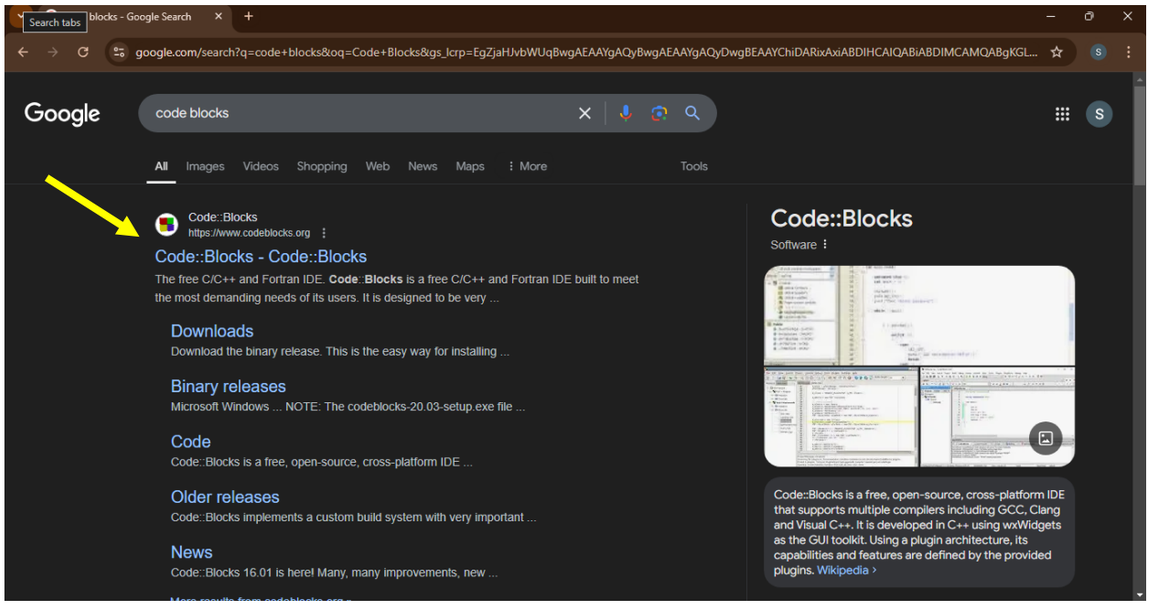
Project Overview

This project provides an implementation of fundamental data structures in C++, including Binary Trees, Binary Search Trees (BST), and Heaps (Max-Heap and Min-Heap). These data structures are essential tools, facilitating efficient data management and optimization for operations such as searching, sorting, and priority-based tasks. The primary aim of this project is to show the practical application of these data structures and to offer an interactive platform for users to perform various operations, such as node insertion, deletion, searching, and traversal. The program presents a user-friendly, menu-driven interface that allows users to test and understand the core functionalities of each data structure.

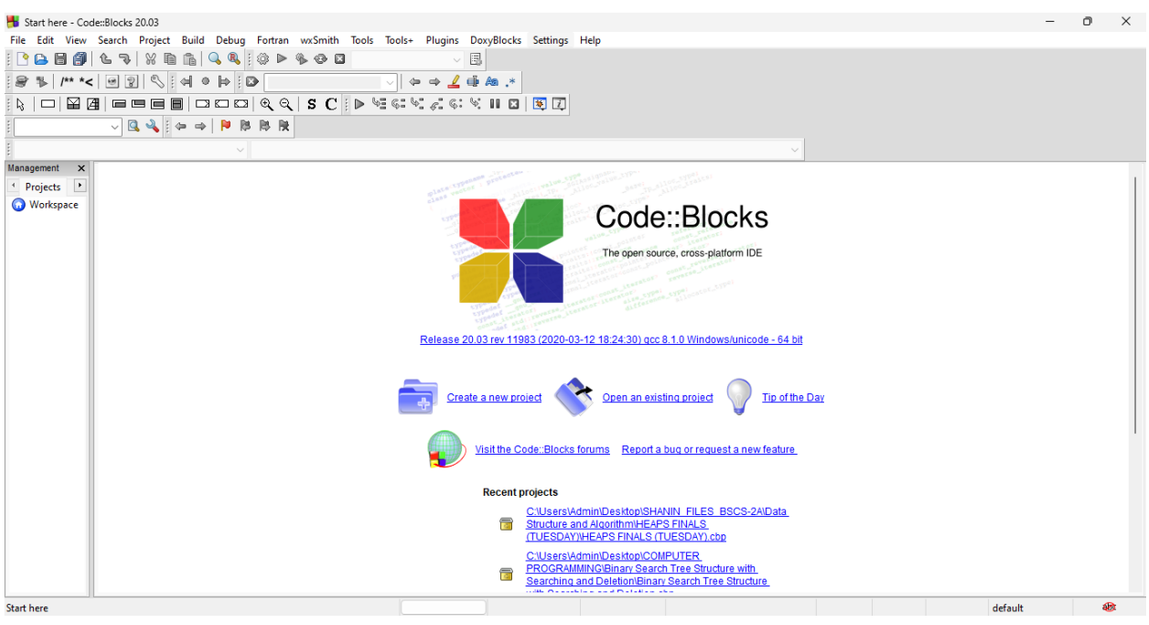
How to Run the Code

Simply follow these steps to run the program in a C++ compiler like Code:Blocks:

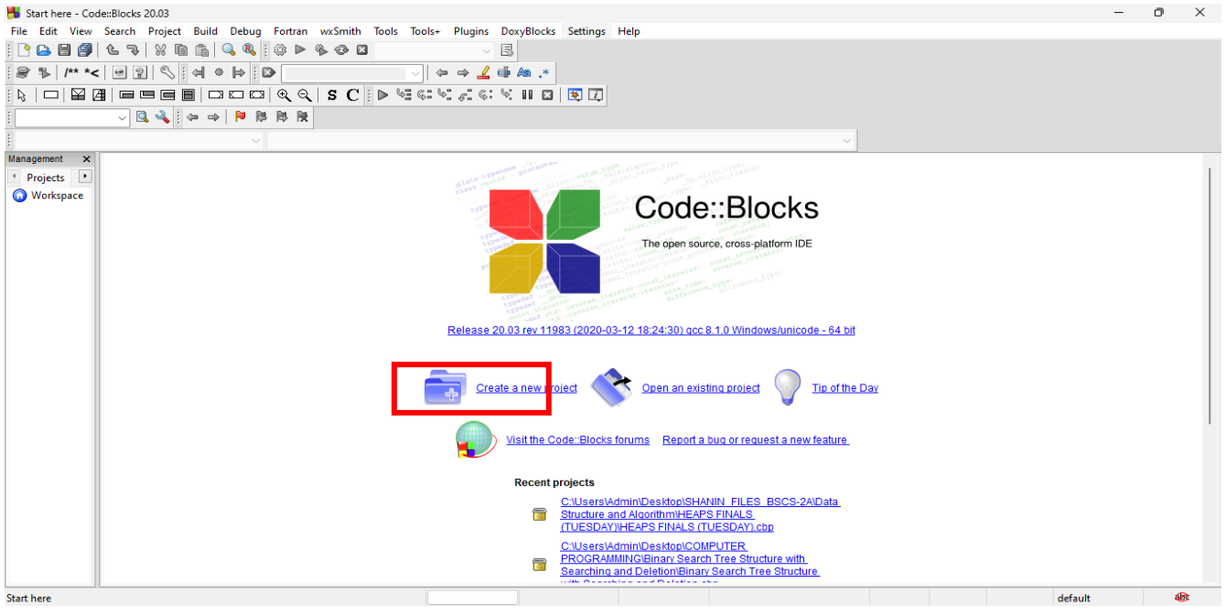
1. **Install Code::Blocks** (if not already installed):
    a. If you do not have Code::Blocks installed, download and install it from the official Code::Blocks website.



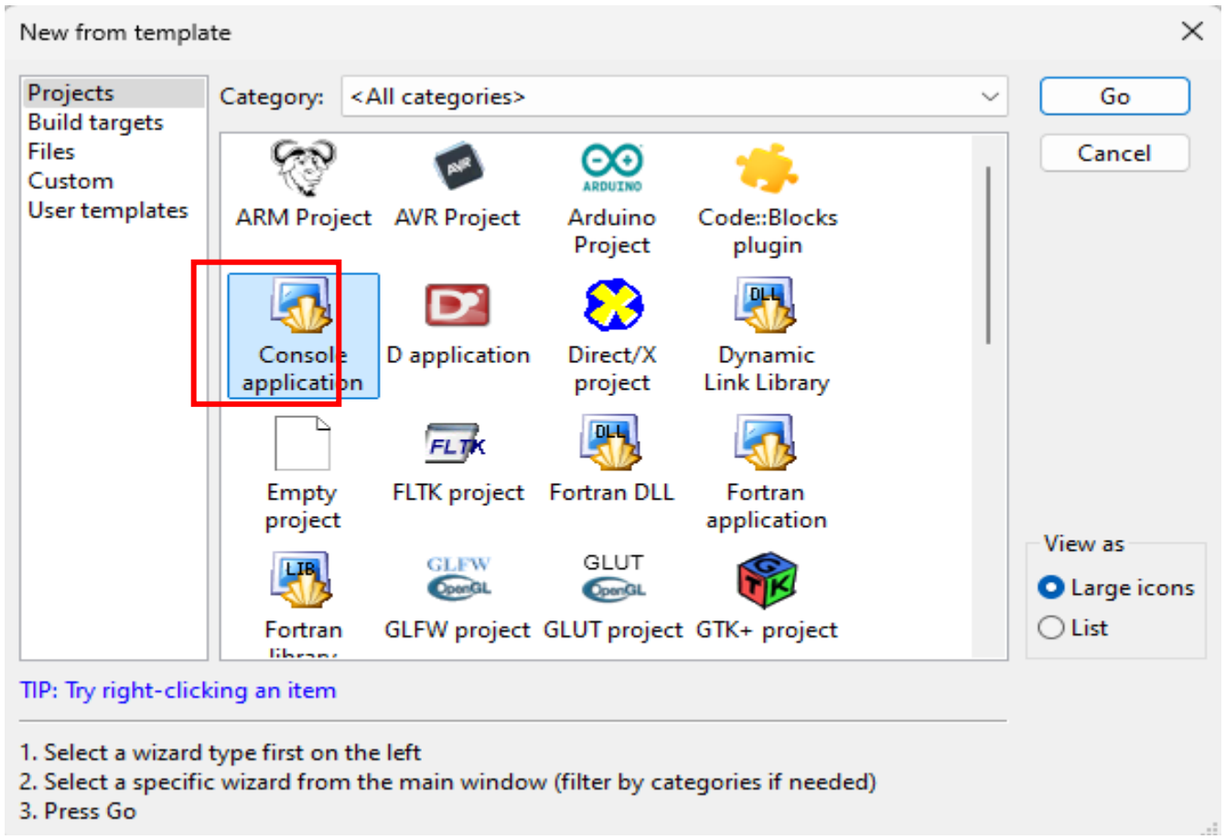
2. **Create a New Project in Code::Blocks**:
    a. Open Code::Blocks.



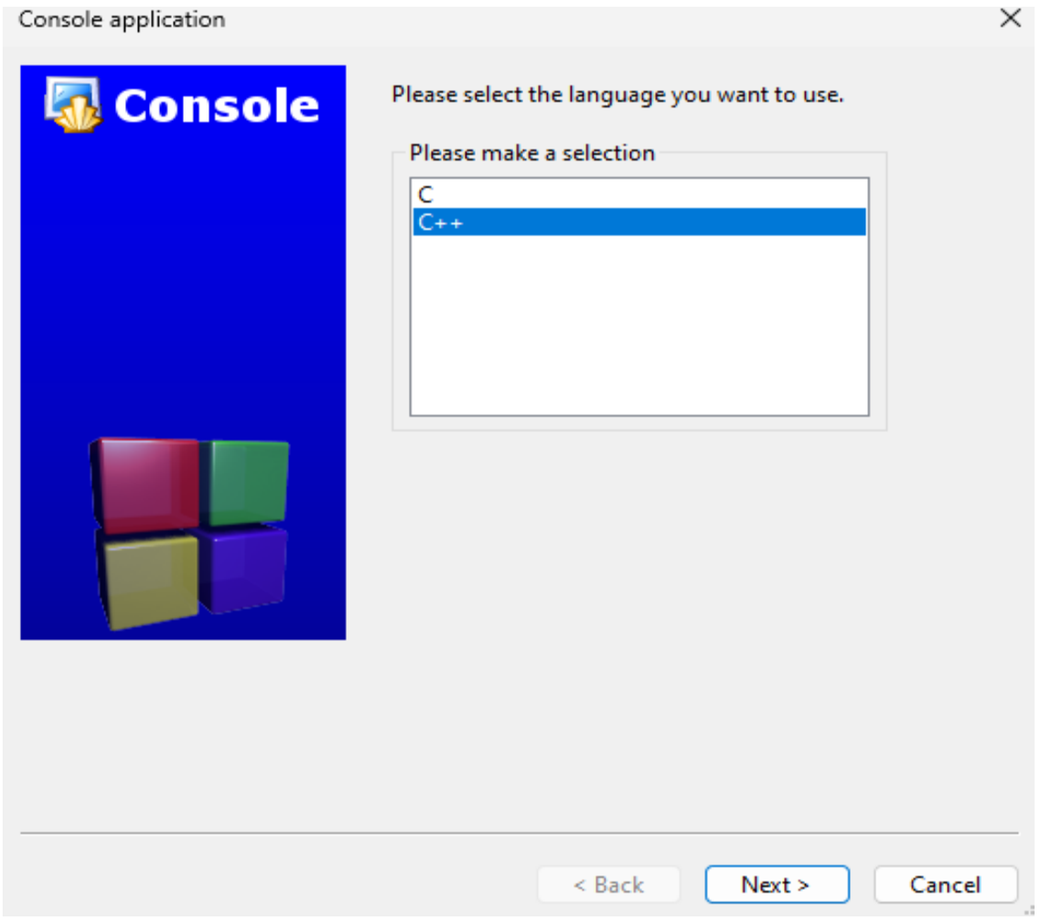
b. Click Create a new project.



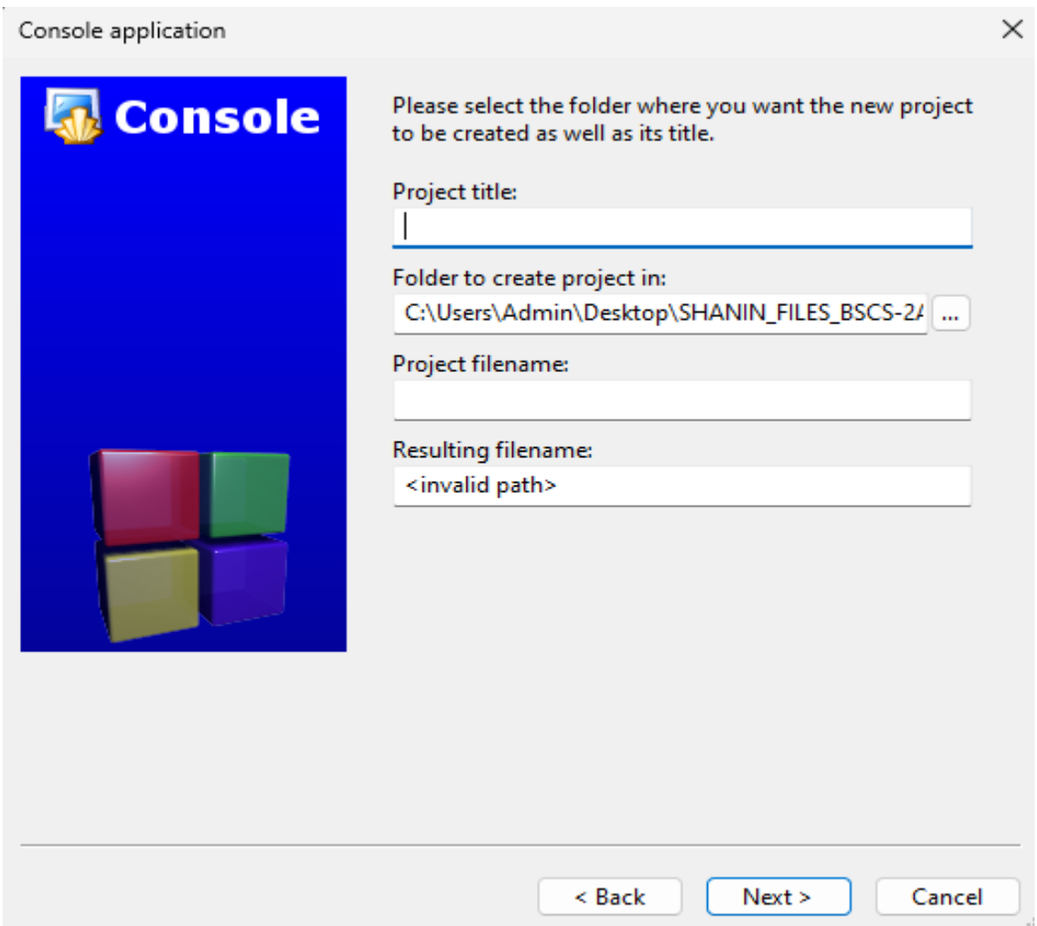
b. Select **Console Application** and click **Go**.



c. Choose **C++** as the language and click **Next**.

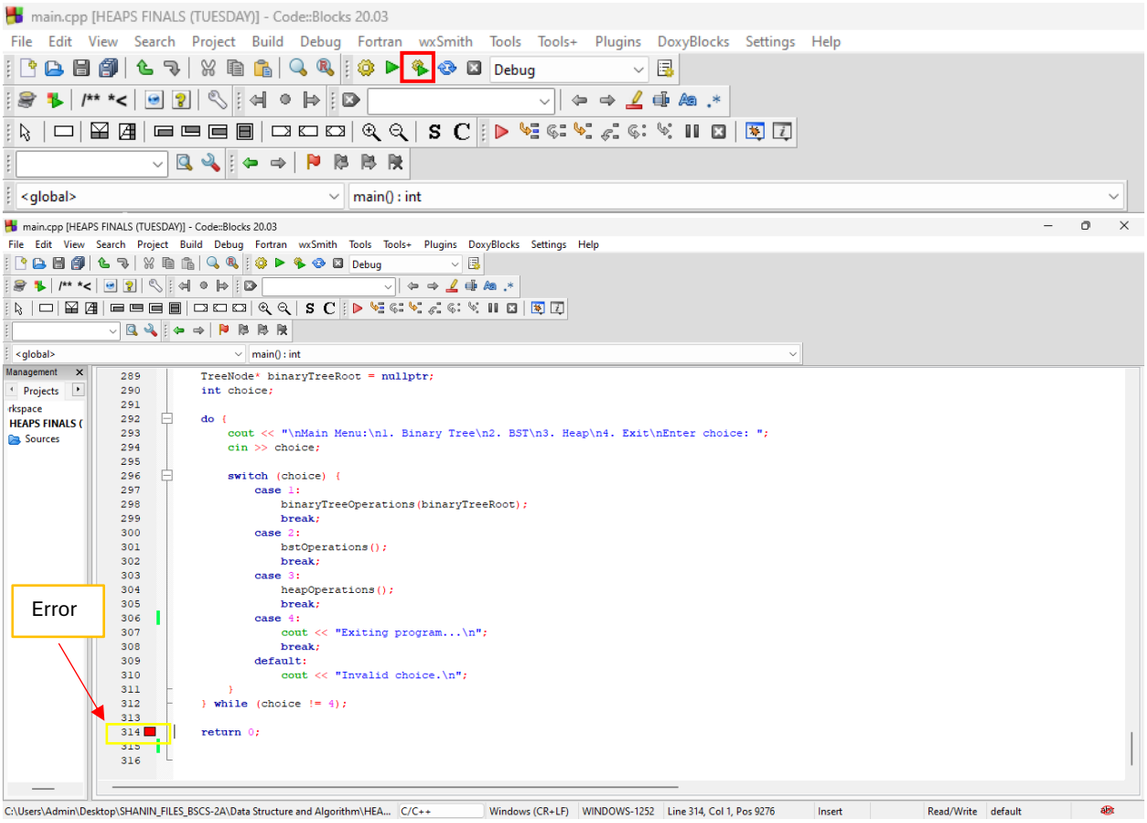


d. Provide a project name and choose a location for your project files. Click **Next** and then **Finish**.



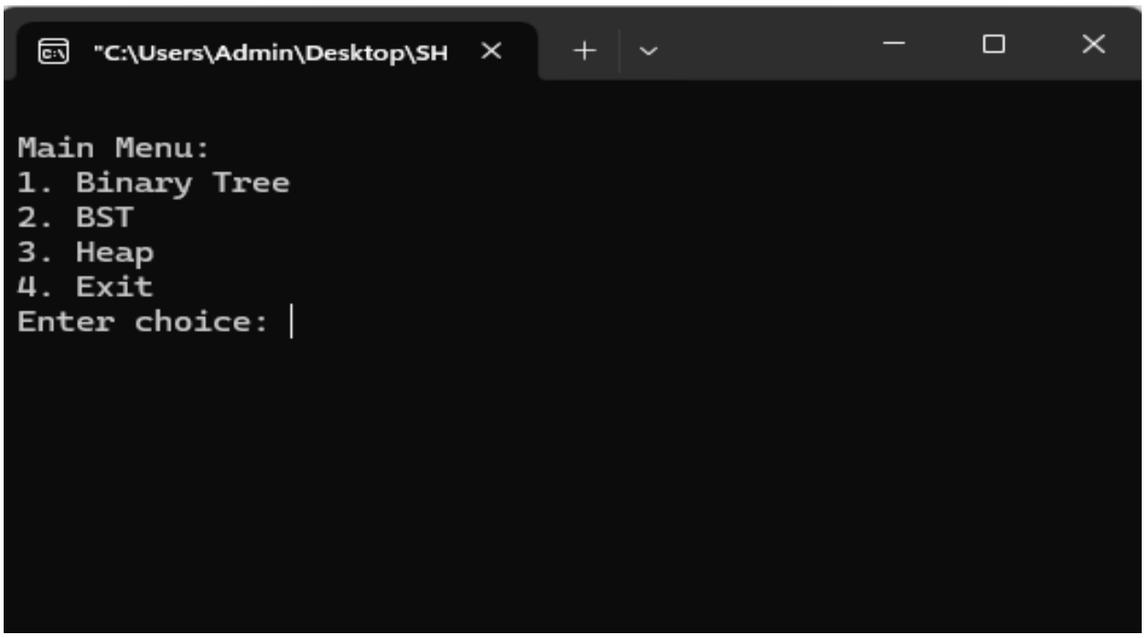
3. **Build the Project**:

In the Code::Blocks IDE, click the **Build and Run** button (gear icon with green play icon) to compile the code. Ensure there are no compilation errors or warnings.



4. **Run the Program**:

The console window will appear, and you will be prompted to interact with the program by choosing options for Binary Trees, BSTs, or Heaps and Exit.



5. **Interact with the Program**:

    a. Follow the on-screen instructions to insert elements, search for values, delete nodes, and explore various tree and heap operations. You can choose from the available menu options to test different functionalities.

Description of Each Functionality

Binary Tree Operations

A Binary Tree is a hierarchical data structure in which each node has at most two children, known as the left and right children. This project supports the following essential Binary Tree operations: insertion, searching, deletion, and traversals (preorder, inorder, and postorder). Insertion follows a level-order technique, with nodes inserted to the next available location. When searching for a node, each node is recursively checked until a match is discovered. Deletion necessitates redesigning the tree to ensure that it remains valid when a node is removed. Traversals allow you to explore the nodes in three different orders: preorder (root, left, right), inorder (left, root, right), and postorder (left, right, root), each presenting a unique picture of the tree.

Binary Search Tree (BST) Operations

A Binary Search Tree (BST) is a sort of binary tree that has the following ordering property: for any given node, all values in the left subtree are smaller, and all values in the right subtree are greater. This characteristic allows for efficient searching, insertion, and deletion operations. Insertion in a BST is conducted using the binary search property, which ensures that the tree remains ordered. Searching in a BST is efficient because it eliminates half of the tree in each phase. Deletion in a BST is more complicated and involves three cases: deleting a leaf node, removing a node with one child, and removing a node with two children. The tree can be navigated in three ways: preorder, inorder, and postorder, each providing a distinct perspective on the structure.

Heap Operations (Max-Heap and Min-Heap)

A heap is a complete binary tree with certain ordering properties. A Max-Heap assures that the value of each parent node is greater than or equal to the values of its offspring, allowing the root to always contain the largest element. In contrast, a Min-Heap ensures that the value of each parent node is less than or equal to that of its offspring, resulting in the lowest element accessible at the root. Heap operations include heapify, which restores the heap property after an operation such as insertion or deletion, and buildHeap, which creates a legal heap from an unsorted array. The software lets users choose between Max-Heap and Min-Heap operations, allowing them to see the differences in how the heap property is maintained and interact with the heap in both forms.

Three fundamental data structures—Binary Trees, Binary Search Trees, and Heaps—are thoroughly introduced in this project. It gives users a hands-on experience with the fundamental ideas and functions that underlie these data structures by providing an interactive and user-friendly environment. The application is intended to be used as a teaching tool and a hands-on example of how these C++ structures work.