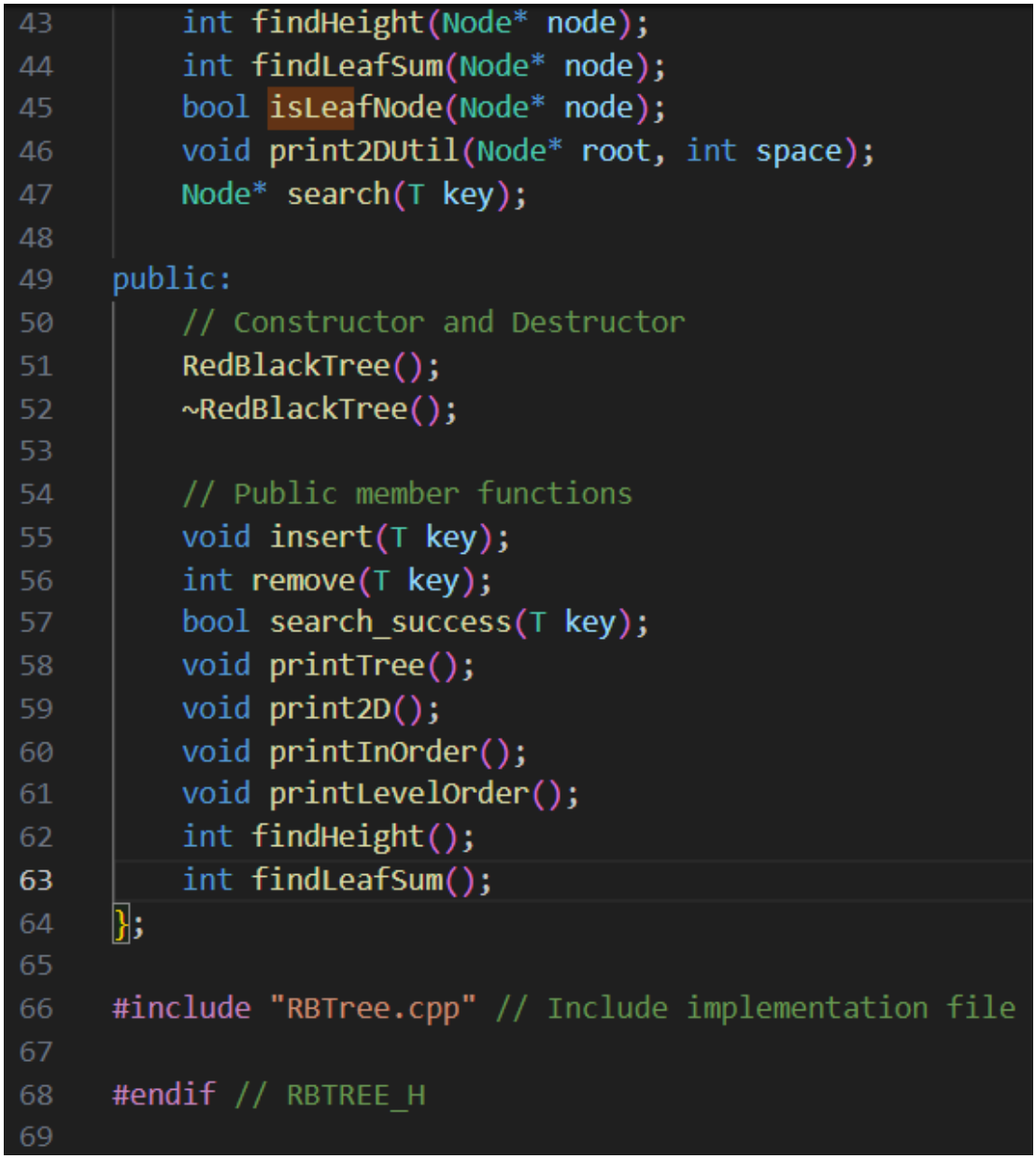
Коды программ

```
#ifndef RBTREE_H
#define RBTREE_H

#include <iostream>
#include <queue>
#include <string>

using namespace std;

// Enumeration for colors of nodes in Red-Black Tree
enum Color { RED, BLACK };

// Class template for Red-Black Tree
template <typename T>
class RedBlackTree {
private:
    // Structure for a node in Red-Black Tree
    struct Node {
        T data;
        Color color;
        Node* parent;
        Node* left;
        Node* right;

        // Constructor to initialize node with data and color
        Node(T value);
    };

    Node* root; // Root of the Red-Black Tree

    // Utility functions
    void rotateLeft(Node*& node);
    void rotateRight(Node*& node);
    void fixInsert(Node*& node);
    void fixDelete(Node*& node);
    Node* minValueNode(Node*& node);
    void transplant(Node*& root, Node*& u, Node*& v);
    void printHelper(Node* root, string indent, bool last);
    void deleteTree(Node* node);
    string nodeToString(Node* node);
    void printInOrder(Node* node);
    void printLevelOrder(Node* node);
```

Рисунок 1 – Фрагмент файла программы RBTree.h (часть 1)



```
43      int findHeight(Node* node);
44      int findLeafSum(Node* node);
45      bool isLeafNode(Node* node);
46      void print2DUtil(Node* root, int space);
47      Node* search(T key);
48
49  public:
50      // Constructor and Destructor
51      RedBlackTree();
52      ~RedBlackTree();
53
54      // Public member functions
55      void insert(T key);
56      int remove(T key);
57      bool search_success(T key);
58      void printTree();
59      void print2D();
60      void printInOrder();
61      void printLevelOrder();
62      int findHeight();
63      int findLeafSum();
64  };
65
66  #include "RBTree.cpp" // Include implementation file
67
68  #endif // RBTREE_H
69
```

Рисунок 2 – Фрагмент файла программы RBTree.h (часть 2)

```cpp
#include "RBTree.h"

// Node constructor
template <typename T>
RedBlackTree<T>::Node::Node(T value)
    : data(value), color(RED), parent(nullptr), left(nullptr), right(nullptr) {}

// Constructor: Initialize Red-Black Tree
template <typename T>
RedBlackTree<T>::RedBlackTree() : root(nullptr) {}

// Destructor: Delete Red-Black Tree
template <typename T>
RedBlackTree<T>::~RedBlackTree() {
    deleteTree(root);
}

// Utility function: Left Rotation
template <typename T>
void RedBlackTree<T>::rotateLeft(Node*& node) {
    Node* child = node->right;
    node->right = child->left;
    if (node->right != nullptr)
        node->right->parent = node;
    child->parent = node->parent;
    if (node->parent == nullptr)
        root = child;
    else if (node == node->parent->left)
        node->parent->left = child;
    else
        node->parent->right = child;
    child->left = node;
    node->parent = child;
}

// Utility function: Right Rotation
template <typename T>
void RedBlackTree<T>::rotateRight(Node*& node) {
    Node* child = node->left;
    node->left = child->right;
    if (node->left != nullptr)
        node->left->parent = node;
    child->parent = node->parent;
    if (node->parent == nullptr)
        root = child;
    else if (node == node->parent->left)
        node->parent->left = child;
    else
        node->parent->right = child;
    child->right = node;
    node->parent = child;
```

Рисунок 3 – Фрагмент файла программы RBTree.cpp (часть 1)

```
}

// Utility function: Fixing Insertion Violation
template <typename T>
void RedBlackTree<T>::fixInsert(Node*& node) {
    Node* parent = nullptr;
    Node* grandparent = nullptr;
    while (node != root && node->color == RED && node->parent->color == RED) {
        parent = node->parent;
        grandparent = parent->parent;
        if (parent == grandparent->left) {
            Node* uncle = grandparent->right;
            if (uncle != nullptr && uncle->color == RED) {
                grandparent->color = RED;
                parent->color = BLACK;
                uncle->color = BLACK;
                node = grandparent;
            } else {
                if (node == parent->right) {
                    rotateLeft(parent);
                    node = parent;
                    parent = node->parent;
                }
                rotateRight(grandparent);
                swap(parent->color, grandparent->color);
                node = parent;
            }
        } else {
            Node* uncle = grandparent->left;
            if (uncle != nullptr && uncle->color == RED) {
                grandparent->color = RED;
                parent->color = BLACK;
                uncle->color = BLACK;
                node = grandparent;
            } else {
                if (node == parent->left) {
                    rotateRight(parent);
                    node = parent;
                    parent = node->parent;
                }
                rotateLeft(grandparent);
                swap(parent->color, grandparent->color);
                node = parent;
            }
        }
    }
    root->color = BLACK;
}

// Utility function: Fixing Deletion Violation
```

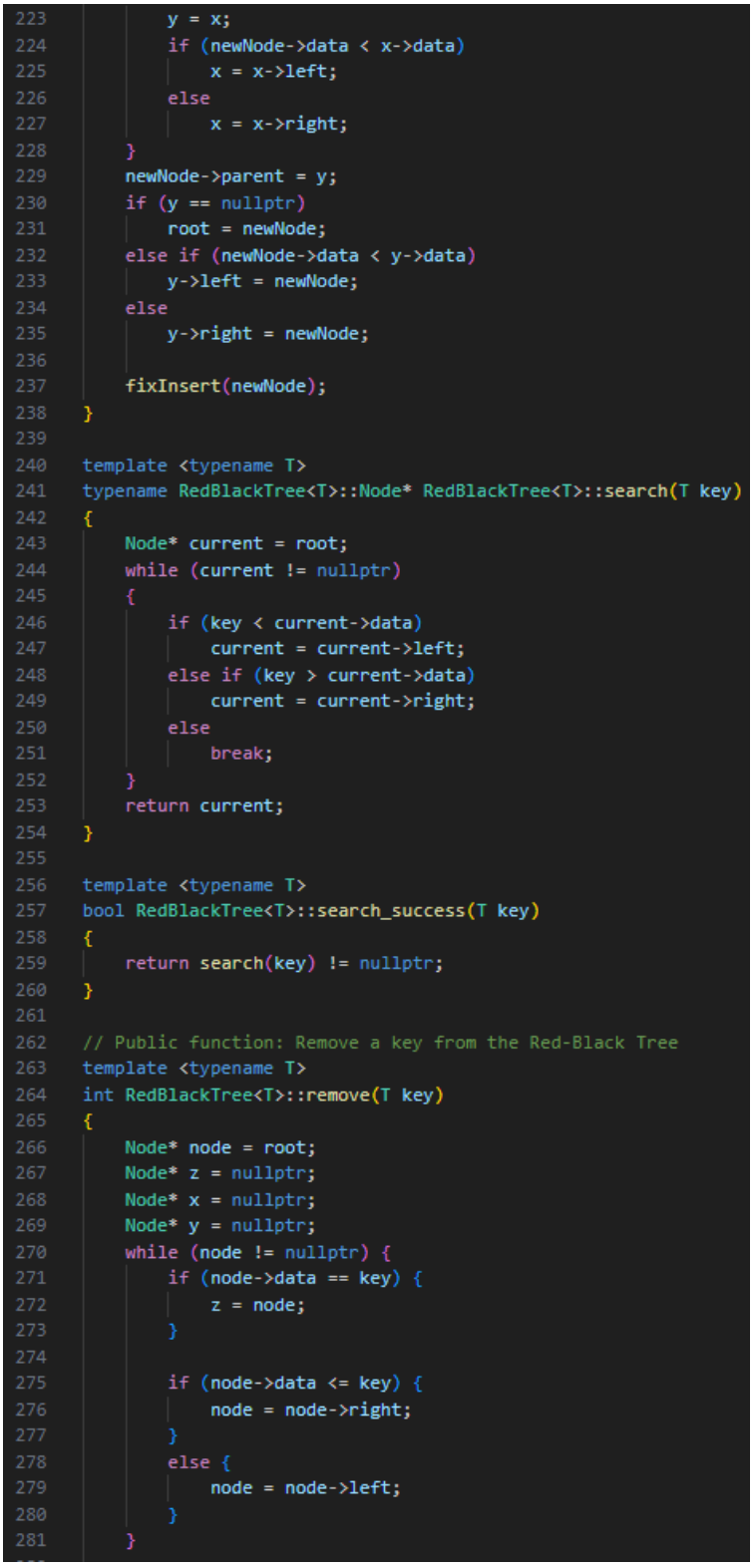
Рисунок 4 – Фрагмент файла программы RBTree.cpp (часть 2)

```cpp
// Utility function: Fixing Deletion Violation
template <typename T>
void RedBlackTree<T>::fixDelete(Node*& node) {
    while (node != root && node->color == BLACK) {
        if (node == node->parent->left) {
            Node* sibling = node->parent->right;
            if (sibling->color == RED) {
                sibling->color = BLACK;
                node->parent->color = RED;
                rotateLeft(node->parent);
                sibling = node->parent->right;
            }
            if ((sibling->left == nullptr || sibling->left->color == BLACK) &&
                (sibling->right == nullptr || sibling->right->color == BLACK)) {
                sibling->color = RED;
                node = node->parent;
            } else {
                if (sibling->right == nullptr || sibling->right->color == BLACK) {
                    if (sibling->left != nullptr)
                        sibling->left->color = BLACK;
                    sibling->color = RED;
                    rotateRight(sibling);
                    sibling = node->parent->right;
                }
                sibling->color = node->parent->color;
                node->parent->color = BLACK;
                if (sibling->right != nullptr)
                    sibling->right->color = BLACK;
                rotateLeft(node->parent);
                node = root;
            }
        } else {
            Node* sibling = node->parent->left;
            if (sibling->color == RED) {
                sibling->color = BLACK;
                node->parent->color = RED;
                rotateRight(node->parent);
                sibling = node->parent->left;
            }
            if ((sibling->left == nullptr || sibling->left->color == BLACK) &&
                (sibling->right == nullptr || sibling->right->color == BLACK)) {
                sibling->color = RED;
                node = node->parent;
            } else {
                if (sibling->left == nullptr || sibling->left->color == BLACK) {
                    if (sibling->right != nullptr)
                        sibling->right->color = BLACK;
                    sibling->color = RED;
                    rotateLeft(sibling);
                    sibling = node->parent->left;
                }
                sibling->color = node->parent->color;
                node->parent->color = BLACK;
                if (sibling->left != nullptr)
                    sibling->left->color = BLACK;
                rotateRight(node->parent);
                node = root;
            }
        }
    }
    node->color = BLACK;
```

Рисунок 5 – Фрагмент файла программы RBTree.cpp (часть 3)

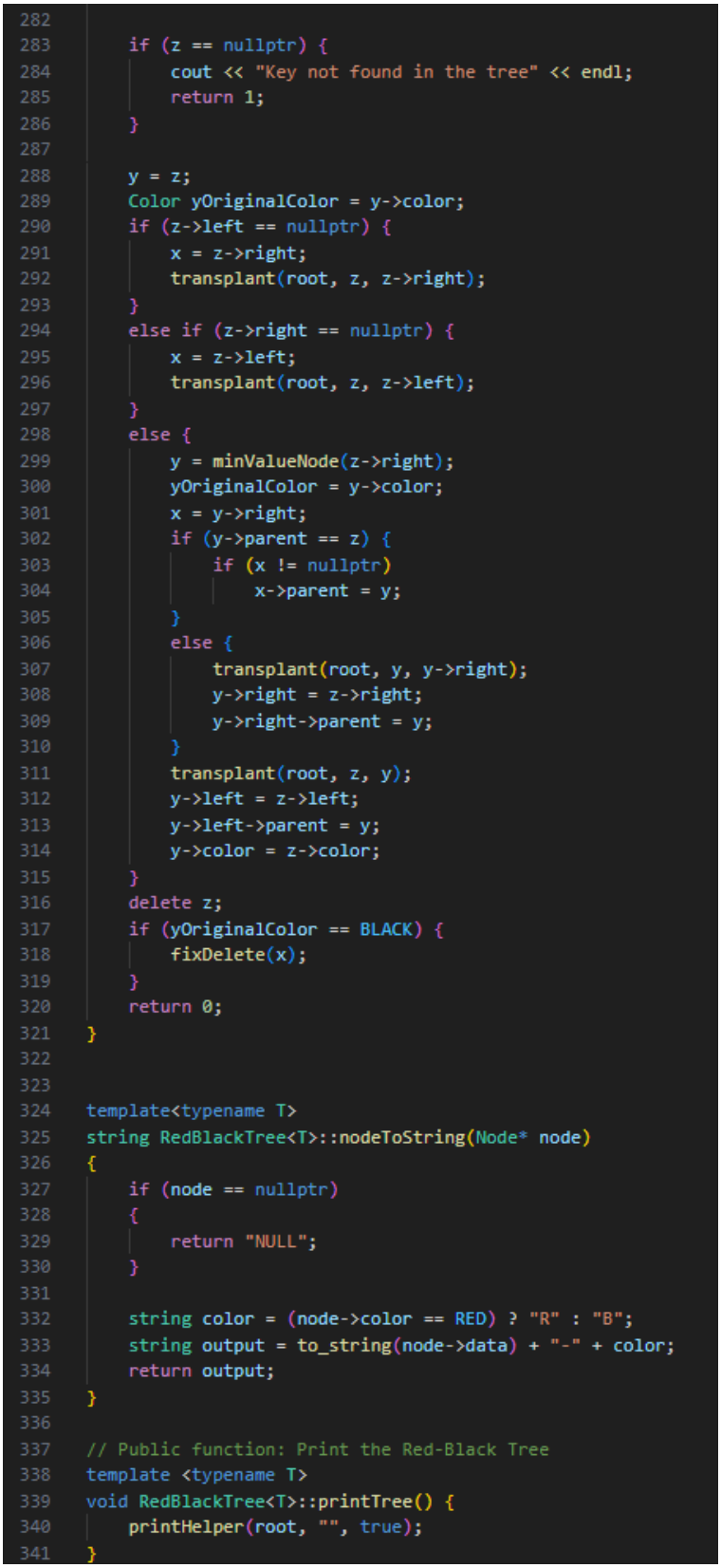
```
}

// Utility function: Find Node with Minimum Value
template <typename T>
typename RedBlackTree<T>::Node* RedBlackTree<T>::minValueNode(Node*& node) {
    Node* current = node;
    while (current->left != nullptr)
        current = current->left;
    return current;
}

// Utility function: Transplant nodes in Red-Black Tree
template <typename T>
void RedBlackTree<T>::transplant(Node*& root, Node*& u, Node*& v) {
    if (u->parent == nullptr)
        root = v;
    else if (u == u->parent->left)
        u->parent->left = v;
    else
        u->parent->right = v;
    if (v != nullptr)
        v->parent = u->parent;
}

// Utility function: Helper to print Red-Black Tree
template <typename T>
void RedBlackTree<T>::printHelper(Node* root, string indent, bool last) {
    if (root != nullptr) {
        cout << indent;
        if (last) {
            cout << "R----";
            indent += "   ";
        } else {
            cout << "L----";
            indent += "|  ";
        }
        string sColor = (root->color == RED) ? "RED" : "BLACK";
        cout << root->data << "(" << sColor << ")" << endl;
        printHelper(root->left, indent, false);
        printHelper(root->right, indent, true);
    }
}

// Utility function: Delete all nodes in the Red-Black Tree
template <typename T>
void RedBlackTree<T>::deleteTree(Node* node) {
    if (node != nullptr) {
        deleteTree(node->left);
        deleteTree(node->right);
        delete node;
    }
}

// Public function: Insert a new key into the Red-Black Tree
template <typename T>
void RedBlackTree<T>::insert(T key) {
    Node* newNode = new Node(key);
    Node* y = nullptr;
    Node* x = root;

    while (x != nullptr) {
```

Рисунок 6 – Фрагмент файла программы RBTree.cpp (часть 4)



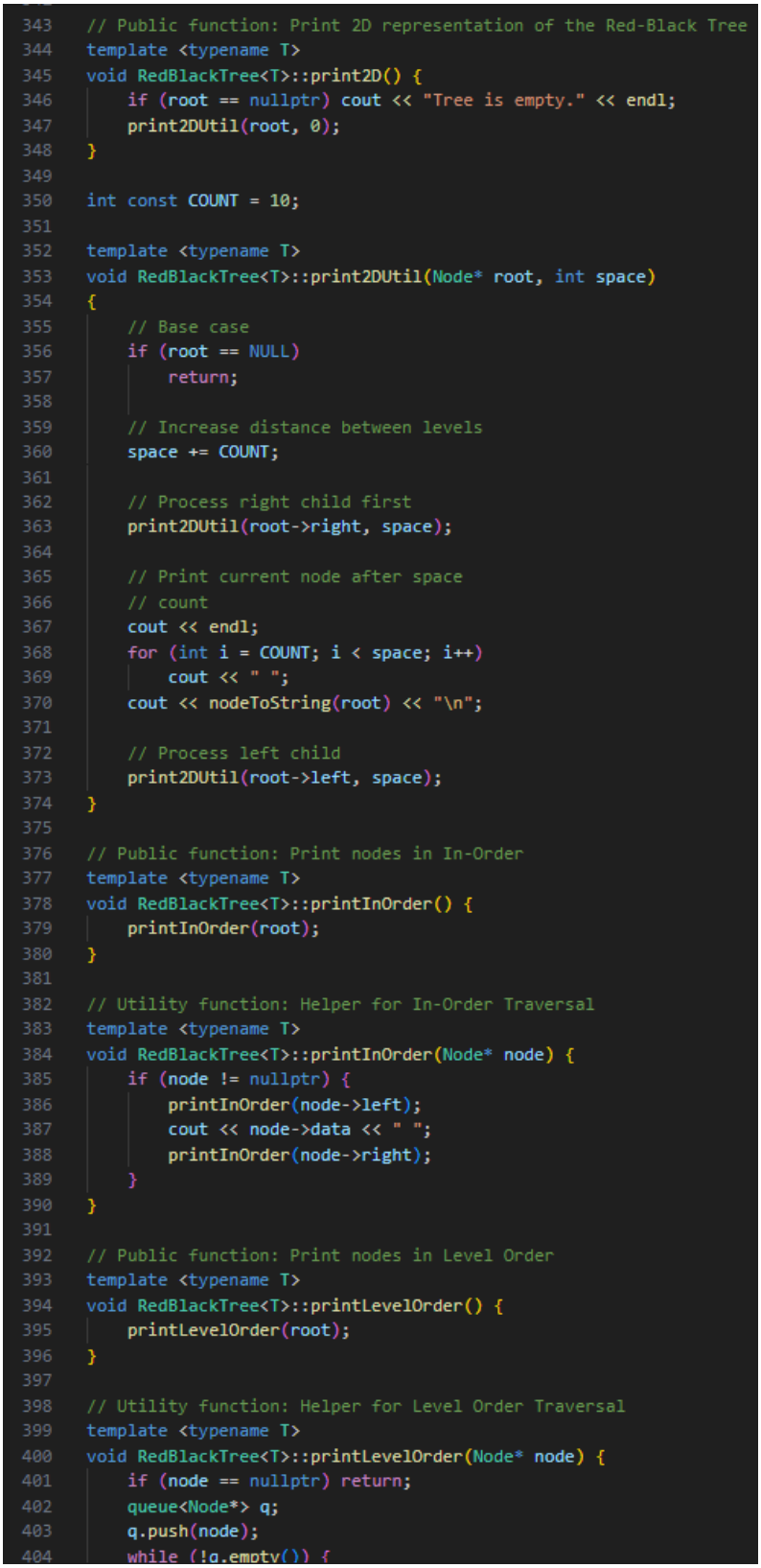
```
223         y = x;
224         if (newNode->data < x->data)
225             x = x->left;
226         else
227             x = x->right;
228     }
229     newNode->parent = y;
230     if (y == nullptr)
231         root = newNode;
232     else if (newNode->data < y->data)
233         y->left = newNode;
234     else
235         y->right = newNode;
236 
237     fixInsert(newNode);
238 }
239 
240 template <typename T>
241 typename RedBlackTree<T>::Node* RedBlackTree<T>::search(T key)
242 {
243     Node* current = root;
244     while (current != nullptr)
245     {
246         if (key < current->data)
247             current = current->left;
248         else if (key > current->data)
249             current = current->right;
250         else
251             break;
252     }
253     return current;
254 }
255 
256 template <typename T>
257 bool RedBlackTree<T>::search_success(T key)
258 {
259     return search(key) != nullptr;
260 }
261 
262 // Public function: Remove a key from the Red-Black Tree
263 template <typename T>
264 int RedBlackTree<T>::remove(T key)
265 {
266     Node* node = root;
267     Node* z = nullptr;
268     Node* x = nullptr;
269     Node* y = nullptr;
270     while (node != nullptr) {
271         if (node->data == key) {
272             z = node;
273         }
274 
275         if (node->data <= key) {
276             node = node->right;
277         }
278         else {
279             node = node->left;
280         }
281     }
```

Рисунок 7 – Фрагмент файла программы RBTree.cpp (часть 5)



```
282
283     if (z == nullptr) {
284         cout << "Key not found in the tree" << endl;
285         return 1;
286     }
287
288     y = z;
289     Color yOriginalColor = y->color;
290     if (z->left == nullptr) {
291         x = z->right;
292         transplant(root, z, z->right);
293     }
294     else if (z->right == nullptr) {
295         x = z->left;
296         transplant(root, z, z->left);
297     }
298     else {
299         y = minValueNode(z->right);
300         yOriginalColor = y->color;
301         x = y->right;
302         if (y->parent == z) {
303             if (x != nullptr)
304                 x->parent = y;
305         }
306         else {
307             transplant(root, y, y->right);
308             y->right = z->right;
309             y->right->parent = y;
310         }
311         transplant(root, z, y);
312         y->left = z->left;
313         y->left->parent = y;
314         y->color = z->color;
315     }
316     delete z;
317     if (yOriginalColor == BLACK) {
318         fixDelete(x);
319     }
320     return 0;
321 }
322
323
324 template<typename T>
325 string RedBlackTree<T>::nodeToString(Node* node)
326 {
327     if (node == nullptr)
328     {
329         return "NULL";
330     }
331
332     string color = (node->color == RED) ? "R" : "B";
333     string output = to_string(node->data) + "-" + color;
334     return output;
335 }
336
337 // Public function: Print the Red-Black Tree
338 template <typename T>
339 void RedBlackTree<T>::printTree() {
340     printHelper(root, "", true);
341 }
```

Рисунок 8 – Фрагмент файла программы RBTree.cpp (часть 6)



```
343  // Public function: Print 2D representation of the Red-Black Tree
344  template <typename T>
345  void RedBlackTree<T>::print2D() {
346      if (root == nullptr) cout << "Tree is empty." << endl;
347      print2DUtil(root, 0);
348  }
349
350  int const COUNT = 10;
351
352  template <typename T>
353  void RedBlackTree<T>::print2DUtil(Node* root, int space)
354  {
355      // Base case
356      if (root == NULL)
357          return;
358
359      // Increase distance between levels
360      space += COUNT;
361
362      // Process right child first
363      print2DUtil(root->right, space);
364
365      // Print current node after space
366      // count
367      cout << endl;
368      for (int i = COUNT; i < space; i++)
369          cout << " ";
370      cout << nodeToString(root) << "\n";
371
372      // Process left child
373      print2DUtil(root->left, space);
374  }
375
376  // Public function: Print nodes in In-Order
377  template <typename T>
378  void RedBlackTree<T>::printInOrder() {
379      printInOrder(root);
380  }
381
382  // Utility function: Helper for In-Order Traversal
383  template <typename T>
384  void RedBlackTree<T>::printInOrder(Node* node) {
385      if (node != nullptr) {
386          printInOrder(node->left);
387          cout << node->data << " ";
388          printInOrder(node->right);
389      }
390  }
391
392  // Public function: Print nodes in Level Order
393  template <typename T>
394  void RedBlackTree<T>::printLevelOrder() {
395      printLevelOrder(root);
396  }
397
398  // Utility function: Helper for Level Order Traversal
399  template <typename T>
400  void RedBlackTree<T>::printLevelOrder(Node* node) {
401      if (node == nullptr) return;
402      queue<Node*> q;
403      q.push(node);
404      while (!q.empty()) {
```

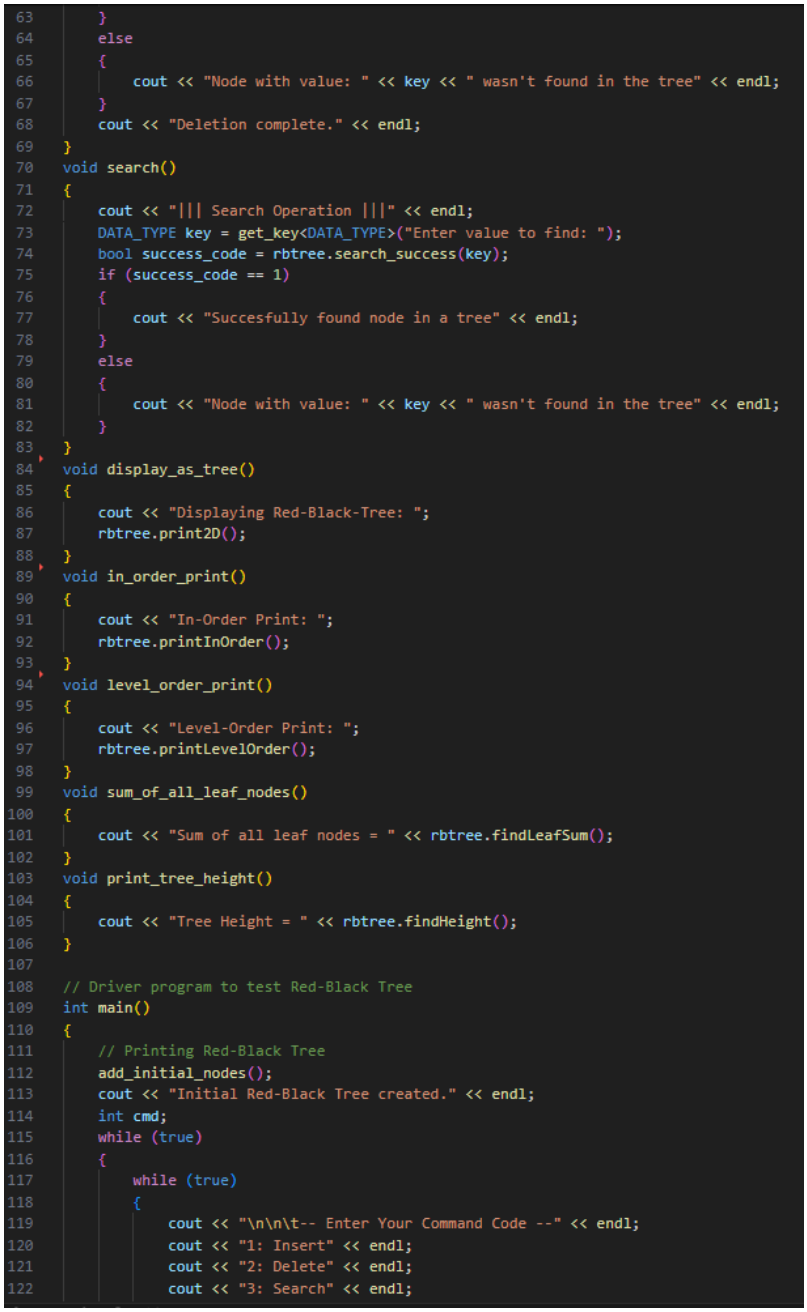
Рисунок 9 – Фрагмент файла программы RBTree.cpp (часть 7)

```
404     while (!q.empty()) {
405         Node* temp = q.front();
406         cout << temp->data << " ";
407         q.pop();
408         if (temp->left != nullptr) q.push(temp->left);
409         if (temp->right != nullptr) q.push(temp->right);
410     }
411 }
412 
413 // Public function: Find height of the tree
414 template <typename T>
415 int RedBlackTree<T>::findHeight() {
416     return findHeight(root);
417 }
418 
419 // Utility function: Helper for height calculation
420 template <typename T>
421 int RedBlackTree<T>::findHeight(Node* node) {
422     if (node == nullptr) return -1;
423     int leftHeight = findHeight(node->left);
424     int rightHeight = findHeight(node->right);
425     return max(leftHeight, rightHeight) + 1;
426 }
427 
428 // Public function: Find sum of all leaf nodes
429 template <typename T>
430 int RedBlackTree<T>::findLeafSum() {
431     return findLeafSum(root);
432 }
433 
434 // Utility function: Helper for finding sum of leaf nodes
435 template <typename T>
436 int RedBlackTree<T>::findLeafSum(Node* node) {
437     if (node == nullptr) return 0;
438     if (isLeafNode(node)) return node->data;
439     return findLeafSum(node->left) + findLeafSum(node->right);
440 }
441 
442 // Utility function: Check if a node is a leaf node
443 template <typename T>
444 bool RedBlackTree<T>::isLeafNode(Node* node) {
445     return node != nullptr && node->left == nullptr && node->right == nullptr;
446 }
```

Рисунок 10 – Фрагмент файла программы RBTree.cpp (часть 8)

```cpp
#include "RBTree.h"
#include <limits>

// Defining what type of data is in nodes of the tree
#define DATA_TYPE int

// Initializing empty Red-Black Tree
RedBlackTree<DATA_TYPE> rbtree;

// Utility function for correctly clearing input stream
void clear_cin()
{
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
void add_initial_nodes()
{
    rbtree.insert(13);
    rbtree.insert(8);
    rbtree.insert(17);
    rbtree.insert(1);
    rbtree.insert(11);
    rbtree.insert(15);
    rbtree.insert(25);
    rbtree.insert(6);
    rbtree.insert(22);
    rbtree.insert(27);
}

template <typename T>
T get_key(const string& prompt)
{
    T key;
    while (true)
    {
        cout << prompt;
        if (cin >> key)
        {
            clear_cin();
            return key;
        }
        clear_cin();
        cout << "Invalid input. Please try again.\n";
    }
}

void insert()
{
    cout << "||| Inserting Operation |||" << endl;
    DATA_TYPE key = get_key<DATA_TYPE>("Enter value to insert: ");
    rbtree.insert(key);
    cout << "Insertion complete." << endl;
}

void del()
{
    cout << "||| Deletion Operation |||" << endl;
    DATA_TYPE key = get_key<DATA_TYPE>("Enter value to delete: ");
    int completion_code = rbtree.remove(key);
    if (completion_code == 0)
    {
        cout << "Succesfully deleted node with value: " << key << endl;
```

Рисунок 11 – Фрагмент файла программы main.cpp (часть 1)



```
63      }
64      else
65      {
66          cout << "Node with value: " << key << " wasn't found in the tree" << endl;
67      }
68      cout << "Deletion complete." << endl;
69  }
70  void search()
71  {
72      cout << "||| Search Operation |||" << endl;
73      DATA_TYPE key = get_key<DATA_TYPE>("Enter value to find: ");
74      bool success_code = rbtree.search_success(key);
75      if (success_code == 1)
76      {
77          cout << "Succesfully found node in a tree" << endl;
78      }
79      else
80      {
81          cout << "Node with value: " << key << " wasn't found in the tree" << endl;
82      }
83  }
84  void display_as_tree()
85  {
86      cout << "Displaying Red-Black-Tree: ";
87      rbtree.print2D();
88  }
89  void in_order_print()
90  {
91      cout << "In-Order Print: ";
92      rbtree.printInOrder();
93  }
94  void level_order_print()
95  {
96      cout << "Level-Order Print: ";
97      rbtree.printLevelOrder();
98  }
99  void sum_of_all_leaf_nodes()
100 {
101     cout << "Sum of all leaf nodes = " << rbtree.findLeafSum();
102 }
103 void print_tree_height()
104 {
105     cout << "Tree Height = " << rbtree.findHeight();
106 }
107
108 // Driver program to test Red-Black Tree
109 int main()
110 {
111     // Printing Red-Black Tree
112     add_initial_nodes();
113     cout << "Initial Red-Black Tree created." << endl;
114     int cmd;
115     while (true)
116     {
117         while (true)
118         {
119             cout << "\n\n\t-- Enter Your Command Code --" << endl;
120             cout << "1: Insert" << endl;
121             cout << "2: Delete" << endl;
122             cout << "3: Search" << endl;
```

Рисунок 12 – Фрагмент файла программы main.cpp (часть 2)

```
            cout << "4: Display as a tree" << endl;
            cout << "5: In-Order print" << endl;
            cout << "6: Level-Order print" << endl;
            cout << "7: Sum of all leaf nodes" << endl;
            cout << "8: Print Tree Height" << endl;
            cout << "0: Exit" << endl;
            cout << "\tCommand Code: ";
            if (cin >> cmd && cmd >= 0 && cmd <= 8) break;
            clear_cin();
            cout << "Invalid Command Code\n";
        }

        cout << endl;
        switch(cmd)
        {
            case 0:
                cout << "Exiting...";
                return 0;
            case 1:
                insert();
                break;
            case 2:
                del();
                break;
            case 3:
                search();
                break;
            case 4:
                display_as_tree();
                break;
            case 5:
                in_order_print();
                break;
            case 6:
                level_order_print();
                break;
            case 7:
                sum_of_all_leaf_nodes();
                break;
            case 8:
                print_tree_height();
                break;
        }
    }


    return 0;
}
```

Рисунок 13 – Фрагмент файла программы main.cpp (часть 3)