# BST Documentation

* Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Name* | *Type* | *Protection* | *Description* | *Rationale* |
| **BST** | **Class** |  | **This class implements a template-based Binary Search Tree using recursive operations for insertion, searching, traversal, and copying.** | **The BST class encapsulates a reusable and templated structure that enables the storage and efficient retrieval of ordered data through recursive tree operations.** |
| M\_root | Node<T>\* | Private | A pointer to the root of the Binary Search Tree. | Made private to enforce encapsulation and prevent external modification and access. It forms the core root to all other nodes in the tree. |
| BST() | Constructor | Public | This constructor will construct a new object with root set to nullptr. | Required to set the root to nullptr to prevent garbage values being assigned to root. |
| BST(const BST<T> & other) | Constructor | Public | This constructor will overload the default constructor, passing a BST object. The passed object's data is copied to the constructed object's, and creating a new tree on the heap that copies the data from the passed object's array. | Enables the creation of a memory-safe deep copy of another BST. |
| ~BST() | Destructor | Public | This destructor will free the dynamically allocated memory on the heap and set the data of every node in the tree to nullptr. | Enables memory clean up due to dynamic memory. |
| Operator=(const BST<T> & other) | Operator | Public | This method overloads the assignment operator, copying the data in the other BST object into the assigned object. | Enables the creation of a memory-safe deep copy of another BST being assigned to the LHS BST. |
| Insert(const T val) | Procedure | Public | Adds a new value into the tree while maintaining BST ordering rules. | Public interface to only allow user to pass the specified value without requiring the passing of pointers which will lead to security violations. |
| Search(const T val) | Method | Public | Checks whether a value exists in the BST. | Public interface to only allow user to pass the specified value without requiring the passing of pointers which will lead to security violations. |
| InOrderTraversal(void (\*fp)(T &)) | Procedure | Public | Perform in-order traversal using a function pointer callback. | Public interface to only allow user to call the method without requiring the passing of pointers which will lead to security violations. |
| PreOrderTraversal(void (\*fp)(T &)) | Procedure | Public | Perform pre-order traversal using a function pointer callback. | Public interface to only allow user to call the method without requiring the passing of pointers which will lead to security violations. |
| PostOrderTraversal(void (\*fp)(T &)) | Procedure | Public | Perform post-order traversal using a function pointer callback. | Public interface to only allow user to call the method without requiring the passing of pointers which will lead to security violations. |
| CopyTree(const BST<T> & other) | Procedure | Private | Creates a deep copy of another BST into the calling object. | Private class method to be called by overloaded assignment operator and copy constructor. Invokes the recursive method for copying the tree nodes. |
| Copy(Node<T>\*& thisNode, Node<T>\* otherNode) | Procedure | Private | Recursively copies all nodes from another tree into this calling object's tree. | Marked private because this method supports internal recursive operations. They should not be exposed to the user directly to avoid misuse or violating the class invariants. |
| DeleteTree(Node<T>\*& node) | Procedure | Private | Frees memory allocated for all nodes starting from the given node. | Marked private because this method supports internal recursive operations. They should not be exposed to the user directly to avoid misuse or violating the class invariants. |
| Insert(Node<T>\*& node, const T& val) | Procedure | Private | Inserts a value into the correct position in the BST. | Marked private because this method supports internal recursive operations. They should not be exposed to the user directly to avoid misuse or violating the class invariants. |
| Search(Node<T>\* node, const T& val) | Method | Private | Searches for a value in the tree. | Marked private because this method supports internal recursive operations. They should not be exposed to the user directly to avoid misuse or violating the class invariants. |
| InOrderTraversal(void (\*fp)(T &), Node<T>\* node) | Procedure | Private | Recursively visits all nodes in ascending order and applies the callback function to each node's data. | Marked private because this method supports internal recursive operations. They should not be exposed to the user directly to avoid misuse or violating the class invariants. |
| PreOrderTraversal(void (\*fp)(T &), Node<T>\* node) | Procedure | Private | Recursively visits the current node before its subtrees and applies the callback function to each node's data. | Marked private because this method supports internal recursive operations. They should not be exposed to the user directly to avoid misuse or violating the class invariants. |
| PostOrderTraversal(void (\*fp)(T &), Node<T>\* node) | Procedure | Private | Recursively visits subtrees before the current node and applies the callback function to each node's data. | Marked private because this method supports internal recursive operations. They should not be exposed to the user directly to avoid misuse or violating the class invariants. |
|  |  |  |  |  |
| **Node** | **Struct** |  | **A record acting as a node of the binary search tree, containing the actual data and pointers to the left and right nodes** | **Stores data as a struct because Node serves as a passive data holder for each node of the BST. It contains no behavior, logic, or processing, as all operations are performed externally, typically through its use in a BST. This design maintains high cohesion and low coupling.** |
| Data | T | Public | The actual data in the node. | Templated to allow the node to be reusable for different data types or classes. |
| Left | Node<T>\* | Public | The pointer to the left subtree. | Templated to allow the node to be reusable for different data types or classes. |
| Right | Node<T>\* | Public | The pointer to the right subtree. | Templated to allow the node to be reusable for different data types or classes. |

* BSTTest.cpp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description (and why) | Actual Test Data | Expected Output | Pass/Fail |
| 1 | Default constructor sets m\_root to null. | N/A - Default constructor called. In-order traversal called. | No output. Empty tree. | Pass |
| 2 | Insert a single value into an empty BST. | Insert: 50  In-order traversal called. | 50 | Pass |
| 3 | Insert unordered values into a BST, and stored in order, validating RI through in-order traversal. | Insert: 50, 6, 25, 1  In-order traversal called. | 1 6 25 50 | Pass |
| 4 | Attempt to insert a duplicate value. | Insert: 50, 6, 50  In-order traversal called | Duplicates not allowed.  6 50 | Pass |
| 5 | Search for an existing value. | Insert: 50, 6, 25, 1  Search: 50 | Number found. | Pass |
| 6 | Search for a non-existing value. | Insert: 50, 6, 25, 1  Search: 20 | Number not found. | Pass |
| 7 | Traverse empty BST. | Empty BST with no insertions.  In-order traversal called. | No output. Empty tree. | Pass |
| 8 | Traverse BST in pre-order. | Insert: 50, 6, 25, 1.  Pre-order traversal called. | 50 6 1 25 | Pass |
| 9 | Traverse BST in post-order. | Insert: 50, 6, 25, 1  Post-order traversal called. | 1 25 6 50 | Pass |
| 10 | Copy constructor and overloaded assignment operator copy all elements in the BST and make a deep copy. | BST<int> A: 50, 6, 25, 1  Copy A to BST<int> B.  Assign A to BST<int> C.  Modify A by inserting 10.  Modify B by inserting 2.  In-order traversal called for all trees. | A: 1 6 10 25 50  B: 1 2 6 25 50  C: 1 6 25 50 | Pass |
| 11 | BST object is copied correctly when passed by value. | BST<int> bst1: 50, 6, 25, 1  BST<int> bst\_copy: 50, 6, 25, 1  Insert to bst\_copy: 23  In-order traversal called for both trees. | Bst\_copy: 1 6 23 25 50  Bst1: 1 6 25 50 | Pass |
| 12 | BST object is correctly modified when passed by non-const reference. | BST<int> bst1: 50, 6, 25, 1  BST<int> bst\_ref: 50, 6, 25, 1  Insert to bst\_ref: 23  In-order traversal called for both trees. | Bst\_ref: 1 6 23 25 50  Bst1: 1 6 23 25 50 | Pass |
| 13 | BST object data can be accessed but not modified when passed by const reference. | BST<int> bst1: 50, 6, 25, 1  BST<int> bst\_const\_ref: 50, 6, 25, 1  Insert to bst\_ const \_ref: 23 | Compiler prints an error when insert in PassByConstRef() is left uncommented and called. | Pass |
| 14 | BST is correctly returned from a function by value. | BST in function: 50, 6, 25, 1 | BST in Function: 1 6 25 50  BST from Return Function: 1 6 25 50 | Pass |
| 15 | BST templating works with custom Date class. | Insert Dates: 01/01/2020, 15/03/2021, 31/12/2019, 29/02/2024, 04/07/2022 In-order traversal called. | 31/12/2019 01/01/2020 15/03/2021 04/07/2022 29/02/2024 | Pass |