Carlos Rangel
1217700029

Homework 4: Submission Template

1. Copy and paste your complete program for question 1 here.

BinarySearchTree.h:

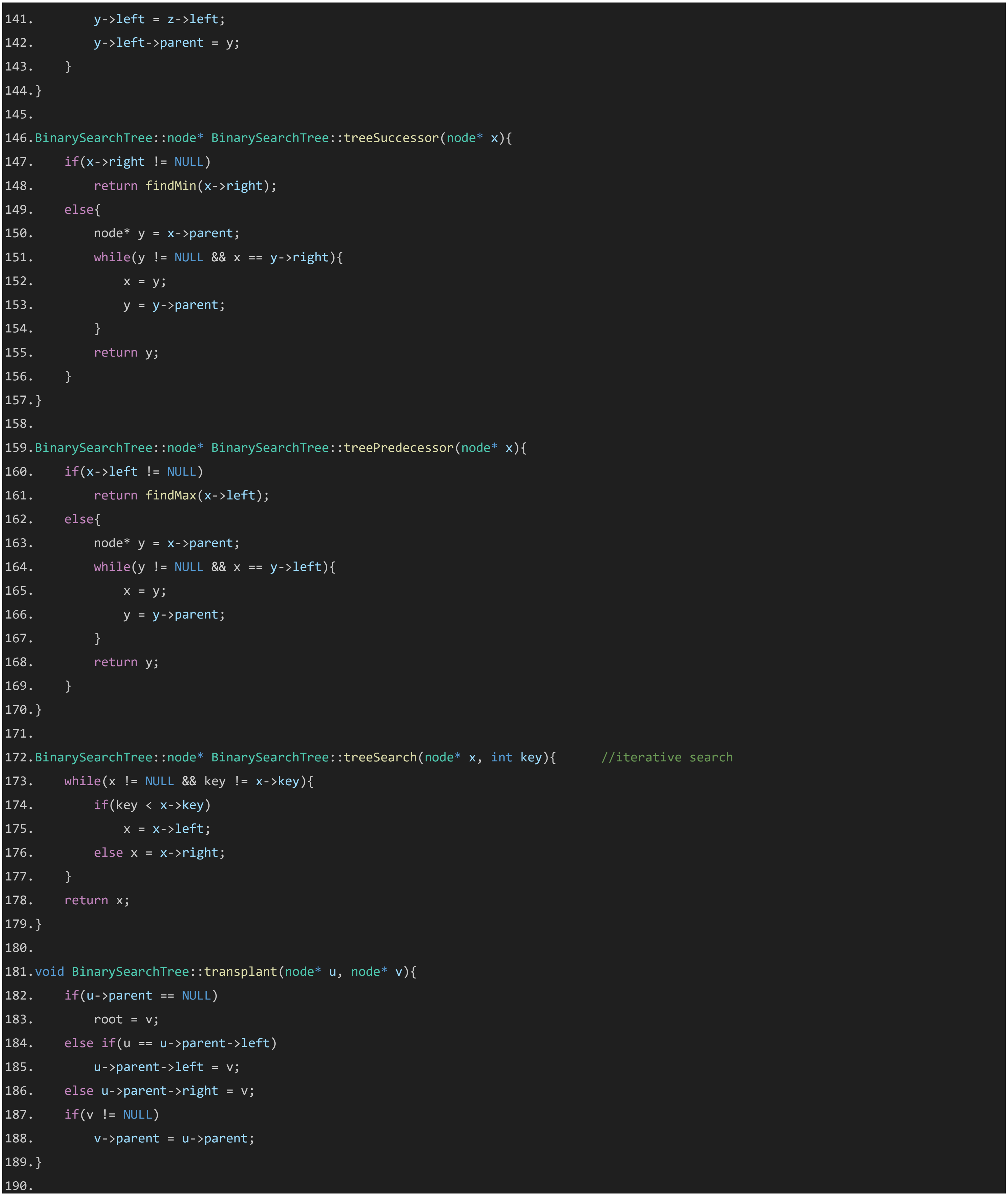
```
#ifndef BINARYSEARCHTREE_H
#define BINARYSEARCHTREE_H

#include <iostream>
#include <cstdlib>
#include <string>
#include <limits>

class BinarySearchTree{
    private:
        class node{
            public:
                node* left;
                node* right;
                node* parent;
                int key;
                std::string data;
        };
        void destroyTree(node* x){
            if(x != NULL){
                destroyTree(x->left);
                destroyTree(x->right);
                delete x;
            }
        }

    public:
        node* root;
        BinarySearchTree();
        ~BinarySearchTree();
        bool isEmpty();
        BinarySearchTree::node* findMin(node* x);
        void removeMin();
        BinarySearchTree::node* findMax(node* x);
        void removeMax();

        void preOrderTreeWalk(node*);
        void inOrderTreeWalk(node*);
        void postOrderTreeWalk(node*);

        void inOrderTreeWalk_flightName(node*);

        void treeInsert(int);
        void treeInsert(int, std::string);
        void treeDelete(node* z);
        BinarySearchTree::node* treeSuccessor(node*);
        BinarySearchTree::node* treePredecessor(node*);
        BinarySearchTree::node* treeSearch(node* x, int key);
        void transplant(node* u, node* v);

};
#endif // BINARYSEARCHTREE_H
```

BinarySearchTree.cpp:

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <limits>

//header file for methods
#include "BinarySearchTree.h"
////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////

//constructor
BinarySearchTree::BinarySearchTree(){
    root = NULL;
}
//destructor
BinarySearchTree::~BinarySearchTree(){
    destroyTree(root);
}

//////////////////////////////////////////////////////////////////////////////
//checking da tree and removing min/max
bool BinarySearchTree::isEmpty(){
    return root == NULL;
}

BinarySearchTree::node* BinarySearchTree::findMin(node* x){
    while(x->left != NULL)
        x = x->left;
    return x;
}
BinarySearchTree::node* BinarySearchTree::findMax(node* x){
    while(x->right != NULL)
        x = x->right;
    return x;
}

void BinarySearchTree::removeMin(){
    treeDelete(findMin(root));
}
void BinarySearchTree::removeMax(){
    treeDelete(findMax(root));
}

////////////////////////////////////////////////////////////////////////////////
//tree walks
void BinarySearchTree::preOrderTreeWalk(node* x){
    if(x != NULL){
        std::cout << " " << x->key << " ";
        if(x->left) inOrderTreeWalk(x->left);
        if(x->right)inOrderTreeWalk(x->right);
    }
}
void BinarySearchTree::inOrderTreeWalk(node* x){
    if(x != NULL){
        if(x->left) inOrderTreeWalk(x->left);
        std::cout << " " << x->key << " ";
        if(x->right)inOrderTreeWalk(x->right);
    }
}
void BinarySearchTree::postOrderTreeWalk(node* x){
    if(x != NULL){
        if(x->left) inOrderTreeWalk(x->left);
        if(x->right)inOrderTreeWalk(x->right);
        std::cout << " " << x->key << " ";
    }
}

//secondary inorder tree walk to be more verbose with flight manager.
void BinarySearchTree::inOrderTreeWalk_flightName(node* x){
    if(x != NULL){
        if(x->left) inOrderTreeWalk_flightName(x->left);
        std::cout << " Flight " << x->data << "\t::\tlanding in " << x->key << " minutes.\n";
        if(x->right)inOrderTreeWalk_flightName(x->right);
    }
}
////////////////////////////////////////////////////////////////////////////////
//insert, delete, and dictionary ops (with helper functions)
void BinarySearchTree::treeInsert(int key){
    node* z = new node();
    z->key = key;
    z->left = NULL;
    z->right = NULL;
    z->parent = NULL;

    node* y = NULL;
    node* x = root;

    while(x != NULL){
        y = x;
        if(z->key < x->key)
```

```
            x = x->left;
        else x = x->right;
    }
    z->parent = y;
    if(y == NULL)
        root = z;
    else if(z->key < y->key)
        y->left = z;
    else y->right = z;
}

//overload to include string input
void BinarySearchTree::treeInsert(int key, std::string data){
    node* z = new node();
    z->key = key;
    z->data = data;
    z->left = NULL;
    z->right = NULL;
    z->parent = NULL;

    node* y = NULL;
    node* x = root;

    while(x != NULL){
        y = x;
        if(z->key < x->key)
            x = x->left;
        else x = x->right;
    }
    z->parent = y;
    if(y == NULL)
        root = z;
    else if(z->key < y->key)
        y->left = z;
    else y->right = z;
}

void BinarySearchTree::treeDelete(node* z){
    if(z->left == NULL)
        transplant(z, z->right);
    else if(z->right == NULL)
        transplant(z, z->left);
    else {
        node* y = findMin(z->right);
        if(y != z->right){
            transplant(y, y->right);
            y->right = z->right;
            y->right->parent = y;
        }
        transplant(z, y);
```

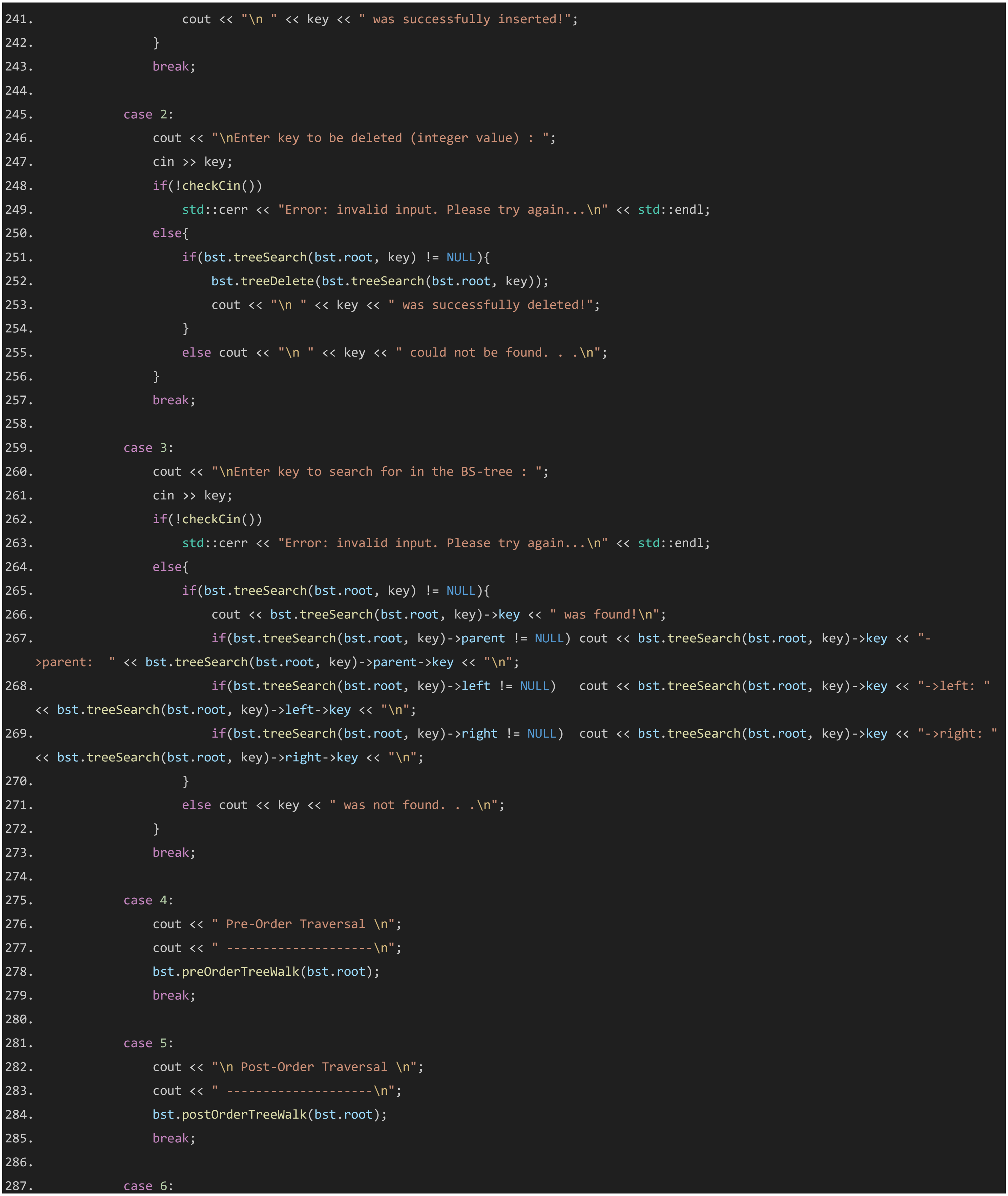
Carlos Rangel
1217700029



```
        y->left = z->left;
        y->left->parent = y;
    }
}

BinarySearchTree::node* BinarySearchTree::treeSuccessor(node* x){
    if(x->right != NULL)
        return findMin(x->right);
    else{
        node* y = x->parent;
        while(y != NULL && x == y->right){
            x = y;
            y = y->parent;
        }
        return y;
    }
}

BinarySearchTree::node* BinarySearchTree::treePredecessor(node* x){
    if(x->left != NULL)
        return findMax(x->left);
    else{
        node* y = x->parent;
        while(y != NULL && x == y->left){
            x = y;
            y = y->parent;
        }
        return y;
    }
}

BinarySearchTree::node* BinarySearchTree::treeSearch(node* x, int key){      //iterative search
    while(x != NULL && key != x->key){
        if(key < x->key)
            x = x->left;
        else x = x->right;
    }
    return x;
}

void BinarySearchTree::transplant(node* u, node* v){
    if(u->parent == NULL)
        root = v;
    else if(u == u->parent->left)
        u->parent->left = v;
    else u->parent->right = v;
    if(v != NULL)
        v->parent = u->parent;
}

```

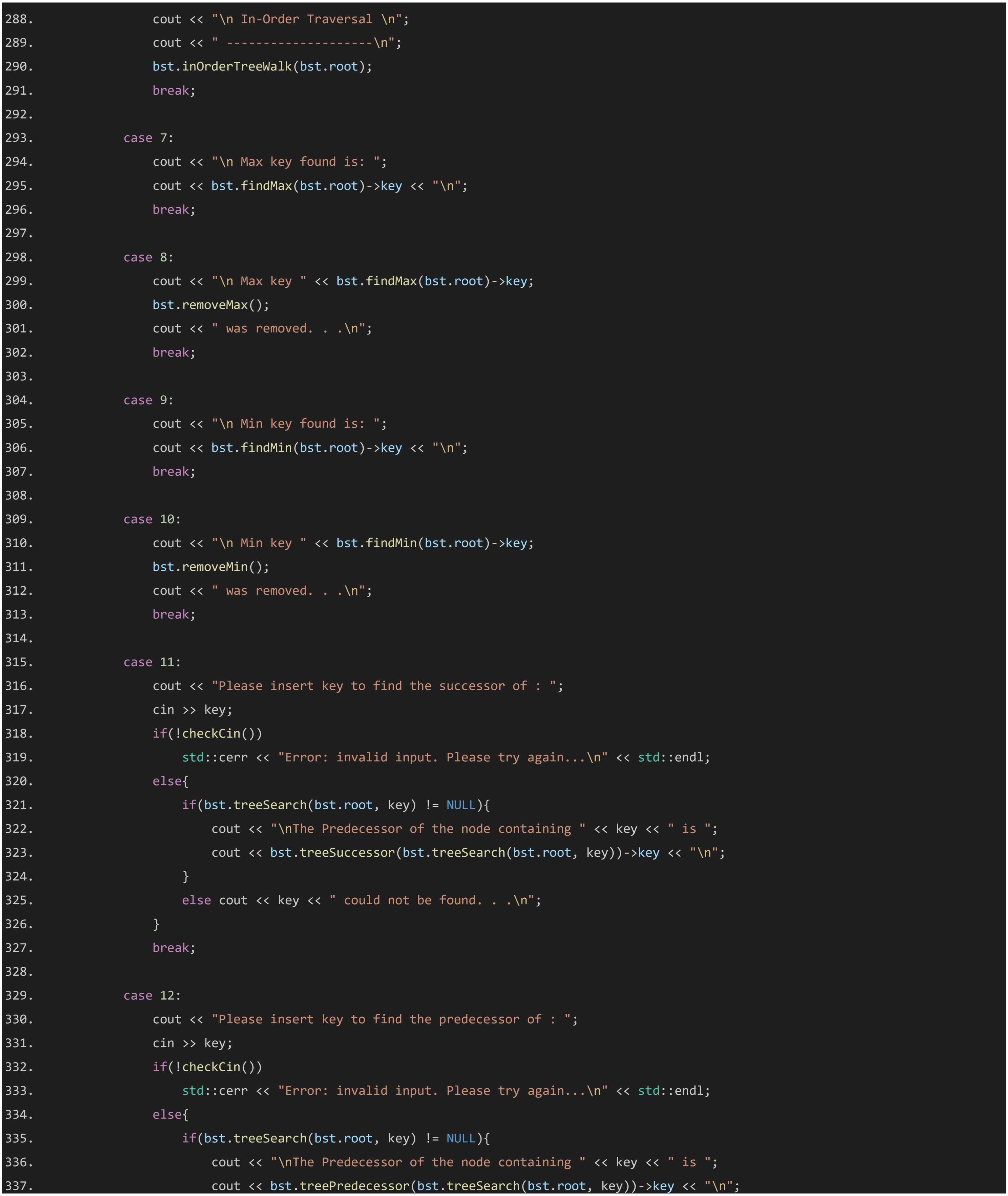
Carlos Rangel
1217700029

```cpp
//helper function to validate user input
bool checkCin(){
    if(std::cin.fail()){
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::min(), '\n');

        return false;
    }
    return true;
}
//////////////////////////////////////////////////////////////////////////////
//
//COMMENTED OUT FOR LINKING PURPOSES
//
//////////////////////////////////////////////////////////////////////////////
///* <= remove to test methods.
int main(){
    using namespace std;

    BinarySearchTree bst;
    int choice, key;

    while(true){
        cout << endl << "\n";
        cout <<  "Binary Search Tree Example \n";
        cout << " ----------------------------- \n";
        cout << " 1.  Insert a Node \n";
        cout << " 2.  Delete a Node\n";
        cout << " 3.  Search for a key\n";
        cout << " 4.  Pre-Order Traversal \n";
        cout << " 5.  Post-Order Traversal \n";
        cout << " 6.  In-Order Traversal\n";
        cout << " 7.  Find Max\n";
        cout << " 8.  Remove Max\n";
        cout << " 9.  Find Min\n";
        cout << " 10. Remove Min\n";
        cout << " 11. Successor\n";
        cout << " 12. Predecessor\n";
        cout << " 13. Exit\n";
        cout << " Enter your choice : \n";
        cin >> choice;

        switch(choice){
            case 1:
                cout << "\nEnter key to be Inserted (integer value) : ";
                cin >> key;
                if(!checkCin())
                    std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
                else{
                    bst.treeInsert(key);
```



```
                    cout << "\n " << key << " was successfully inserted!";
                }
                break;

            case 2:
                cout << "\nEnter key to be deleted (integer value) : ";
                cin >> key;
                if(!checkCin())
                    std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
                else{
                    if(bst.treeSearch(bst.root, key) != NULL){
                        bst.treeDelete(bst.treeSearch(bst.root, key));
                        cout << "\n " << key << " was successfully deleted!";
                    }
                    else cout << "\n " << key << " could not be found. . .\n";
                }
                break;

            case 3:
                cout << "\nEnter key to search for in the BS-tree : ";
                cin >> key;
                if(!checkCin())
                    std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
                else{
                    if(bst.treeSearch(bst.root, key) != NULL){
                        cout << bst.treeSearch(bst.root, key)->key << " was found!\n";
                        if(bst.treeSearch(bst.root, key)->parent != NULL) cout << bst.treeSearch(bst.root, key)->key << "->parent:  " << bst.treeSearch(bst.root, key)->parent->key << "\n";
                        if(bst.treeSearch(bst.root, key)->left != NULL)   cout << bst.treeSearch(bst.root, key)->key << "->left: " << bst.treeSearch(bst.root, key)->left->key << "\n";
                        if(bst.treeSearch(bst.root, key)->right != NULL)  cout << bst.treeSearch(bst.root, key)->key << "->right: " << bst.treeSearch(bst.root, key)->right->key << "\n";
                    }
                    else cout << key << " was not found. . .\n";
                }
                break;

            case 4:
                cout << " Pre-Order Traversal \n";
                cout << " --------------------\n";
                bst.preOrderTreeWalk(bst.root);
                break;

            case 5:
                cout << "\n Post-Order Traversal \n";
                cout << " --------------------\n";
                bst.postOrderTreeWalk(bst.root);
                break;

            case 6:
```

Carlos Rangel
1217700029



```
            cout << "\n In-Order Traversal \n";
            cout << " --------------------\n";
            bst.inOrderTreeWalk(bst.root);
            break;

        case 7:
            cout << "\n Max key found is: ";
            cout << bst.findMax(bst.root)->key << "\n";
            break;

        case 8:
            cout << "\n Max key " << bst.findMax(bst.root)->key;
            bst.removeMax();
            cout << " was removed. . .\n";
            break;

        case 9:
            cout << "\n Min key found is: ";
            cout << bst.findMin(bst.root)->key << "\n";
            break;

        case 10:
            cout << "\n Min key " << bst.findMin(bst.root)->key;
            bst.removeMin();
            cout << " was removed. . .\n";
            break;

        case 11:
            cout << "Please insert key to find the successor of : ";
            cin >> key;
            if(!checkCin())
                std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
            else{
                if(bst.treeSearch(bst.root, key) != NULL){
                    cout << "\nThe Predecessor of the node containing " << key << " is ";
                    cout << bst.treeSuccessor(bst.treeSearch(bst.root, key))->key << "\n";
                }
                else cout << key << " could not be found. . .\n";
            }
            break;

        case 12:
            cout << "Please insert key to find the predecessor of : ";
            cin >> key;
            if(!checkCin())
                std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
            else{
                if(bst.treeSearch(bst.root, key) != NULL){
                    cout << "\nThe Predecessor of the node containing " << key << " is ";
                    cout << bst.treePredecessor(bst.treeSearch(bst.root, key))->key << "\n";
```

Carlos Rangel
1217700029

```
                    }
                    else cout << key << " could not be found. . .\n";
                }
                break;

            case 13:
                system("pause");
                return 0;
                break;

            default:
                cout << "Invalid choice\n";
                break;
        }
    }
}
//*/
```

Carlos Rangel
1217700029

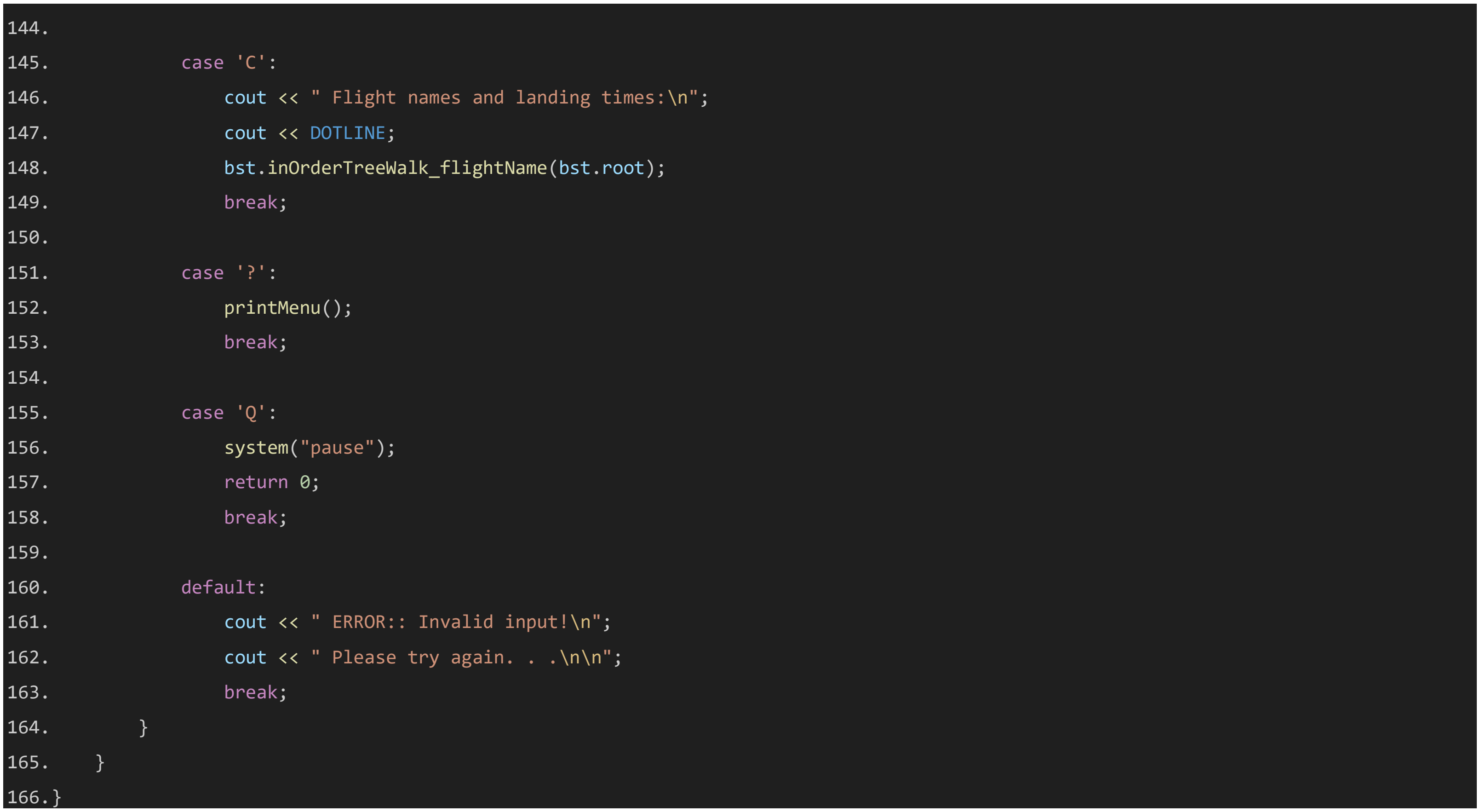
2. Copy and paste your complete program for question 2 here.

```
////////////////////////////////////////////////////////
// Flight management program by Carlos Rangel
// Made for CSE310 to study and implement BSTree
///////////////////////////////////////////////////////

#include <iostream>
#include <cstdlib>
#include <vector>
#include <limits>
#include "BinarySearchTree.h"

#define DOTLINE  " ----------------------------------------\n"

static bool checkCin(){
    if(std::cin.fail()){
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::min(), '\n');

        return false;
    }
    return true;
}

static void printMenu(){
    using namespace std;

    cout << DOTLINE;
    cout << " A. Request Landing\n";
    cout << " B. Withdraw landing request\n";
    cout << " C. List landing times & flight Numbers\n";
    cout << " ?. Print this menu\n";
    cout << " Q. Exit\n\n";

}

int main(){
    using namespace std;

    BinarySearchTree bst;
    string input_s;
    char input;
    int landingTime, timeGap;
    string flightName;

    cout << " Welcome to the Plane Landing System! \n";
    cout << DOTLINE;


    while(true){
        cout << " To begin, please initialize the time gap for landing requests: ";
        cin >> timeGap;
        if(!checkCin()){
            cout << " ERROR:: Invalid input!\n";
            cout << " Please enter an integer to initialize the time gap for landing requests.\n ";
        }
        else break;
    }

    cout << " The inputted time gap is " << timeGap << "\n" << endl;

    while(true){
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

        printMenu();
        cout << " Input: ";
        input = getchar();
        input = toupper(input);


        switch(input){
            //flush cin


            //request a landing
            case 'A':
                cout << " Making a plane landing request. Please enter the following:\n";
                cout << " Plane Flight Number: ";
                cin >> flightName;

                //checking input
                cout << " Landing Time (integers): ";
                while(true){
                    cin >> landingTime;
                    if(!checkCin()){
                        cout << " ERROR:: Invalid input!\n";
                        cout << " Please enter Landing Time (Enter an integer): ";
                    }
                    else break;
                }

                bst.treeInsert(landingTime, flightName);

                //checking for the timeGap (K) constraint.
                if(bst.treeSuccessor(bst.treeSearch(bst.root, landingTime)) != NULL)
                    if( abs(bst.treeSearch(bst.root, landingTime)->key - bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->key) < timeGap){
                        cout << "successor: " << bst.treeSearch(bst.root, landingTime)->key - bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->key << "\n";
                        cout << "\n The time gap between " << landingTime << " and flight ";
```

Carlos Rangel
1217700029

```
                        cout << bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->data << "(" << bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->key << ") is too small!\n";
                        cout << " Plane request denied!\n";
                        bst.treeDelete(bst.treeSearch(bst.root, landingTime));
                        break;
                    }

                if(bst.treePredecessor(bst.treeSearch(bst.root, landingTime)) != NULL)
                    if( abs(bst.treeSearch(bst.root, landingTime)->key - bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->key) < timeGap){
                        cout << "predecessor: " << bst.treeSearch(bst.root, landingTime)->key - bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->key << "\n";;
                        cout << "\n The time gap between " << landingTime << " and flight";
                        cout << bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->data << "(" << bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->key << ") is too small!\n";
                        cout << " Plane request denied!\n";
                        bst.treeDelete(bst.treeSearch(bst.root, landingTime));
                        break;
                    }
                cout << "\n Plane " << bst.treeSearch(bst.root, landingTime)->data << " was added!\n";
                break;

            //withdraw landing request
            //I was gonna do removal by searching landing time
            //but thought that would be weird in a real setting so
            //i overloaded treeSearch to work with inputted strings
            case 'B':
                cout << " Flight names and landing times:\n";
                cout << DOTLINE;
                bst.inOrderTreeWalk_flightName(bst.root);
                cout << " Please enter Landing time of the flight to remove: ";
                while(true){
                        cin >> landingTime;
                        if(!checkCin()){
                            cout << " ERROR:: Invalid input!\n";
                            cout << " Please enter Landing Time (Enter an integer): ";
                        }
                        else break;
                    }

                if(bst.treeSearch(bst.root, landingTime) == NULL){
                    cout << " ERROR:: Flight Not fould!";
                    break;
                }
                else {
                    cout << " Flight " << bst.treeSearch(bst.root, landingTime)->data;
                    bst.treeDelete(bst.treeSearch(bst.root, landingTime));
                    cout << " has been deleted.\n";
                }
                break;
```

Carlos Rangel
1217700029



```cpp

            case 'C':
                cout << " Flight names and landing times:\n";
                cout << DOTLINE;
                bst.inOrderTreeWalk_flightName(bst.root);
                break;

            case '?':
                printMenu();
                break;

            case 'Q':
                system("pause");
                return 0;
                break;

            default:
                cout << " ERROR:: Invalid input!\n";
                cout << " Please try again. . .\n\n";
                break;
        }
    }
}
```

167. Upload your programs for question 1 and question 2 to google drive and provide the links for your programs

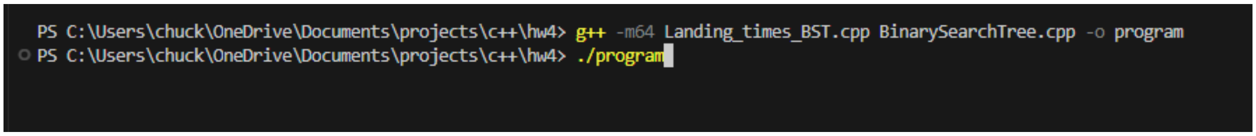
Link for the program for question1

- https://drive.google.com/drive/folders/1eTQNUSk7Abf_Kzu9psHTSCVxtNmHbmQm?usp=sharing

Link for the program for question2

- https://drive.google.com/drive/folders/1z9u2QwJEHkZvqVfBzGQS2nXX_S_5TomL?usp=sharing

I've also provided the executable, as I was having compiling problems myself in case of the need to test. If you compile the .cpp files yourself, please compile using -m64 in gcc. Not sure why it doesn't do it by default.



```
  PS C:\Users\chuck\OneDrive\Documents\projects\c++\hw4> g++ -m64 Landing_times_BST.cpp BinarySearchTree.cpp -o program
○ PS C:\Users\chuck\OneDrive\Documents\projects\c++\hw4> ./program
```

Also, for case A in Landing_time_BST.cpp, the checks for the K time gap are very big and badly formatted in this document sorry!

ALSO ALSO, I decided to get rid of the word margins, as I am assuming you have no intention of printing this page, and I believe it makes the code easier to see. Please let me know if you need a printable version of this document!