Q1. Program to implement the following functions in a Binary search tree using templates.

a) Insertion

b) Inorder Traversal

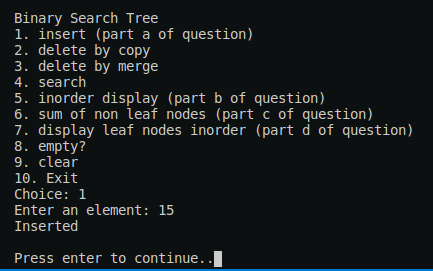
c) Function to compute the sum of values of all non-leaf nodes in a tree.

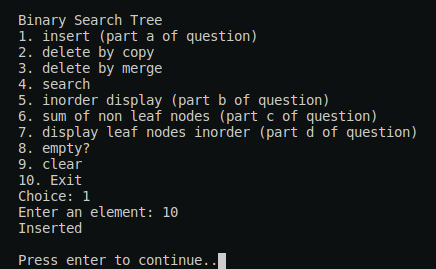
d) Display only the leaf nodes of the tree in inorder.

**Code is given below output.**

Outputs -

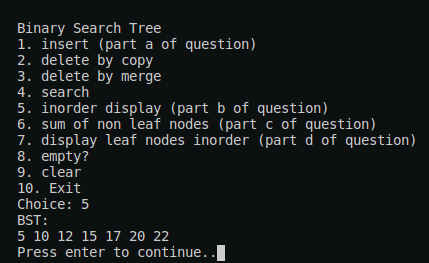
# Insertion



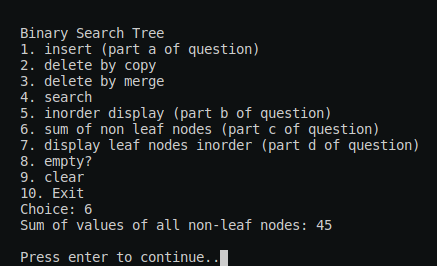


### Similarly, 20, 5, 12, 17, 22 is inserted.

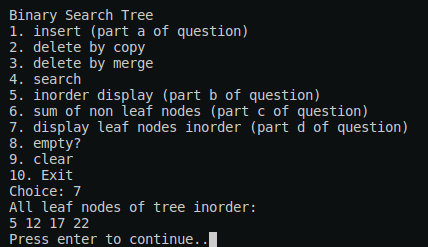
# Inorder Traversal



# Function to compute the sum of values of all non-leaf nodes in a tree.



# Display only the leaf nodes of the tree in inorder.



# 

Code -

### **bst.h**

|  |
| --- |
| #ifndef BST\_H  #define BST\_H  #include <iostream>  #include <queue>  *template* <*typename* type>  *class* BST;  *template* <*typename* type>  *class* BSTNode  {  *private:*  type key;  BSTNode<type>\* left;  BSTNode<type>\* right;  friend *class* BST<type>;  *public:*  BSTNode();  BSTNode(const type& *key*, BSTNode<type>\* *left*=nullptr, BSTNode<type>\* *right*=nullptr);  };  *template* <*typename* type>  *class* BST  {  *private:*  BSTNode<type>\* root;  *void* clear(BSTNode<type>\* *node*);  BSTNode<type>\* search(BSTNode<type>\* *node*, const type& *key*) const;  *void* inorder(BSTNode<type>\* *node*);  inline *void* visit(BSTNode<type>\* *node*) const;  *int* sum\_nonleaf(BSTNode<type>\* *node*); // part c  *void* leaf\_inorder(BSTNode<type>\* *node*); // part d  *public:*  BST();  ~BST();  inline *void* clear();  inline *bool* empty() const;  *void* insert(const type& *key*); // part a  *void* find\_del\_copy(const type& *key*);  *void* del\_copy(BSTNode<type>\*& *node*);  *void* find\_del\_merge(const type& *key*);  *void* del\_merge(BSTNode<type>\*& *node*);  BSTNode<type>\* search(const type& *key*) const;  *void* inorder(); // part b  inline *int* sum\_nonleaf(); // part c  *void* leaf\_inorder(); // part d  };  #include "bst.tpp"  #endif |

**bst.tpp -**

|  |
| --- |
| **#include "bst.h"**  **// Binary Search Node Class**  ***template* <*typename* *type*>**  ***BSTNode*<*type*>::BSTNode()**  **{**  **left = right = nullptr;**  **}**  ***template* <*typename* *type*>**  ***BSTNode*<*type*>::BSTNode(const *type*& *key*, *BSTNode*<*type*>\* *left*, *BSTNode*<*type*>\* *right*)**  **{**  **this->key = *key*;**  **this->left = *left*;**  **this->right = *right*;**  **}**  **// Binary Search Tree Class**  ***template* <*typename* *type*>**  ***BST*<*type*>::BST()**  **{**  **this->root = nullptr;**  **}**  ***template* <*typename* *type*>**  ***BST*<*type*>::~BST()**  **{**  **this->clear();**  **}**  ***template* <*typename* *type*>**  **inline *void* *BST*<*type*>::clear() // Public clear()**  **{**  **this->clear(this->root);**  **this->root = nullptr;**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::clear(*BSTNode*<*type*>\* *node*) // Private clear()**  **{**  **if (*node* == nullptr)**  **return;**  **this->clear(*node*->left);**  **this->clear(*node*->right);**  **delete *node*;**  **}**  ***template* <*typename* *type*>**  **inline *bool* *BST*<*type*>::empty() const**  **{**  **return this->root == nullptr;**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::insert(const *type*& *key*)**  **{**  **if (this->root == nullptr)**  **this->root = new *BSTNode*<*type*>(*key*);**  **else**  **{**  ***BSTNode*<*type*>\* temp\_node = this->root; // to traverse the tree**  ***BSTNode*<*type*>\* prev\_node = nullptr; // parent of temp\_node**  **while (temp\_node != nullptr) // find right position**  **{**  **prev\_node = temp\_node;**  **if (temp\_node->key < *key*)**  **temp\_node = temp\_node->right;**  **else // if (temp\_node->key >= key)**  **temp\_node = temp\_node->left;**  **}**  **if (prev\_node->key < *key*)**  **prev\_node->right = new *BSTNode*<*type*>(*key*);**  **else**  **prev\_node->left = new *BSTNode*<*type*>(*key*);**  **}**  **}**  ***template* <*typename* *type*>**  ***BSTNode*<*type*>\* *BST*<*type*>::search(const *type*& *key*) const**  **{**  **return this->search(this->root, *key*);**  **}**  ***template* <*typename* *type*>**  ***BSTNode*<*type*>\* *BST*<*type*>::search(*BSTNode*<*type*>\* *node*, const *type*& *key*) const**  **{**  **while (*node* != nullptr)**  **{**  **if (*node*->key == *key*)**  **return *node*;**  **if (*node*->key < *key*)**  ***node* = *node*->right;**  **else**  ***node* = *node*->left;**  **}**  **return nullptr;**  **}**  **// find the parent to node ptr with key and send it to del\_copy function to delete**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::find\_del\_merge(const *type*& *key*)**  **{**  ***BSTNode*<*type*>\* temp\_node = this->root;**  ***BSTNode*<*type*>\* prev\_node = nullptr;**  **// find the ptr from parent to the node with key = key**  **while (temp\_node != nullptr && temp\_node->key != *key*)**  **{**  **prev\_node = temp\_node;**  **if (temp\_node->key < *key*)**  **temp\_node = temp\_node->right;**  **else**  **temp\_node = temp\_node->left;**  **}**  **if (temp\_node != nullptr && temp\_node->key == *key*)**  **{**  **if (temp\_node == this->root)**  **del\_merge(root);**  **else if (temp\_node == prev\_node->left)**  **del\_merge(prev\_node->left);**  **else**  **del\_merge(prev\_node->right);**  **}**  **else if (this->root != nullptr)**  **std::cout << "Key not found in the binary search tree\n"; // should have thrown an error**  **else**  **std::cout << "Binary Search Tree is empty\n"; // should have thrown an error**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::del\_merge(*BSTNode*<*type*>\*& *node*) // node is parent -> child ptr**  **{**  **if (*node* != nullptr)**  **{**  ***BSTNode*<*type*>\* to\_delete = *node*;**  **if (*node*->left == nullptr)**  ***node* = *node*->right;**  **else if (*node*->right == nullptr)**  ***node* = *node*->left;**  **else // both left and right child exists**  **{**  ***BSTNode*<*type*>\* temp\_node = *node*->left;**  **while (temp\_node->right != nullptr)**  **temp\_node = temp\_node->right; // get the right most child of left child of to\_delete node**    **temp\_node->right = *node*->right;**  ***node* = *node*->left;**  **}**  **delete to\_delete;**  **}**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::find\_del\_copy(const *type*& *key*)**  **{**  ***BSTNode*<*type*>\* temp\_node = this->root;**  ***BSTNode*<*type*>\* prev\_node = nullptr;**  **while (temp\_node != nullptr && temp\_node->key != *key*)**  **{**  **prev\_node = temp\_node;**  **if (temp\_node->key < *key*)**  **temp\_node = temp\_node->right;**  **else**  **temp\_node = temp\_node->left;**  **}**  **if (temp\_node != nullptr && temp\_node->key == *key*)**  **{**  **if (temp\_node == this->root)**  **del\_copy(root);**  **else if (temp\_node == prev\_node->right)**  **del\_copy(prev\_node->right);**  **else**  **del\_copy(prev\_node->left);**  **}**  **else if (this->root != nullptr)**  **std::cout << "Key not found in the binary search tree\n"; // should have thrown an error**  **else**  **std::cout << "Binary search tree is empty\n"; // should have thrown an error**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::del\_copy(*BSTNode*<*type*>\*& *node*)**  **{**  **if (*node* != nullptr)**  **{**  **if (*node*->left == nullptr)**  ***node* = *node*->right;**  **else if (*node*->right == nullptr)**  ***node* = *node*->left;**  **else**  **{**  ***BSTNode*<*type*>\*prev\_node = *node*;**  ***BSTNode*<*type*>\* temp\_node = *node*->left;**  **while (temp\_node->right != nullptr)**  **{**  **prev\_node = temp\_node;**  **temp\_node = temp\_node->right;**  **}**  ***node*->key = temp\_node->key; // copy the elements**  **if (prev\_node == *node*) // handle the left sub tree of temp\_node**  **prev\_node->left = temp\_node->left;**  **else**  **prev\_node->right = temp\_node->left;**  **delete temp\_node;**  **}**  **}**  **}**  ***template* <*typename* *type*>**  **inline *void* *BST*<*type*>::visit(*BSTNode*<*type*>\* *node*) const**  **{**  **std::cout << *node*->key << " ";**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::inorder()**  **{**  **inorder(this->root);**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::inorder(*BSTNode*<*type*>\* *node*)**  **{**  **if (*node* == nullptr)**  **return;**  **inorder(*node*->left);**  **visit(*node*);**  **inorder(*node*->right);**  **}**  ***template* <*typename* *type*>**  ***int* *BST*<*type*>::sum\_nonleaf()**  **{**  **return sum\_nonleaf(this->root);**  **}**  ***template* <*typename* *type*>**  ***int* *BST*<*type*>::sum\_nonleaf(*BSTNode*<*type*>\* *node*)**  **{**  **if (*node* == nullptr)**  **return 0;**  **if (*node*->left == nullptr && *node*->right == nullptr)**  **return 0;**    **return sum\_nonleaf(*node*->left) + sum\_nonleaf(*node*->right) + *node*->key;**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::leaf\_inorder()**  **{**  **leaf\_inorder(this->root);**  **}**  ***template* <*typename* *type*>**  ***void* *BST*<*type*>::leaf\_inorder(*BSTNode*<*type*>\* *node*)**  **{**  **if (*node* == nullptr)**  **return;**  **leaf\_inorder(*node*->left);**  **if (*node*->right == nullptr && *node*->left == nullptr) // if leaf node**  **visit(*node*);**    **leaf\_inorder(*node*->right);**  **}** |

### **main.cpp --**

|  |
| --- |
| **#include "bst.h"**  **#include <iostream>**  ***void* clearScreen();**  ***int* main()**  **{**  ***BST*<*int*> bst;**  **while (true)**  **{**  **clearScreen();**  **std::cout << "Binary Search Tree\n";**  **std::cout << "1. insert (part a of question)\n"**  **<< "2. delete by copy\n"**  **<< "3. delete by merge\n"**  **<< "4. search\n"**  **<< "5. inorder display (part b of question)\n"**  **<< "6. sum of non leaf nodes (part c of question)\n"**  **<< "7. display leaf nodes inorder (part d of question)\n"**  **<< "8. empty?\n"**  **<< "9. clear\n"**  **<< "10. Exit\n";**  ***int* choice;**  **std::cout << "Choice: ";**  **std::cin >> choice;**    **switch (choice)**  **{**  **case 1: // Part a of question**  **{**  ***int* element;**  **std::cout << "Enter an element: ";**  **std::cin >> element;**  **bst.insert(element);**  **std::cout << "Inserted\n";**  **}**  **break;**  **case 2:**  **{**  ***int* element;**  **std::cout << "Enter an element to delete: ";**  **std::cin >> element;**  **bst.find\_del\_copy(element);**  **std::cout << "Deleted\n";**  **}**  **break;**  **case 3:**  **{**  ***int* element;**  **std::cout << "Enter an element to delete: ";**  **std::cin >> element;**  **bst.find\_del\_merge(element);**  **std::cout << "Deleted\n";**  **}**  **break;**  **case 4:**  **{**  ***int* element;**  **std::cout << "Enter element to search: ";**  **std::cin >> element;**  **if (bst.search(element) != nullptr)**  **std::cout << "Element found in the tree\n";**  **else**  **std::cout << "Element not found in the tree\n";**  **}**  **break;**  **case 5: // part b of question**  **{**  **if (bst.empty())**  **std::cout << "Tree is empty\n";**  **else**  **{**  **std::cout << "BST:\n";**  **bst.inorder();**  **}**  **}**  **break;**  **case 6: // part c of question**  **{**  **std::cout << "Sum of values of all non-leaf nodes: " << bst.sum\_nonleaf() << "\n";**  **}**  **break;**  **case 7:**  **{**  **if (bst.empty())**  **std::cout << "Tree is empty\n";**  **else**  **{**  **std::cout << "All leaf nodes of tree inorder:\n";**  **bst.leaf\_inorder();**  **}**  **}**  **break;**  **case 8:**  **{**  **if (bst.empty())**  **std::cout << "Tree is empty\n";**  **else**  **std::cout << "Tree is not empty\n";**  **}**  **break;**  **case 9:**  **{**  **bst.clear();**  **std::cout << "Tree cleared\n";**  **}**  **break;**  **case 10:**  **{**  **std::cout << "Exiting...\n";**  **exit(0);**  **}**  **break;**  **default:**  **std::cout << "Invalid option\n";**  **break;**  **}**  **std::cout << "\nPress enter to continue..";**  **std::cin.ignore();**  **std::cin.get();**  **}**  **return 0;**  **}**  ***void* clearScreen()**  **{**  **for (*int* i = 0; i < 160; i++)**  **std::cout << "\n";**  **}** |