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:

1. #include <iostream>
2. #include <cstdlib>
3. #include <vector>
4. #include <limits>
5. //header file for methods
6. #include "BinarySearchTree.h"
7. ///////////////////////////////////////////////////////////////////
8. ////////////////////////////////////////////////////////////////////
9. //constructor
10. BinarySearchTree::BinarySearchTree(){
11. root = NULL;
12. }
13. //destructor
14. BinarySearchTree::~BinarySearchTree(){
15. destroyTree(root);
16. }
17. //////////////////////////////////////////////////////////////////////////////
18. //checking da tree and removing min/max
19. bool BinarySearchTree::isEmpty(){
20. return root == NULL;
21. }
22. BinarySearchTree::node\* BinarySearchTree::findMin(node\* x){
23. while(x->left != NULL)
24. x = x->left;
25. return x;
26. }
27. BinarySearchTree::node\* BinarySearchTree::findMax(node\* x){
28. while(x->right != NULL)
29. x = x->right;
30. return x;
31. }
32. void BinarySearchTree::removeMin(){
33. treeDelete(findMin(root));
34. }
35. void BinarySearchTree::removeMax(){
36. treeDelete(findMax(root));
37. }
38. //////////////////////////////////////////////////////////////////////////////
39. //tree walks
40. void BinarySearchTree::preOrderTreeWalk(node\* x){
41. if(x != NULL){
42. std::cout << " " << x->key << " ";
43. if(x->left) inOrderTreeWalk(x->left);
44. if(x->right)inOrderTreeWalk(x->right);
45. }
46. }
47. void BinarySearchTree::inOrderTreeWalk(node\* x){
48. if(x != NULL){
49. if(x->left) inOrderTreeWalk(x->left);
50. std::cout << " " << x->key << " ";
51. if(x->right)inOrderTreeWalk(x->right);
52. }
53. }
54. void BinarySearchTree::postOrderTreeWalk(node\* x){
55. if(x != NULL){
56. if(x->left) inOrderTreeWalk(x->left);
57. if(x->right)inOrderTreeWalk(x->right);
58. std::cout << " " << x->key << " ";
59. }
60. }
61. //secondary inorder tree walk to be more verbose with flight manager.
62. void BinarySearchTree::inOrderTreeWalk\_flightName(node\* x){
63. if(x != NULL){
64. if(x->left) inOrderTreeWalk\_flightName(x->left);
65. std::cout << " Flight " << x->data << "\t::\tlanding in " << x->key << " minutes.\n";
66. if(x->right)inOrderTreeWalk\_flightName(x->right);
67. }
68. }
69. //////////////////////////////////////////////////////////////////////////////
70. //insert, delete, and dictionary ops (with helper functions)
71. void BinarySearchTree::treeInsert(int key){
72. node\* z = new node();
73. z->key = key;
74. z->left = NULL;
75. z->right = NULL;
76. z->parent = NULL;
77. node\* y = NULL;
78. node\* x = root;
79. while(x != NULL){
80. y = x;
81. if(z->key < x->key)
82. x = x->left;
83. else x = x->right;
84. }
85. z->parent = y;
86. if(y == NULL)
87. root = z;
88. else if(z->key < y->key)
89. y->left = z;
90. else y->right = z;
91. }
92. //overload to include string input
93. void BinarySearchTree::treeInsert(int key, std::string data){
94. node\* z = new node();
95. z->key = key;
96. z->data = data;
97. z->left = NULL;
98. z->right = NULL;
99. z->parent = NULL;
100. node\* y = NULL;
101. node\* x = root;
102. while(x != NULL){
103. y = x;
104. if(z->key < x->key)
105. x = x->left;
106. else x = x->right;
107. }
108. z->parent = y;
109. if(y == NULL)
110. root = z;
111. else if(z->key < y->key)
112. y->left = z;
113. else y->right = z;
114. }
115. void BinarySearchTree::treeDelete(node\* z){
116. if(z->left == NULL)
117. transplant(z, z->right);
118. else if(z->right == NULL)
119. transplant(z, z->left);
120. else {
121. node\* y = findMin(z->right);
122. if(y != z->right){
123. transplant(y, y->right);
124. y->right = z->right;
125. y->right->parent = y;
126. }
127. transplant(z, y);
128. y->left = z->left;
129. y->left->parent = y;
130. }
131. }
132. BinarySearchTree::node\* BinarySearchTree::treeSuccessor(node\* x){
133. if(x->right != NULL)
134. return findMin(x->right);
135. else{
136. node\* y = x->parent;
137. while(y != NULL && x == y->right){
138. x = y;
139. y = y->parent;
140. }
141. return y;
142. }
143. }
144. BinarySearchTree::node\* BinarySearchTree::treePredecessor(node\* x){
145. if(x->left != NULL)
146. return findMax(x->left);
147. else{
148. node\* y = x->parent;
149. while(y != NULL && x == y->left){
150. x = y;
151. y = y->parent;
152. }
153. return y;
154. }
155. }
156. BinarySearchTree::node\* BinarySearchTree::treeSearch(node\* x, int key){      //iterative search
157. while(x != NULL && key != x->key){
158. if(key < x->key)
159. x = x->left;
160. else x = x->right;
161. }
162. return x;
163. }
164. void BinarySearchTree::transplant(node\* u, node\* v){
165. if(u->parent == NULL)
166. root = v;
167. else if(u == u->parent->left)
168. u->parent->left = v;
169. else u->parent->right = v;
170. if(v != NULL)
171. v->parent = u->parent;
172. }
173. //helper function to validate user input
174. bool checkCin(){
175. if(std::cin.fail()){
176. std::cin.clear();
177. std::cin.ignore(std::numeric\_limits<std::streamsize>::min(), '\n');
178. return false;
179. }
180. return true;
181. }
182. //////////////////////////////////////////////////////////////////////////////
183. //
184. //COMMENTED OUT FOR LINKING PURPOSES
185. //
186. //////////////////////////////////////////////////////////////////////////////
187. ///\* <= remove to test methods.
188. int main(){
189. using namespace std;
190. BinarySearchTree bst;
191. int choice, key;
192. while(true){
193. cout << endl << "\n";
194. cout <<  "Binary Search Tree Example \n";
195. cout << " ----------------------------- \n";
196. cout << " 1.  Insert a Node \n";
197. cout << " 2.  Delete a Node\n";
198. cout << " 3.  Search for a key\n";
199. cout << " 4.  Pre-Order Traversal \n";
200. cout << " 5.  Post-Order Traversal \n";
201. cout << " 6.  In-Order Traversal\n";
202. cout << " 7.  Find Max\n";
203. cout << " 8.  Remove Max\n";
204. cout << " 9.  Find Min\n";
205. cout << " 10. Remove Min\n";
206. cout << " 11. Successor\n";
207. cout << " 12. Predecessor\n";
208. cout << " 13. Exit\n";
209. cout << " Enter your choice : \n";
210. cin >> choice;
211. switch(choice){
212. case 1:
213. cout << "\nEnter key to be Inserted (integer value) : ";
214. cin >> key;
215. if(!checkCin())
216. std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
217. else{
218. bst.treeInsert(key);
219. cout << "\n " << key << " was successfully inserted!";
220. }
221. break;
222. case 2:
223. cout << "\nEnter key to be deleted (integer value) : ";
224. cin >> key;
225. if(!checkCin())
226. std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
227. else{
228. if(bst.treeSearch(bst.root, key) != NULL){
229. bst.treeDelete(bst.treeSearch(bst.root, key));
230. cout << "\n " << key << " was successfully deleted!";
231. }
232. else cout << "\n " << key << " could not be found. . .\n";
233. }
234. break;
235. case 3:
236. cout << "\nEnter key to search for in the BS-tree : ";
237. cin >> key;
238. if(!checkCin())
239. std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
240. else{
241. if(bst.treeSearch(bst.root, key) != NULL){
242. cout << bst.treeSearch(bst.root, key)->key << " was found!\n";
243. if(bst.treeSearch(bst.root, key)->parent != NULL) cout << bst.treeSearch(bst.root, key)->key << "->parent:  " << bst.treeSearch(bst.root, key)->parent->key << "\n";
244. if(bst.treeSearch(bst.root, key)->left != NULL)   cout << bst.treeSearch(bst.root, key)->key << "->left: " << bst.treeSearch(bst.root, key)->left->key << "\n";
245. if(bst.treeSearch(bst.root, key)->right != NULL)  cout << bst.treeSearch(bst.root, key)->key << "->right: " << bst.treeSearch(bst.root, key)->right->key << "\n";
246. }
247. else cout << key << " was not found. . .\n";
248. }
249. break;
251. case 4:
252. cout << " Pre-Order Traversal \n";
253. cout << " --------------------\n";
254. bst.preOrderTreeWalk(bst.root);
255. break;
257. case 5:
258. cout << "\n Post-Order Traversal \n";
259. cout << " --------------------\n";
260. bst.postOrderTreeWalk(bst.root);
261. break;
262. case 6:
263. cout << "\n In-Order Traversal \n";
264. cout << " --------------------\n";
265. bst.inOrderTreeWalk(bst.root);
266. break;
267. case 7:
268. cout << "\n Max key found is: ";
269. cout << bst.findMax(bst.root)->key << "\n";
270. break;
271. case 8:
272. cout << "\n Max key " << bst.findMax(bst.root)->key;
273. bst.removeMax();
274. cout << " was removed. . .\n";
275. break;
276. case 9:
277. cout << "\n Min key found is: ";
278. cout << bst.findMin(bst.root)->key << "\n";
279. break;
280. case 10:
281. cout << "\n Min key " << bst.findMin(bst.root)->key;
282. bst.removeMin();
283. cout << " was removed. . .\n";
284. break;
286. case 11:
287. cout << "Please insert key to find the successor of : ";
288. cin >> key;
289. if(!checkCin())
290. std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
291. else{
292. if(bst.treeSearch(bst.root, key) != NULL){
293. cout << "\nThe Predecessor of the node containing " << key << " is ";
294. cout << bst.treeSuccessor(bst.treeSearch(bst.root, key))->key << "\n";
295. }
296. else cout << key << " could not be found. . .\n";
297. }
298. break;
299. case 12:
300. cout << "Please insert key to find the predecessor of : ";
301. cin >> key;
302. if(!checkCin())
303. std::cerr << "Error: invalid input. Please try again...\n" << std::endl;
304. else{
305. if(bst.treeSearch(bst.root, key) != NULL){
306. cout << "\nThe Predecessor of the node containing " << key << " is ";
307. cout << bst.treePredecessor(bst.treeSearch(bst.root, key))->key << "\n";
308. }
309. else cout << key << " could not be found. . .\n";
310. }
311. break;
312. case 13:
313. system("pause");
314. return 0;
315. break;
317. default:
318. cout << "Invalid choice\n";
319. break;
320. }
321. }
322. }
323. //\*/

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

1. ////////////////////////////////////////////////////////
2. // Flight management program by Carlos Rangel
3. // Made for CSE310 to study and implement BSTree
4. ///////////////////////////////////////////////////////
5. #include <iostream>
6. #include <cstdlib>
7. #include <vector>
8. #include <limits>
9. #include "BinarySearchTree.h"
10. #define DOTLINE  " ----------------------------------------\n"
11. static bool checkCin(){
12. if(std::cin.fail()){
13. std::cin.clear();
14. std::cin.ignore(std::numeric\_limits<std::streamsize>::min(), '\n');
15. return false;
16. }
17. return true;
18. }
19. static void printMenu(){
20. using namespace std;
22. cout << DOTLINE;
23. cout << " A. Request Landing\n";
24. cout << " B. Withdraw landing request\n";
25. cout << " C. List landing times & flight Numbers\n";
26. cout << " ?. Print this menu\n";
27. cout << " Q. Exit\n\n";
28. }
29. int main(){
30. using namespace std;
31. BinarySearchTree bst;
32. string input\_s;
33. char input;
34. int landingTime, timeGap;
35. string flightName;
36. cout << " Welcome to the Plane Landing System! \n";
37. cout << DOTLINE;

40. while(true){
41. cout << " To begin, please initialize the time gap for landing requests: ";
42. cin >> timeGap;
43. if(!checkCin()){
44. cout << " ERROR:: Invalid input!\n";
45. cout << " Please enter an integer to initialize the time gap for landing requests.\n ";
46. }
47. else break;
48. }
50. cout << " The inputted time gap is " << timeGap << "\n" << endl;
51. while(true){
52. std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');
53. printMenu();
54. cout << " Input: ";
55. input = getchar();
56. input = toupper(input);
58. switch(input){
59. //flush cin
61. //request a landing
62. case 'A':
63. cout << " Making a plane landing request. Please enter the following:\n";
64. cout << " Plane Flight Number: ";
65. cin >> flightName;
66. //checking input
67. cout << " Landing Time (integers): ";
68. while(true){
69. cin >> landingTime;
70. if(!checkCin()){
71. cout << " ERROR:: Invalid input!\n";
72. cout << " Please enter Landing Time (Enter an integer): ";
73. }
74. else break;
75. }
76. bst.treeInsert(landingTime, flightName);
77. //checking for the timeGap (K) constraint.
78. if(bst.treeSuccessor(bst.treeSearch(bst.root, landingTime)) != NULL)
79. if( abs(bst.treeSearch(bst.root, landingTime)->key - bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->key) < timeGap){
80. cout << "successor: " << bst.treeSearch(bst.root, landingTime)->key - bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->key << "\n";
81. cout << "\n The time gap between " << landingTime << " and flight ";
82. cout << bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->data << "(" << bst.treeSuccessor(bst.treeSearch(bst.root, landingTime))->key << ") is too small!\n";
83. cout << " Plane request denied!\n";
84. bst.treeDelete(bst.treeSearch(bst.root, landingTime));
85. break;
86. }
87. if(bst.treePredecessor(bst.treeSearch(bst.root, landingTime)) != NULL)
88. if( abs(bst.treeSearch(bst.root, landingTime)->key - bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->key) < timeGap){
89. cout << "predecessor: " << bst.treeSearch(bst.root, landingTime)->key - bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->key << "\n";;
90. cout << "\n The time gap between " << landingTime << " and flight";
91. cout << bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->data << "(" << bst.treePredecessor(bst.treeSearch(bst.root, landingTime))->key << ") is too small!\n";
92. cout << " Plane request denied!\n";
93. bst.treeDelete(bst.treeSearch(bst.root, landingTime));
94. break;
95. }
96. cout << "\n Plane " << bst.treeSearch(bst.root, landingTime)->data << " was added!\n";
97. break;
98. //withdraw landing request
99. //I was gonna do removal by searching landing time
100. //but thought that would be weird in a real setting so
101. //i overloaded treeSearch to work with inputted strings
102. case 'B':
103. cout << " Flight names and landing times:\n";
104. cout << DOTLINE;
105. bst.inOrderTreeWalk\_flightName(bst.root);
106. cout << " Please enter Landing time of the flight to remove: ";
107. while(true){
108. cin >> landingTime;
109. if(!checkCin()){
110. cout << " ERROR:: Invalid input!\n";
111. cout << " Please enter Landing Time (Enter an integer): ";
112. }
113. else break;
114. }
116. if(bst.treeSearch(bst.root, landingTime) == NULL){
117. cout << " ERROR:: Flight Not fould!";
118. break;
119. }
120. else {
121. cout << " Flight " << bst.treeSearch(bst.root, landingTime)->data;
122. bst.treeDelete(bst.treeSearch(bst.root, landingTime));
123. cout << " has been deleted.\n";
124. }
125. break;
127. case 'C':
128. cout << " Flight names and landing times:\n";
129. cout << DOTLINE;
130. bst.inOrderTreeWalk\_flightName(bst.root);
131. break;
133. case '?':
134. printMenu();
135. break;
136. case 'Q':
137. system("pause");
138. return 0;
139. break;
141. default:
142. cout << " ERROR:: Invalid input!\n";
143. cout << " Please try again. . .\n\n";
144. break;
145. }
146. }
147. }
148. Upload your programs for question 1 and question 2 to google drive and provide the links for your programs

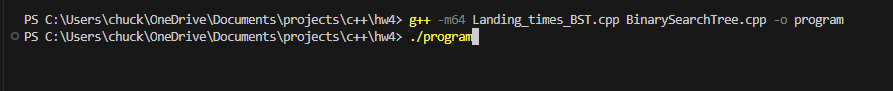
Link for the program for question1

* <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.



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!