|  |  |
| --- | --- |
| **Name** | Shaheena ebadi |
| **ID** | 2232198642 |
| **Section** | 9 |
| **Date** | 23/05/2024 |

|  |
| --- |
| **Code for the given Work (only main.cpp)** |
| #include "binarysearchtree.h"  #include "binarysearchtree.cpp"  #include <iostream>  using namespace std;  void checkEmpty(bool b) {  if (b)  cout << "Tree is Empty" << endl;  else  cout << "Tree is not Empty" << endl;  }  void itemFound(TreeType<int> &tree, int item) {  bool found;  tree.RetrieveItem(item, found);  if (found)  cout << "Item is found" << endl;  else  cout << "Item is not found" << endl;  }  void insertRecursively(TreeType<int> &newTree, int array[], int start, int end) {  int mid = (start + end) / 2;  newTree.InsertItem(array[mid]);  if (start == end)  return;  if (start < mid)  insertRecursively(newTree, array, start, mid - 1);  insertRecursively(newTree, array, mid + 1, end);  }  int main() {  // Initialize an empty tree object  TreeType<int> bst;  // Print if the tree is empty  int items[] = {10, 23, 6, 13, 1, 40, 31, 5, 4, 25};  for (int i = 0; i < sizeof(items) / sizeof(items[0]); i++) {  bst.InsertItem(items[i]);  }  // Print if the tree is empty (after insertions)  if (!bst.IsEmpty()) {  cout << "Tree is not empty" << endl;  }  // Print the length of the tree  cout << "Length of the tree: " << bst.LengthIs() << endl;  // Retrieve 40 and print whether found or not  int value = 40;  bool found;  bst.RetrieveItem(value, found);  cout << "Item " << value << (found ? " is found" : " is not found") << endl;  // Retrieve 2 and print whether found or not  value = 2;  bst.RetrieveItem(value, found);  cout << "Item " << value << (found ? " is found" : " is not found") << endl;  // Print the elements in the tree (inorder)  cout << "Inorder traversal: ";  bst.ResetTree(IN\_ORDER);  bool finished = false;  while (!finished) {  int item;  bst.GetNextItem(item, IN\_ORDER, finished);  cout << item << " ";  if (finished) {  cout << endl;  }  }  cout << "Preorder traversal: ";  bst.ResetTree(PRE\_ORDER);  finished = false;  while (!finished) {  int item;  bst.GetNextItem(item, PRE\_ORDER, finished);  cout << item << " ";  if (finished) {  cout << endl;  }  }  cout << "Postorder traversal: ";  bst.ResetTree(POST\_ORDER);  finished = false;  while (!finished) {  int item;  bst.GetNextItem(item, POST\_ORDER, finished);  cout << item << " ";  if (finished) {  cout << endl;  }  }  // Make the tree empty  bst.MakeEmpty();  TreeType<int> sequenceTree;  int num = 10;  int sequence[] = {10 ,23, 6, 13, 1 ,40, 31, 5, 4 ,25};  for (int i = 0; i < num; i++) {  sequenceTree.InsertItem(sequence[i]);  }  int n, arr[num + 10], index = 0;  sequenceTree.ResetTree(IN\_ORDER);  finished = false;  while (!finished) {  sequenceTree.GetNextItem(n, IN\_ORDER, finished);  arr[index++] = n;  }  TreeType<int> newTree;  insertRecursively(newTree, arr, 0, index - 1);  newTree.ResetTree(PRE\_ORDER);  finished= false;  while (!finished) {  newTree.GetNextItem(n, PRE\_ORDER, finished);  cout << n << " ";  }  cout << endl;  return 0;  } |